DOI 10.29042/2018-3884-3887