INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 5, Issue 2)
Available online at: www.ijariit.com

Load rebalancing for distributed file systems in clouds

Harsh Vardhan Pawar
pharsh5720@gmail.com
SRM Institute of Science and Technology, Chennai, Tamil Nadu

Aditya Sinha
aditya.shine125@gmail.com
SRM Institute of Science and Technology, Chennai, Tamil Nadu

Richa Sharma
richasharma5912@gmail.com
SRM Institute of Science and Technology, Chennai, Tamil Nadu

S. Babeetha
babeethas14@gmail.com
SRM Institute of Science and Technology, Chennai, Tamil Nadu

ABSTRACT

The key building blocks for cloud computing applications are Distributed File Systems which are based on the Map-Reduce programming paradigm. Nodes simultaneously serve computing and storage functions in such file systems. A file is partitioned into a number of chunks allocated in distinct nodes in order so, over the nodes, MapReduce tasks can be performed in parallel. However, failure is the norm in a cloud computing environment. In the system, nodes may be either upgraded, or replaced, or added. Files can also be dynamically created, or deleted, and appended. Our algorithm is compared against a centralized approach in a production system and a competing distributed solution presented in the literature. The results indicate that our proposal is comparable with the existing centralized approach. It also considerably outperforms the prior distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead.

Keywords—*Chunk, DHT, MapReduce*

1. INTRODUCTION

The key building blocks for cloud computing applications are Distributed File Systems which are based on the Map-Reduce programming paradigm. Nodes simultaneously serve computing and storage functions in such file systems. A file is partitioned into a number of chunks allocated in distinct nodes in order so, over the nodes, MapReduce tasks can be performed in parallel. However, failure is the norm in a cloud computing environment. In the system, nodes may be either upgraded, or replaced, or added. Files can also be dynamically created, or deleted, and appended resulting in load imbalance in a distributed file system. Nowadays for chunk reallocation, Emerging distributed file systems in production systems strongly depend on a central node. This is clearly inadequate for a large-scale environment which is failure-prone. It is because the central load balancer is put under a considerable workload that is linearly scaled with the system size, results in becoming the performance bottleneck and the only point of failure. In this paper, we have tried to present a fully distributed load rebalancing algorithm in order to cope with the load imbalance problem.

Our algorithm is compared against a centralized approach in a production system and a competing distributed solution presented in the literature.

The results indicate that our proposal is comparable with the existing centralized approach. It also considerably outperforms the prior distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead.

2. EXISTING SYSTEM

Some use the concept of a virtual server. However:
1. It either ignores the heterogeneity of node capabilities.
2. Or transfer loads without considering proximity relationships between nodes.

3. IDENTIFICATION OF NEED

For chunk reallocation, Emerging distributed file systems in production systems strongly depend on a central node. This is clearly inadequate for a large-scale environment which is failure-prone. It is because the central load balancer is put under a considerable workload that is linearly scaled with the system size, results in becoming the performance bottleneck and the only point of failure

4. PROPOSED MODELING

When load balancing is performed. The load of each virtual server is stable over the timescale Load balancing is performed in a proximity-aware manner. It is done to minimize the overhead of load movement (bandwidth usage) allowing more efficient and fast load balancing.

4.1 Chunk creation

A file is partitioned into a number of chunks which are allocated in different nodes. It is done so that Map Reduce Tasks can be performed in parallel over the nodes. A load of a node is proportional to the number of file chunks the node possesses. The file chunks are not distributed as uniformly as possible among the nodes because the files in a cloud can be either arbitrarily created or deleted or appended, and nodes can be upgraded and added in the file system. Our objective is to allocate the chunks of files as uniformly as possible among the nodes. By doing that no node manages an excessive number of chunks.

4.2 DHT formulation

The storage nodes are like a network based on DHTs. Discovering a file chunk can simply refer to rapid key lookup in DHTs, given that a unique identifier is assigned to each file chunk. DHTs enable nodes to self-organize and repair. It also constantly offers lookup functionality in node dynamism which simplifies the system provision as well as the management. The chunk servers are organized as a DHT network in our proposal. In a typical DHTs if a node leaves, then its locally hosted chunks are reliably migrated to its successor. Or then it allocates the chunks whose IDs immediately precede the joining node from its successor to manage if a node joins.

4.3 Load balancing algorithm

In our proposed calculation, each lump server hub (node) I first gauge whether it is under stacked (light) or over-burden (overwhelming) without worldwide information. A hub is light if the quantity of lumps it has is littler than the limit. In particular, every hub contacts various arbitrarily chosen hubs in the framework and constructs a vector indicated by V. A vector comprises of passages, and every section contains the ID, arrange address and burden status of an arbitrarily chosen hub.

4.4 Replica management

In appropriated record frameworks (e.g., Google GFS and Hadoop HDFS), a steady number of limitations for each document piece are kept up in particular hubs to improve document accessibility concerning hub disappointments and flights. Our present burden adjusting calculation does not treat copies unmistakably. It is far-fetched that at least two reproductions are put in an indistinguishable hub in light of the irregular idea of our heap rebalancing calculation. All the more explicitly, each under stacked hub tests various hubs, each chose with a likelihood of 1/n, to share their heaps (where n is the absolute number of capacity hubs).

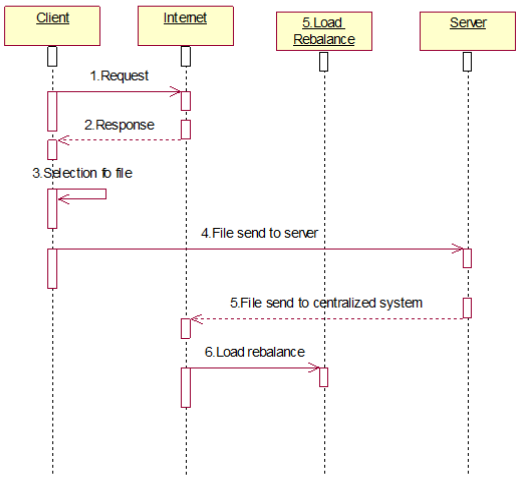

Fig. 1: Sequential diagram

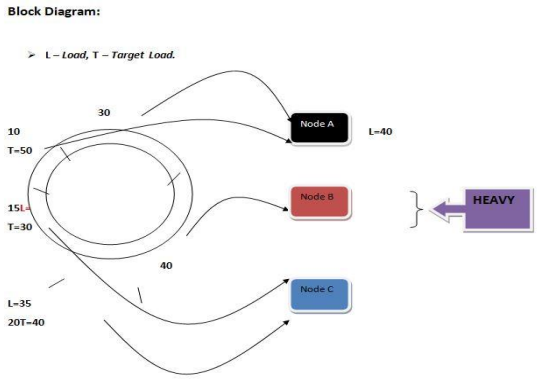

Fig. 2: Architecture diagram

4.5 MapReduce: simplified data processing on large clusters

MapReduce is a programming model and a related execution for preparing and creating huge informational collections. Clients indicate a guide work that forms a key/esteem pair to produce a lot of middle of the road key/esteem sets, and a diminish work that combines every single halfway esteem related with a similar moderate key. Numerous true assignments are expressible in this model, has appeared in the paper.

Projects written in this practical style are consequently parallelized and executed on an expansive group of ware machines. The run-time framework deals with the subtleties of apportioning the information, booking the program's execution over a lot of machines, taking care of machine disappointments, and dealing with the required between machine correspondence. This permits developers with no involvement with parallel and appropriated frameworks to effortlessly use the assets of a vast dispersed framework.

5. FUTURE WORK

There are some more features that could be added to this project to improve the efficiency of the project and provide better accuracy. As, we have worked with labeled data by following a supervised learning algorithm, in future we would Add more disorders and functionality to the application. Work on the real-life dataset. To develop a messaging system that informs the customer if information is out of reach.

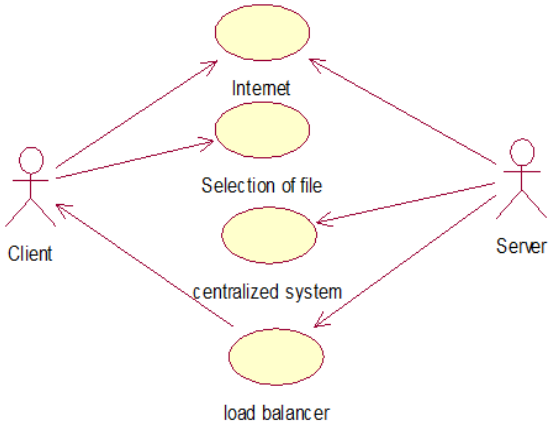

Fig. 3: Use case diagram

5.1 Operational feasibility

This proposed framework can undoubtedly be executed, as this depends on JSP coding (JAVA) and HTML . The database made is with MySql server which is progressively secure and simple to deal with. The assets that are required to execute/introduce these are accessible. The individual of the

association as of now has enough presentation to PCs. So the task is operationally practical.

5.2 Economical feasibility

The monetary investigation is the most every now and again utilized a strategy for assessing the adequacy of another framework. All the more normally known cost/advantage investigation, the methodology is to decide the advantages and reserve funds that are normal from an applicant framework and contrast them and expenses. In the event that benefits exceed costs, at that point, the choice is made to structure and execute the framework. A business visionary should precisely gauge the expense versus benefits before making a move. This framework is all the more monetarily attainable which survey the mind limit with a brisk and online test. So it is monetarily a decent venture.

6. RESULTS AND DISCUSSION

Our first protocol balances the distribution of the key address space to nodes, which yields a load-balanced system when the DHT maps items "randomly" into the address space. To our knowledge, this yields the first P2P scheme simultaneously achieving O(logn) degree, O(logn) look-up cost, and constant-factor load balance (previous schemes settled for any two of the three). Our second protocol aims to directly balance the distribution of items among the nodes.

7. ACKNOWLEDGEMENT

This paper was supported by Mrs Babeetha, our guide. We thank our faculties from Srmist who provided insight and expertise that greatly assisted the research, although they may not agree with all of the interpretations conclusions of this paper.

8. REFERENCES

[1] Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proc. 6th Symp. Operating System Design and Implementation *(OSDI'04)*, Dec. 2004, pp. 137–150.
[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in Proc. 19th ACM Symp. Operating Systems Principles (SOSP'03), Oct. 2003, pp. 29–43.
[3] Hadoop Distributed File System, ttp://hadoop.apache.org /hdfs/.
[4] VMware, http://www.vmware.com/.
[5] Xen, http://www.xen.org/.
[6] Apache Hadoop, http://hadoop.apache.org/.

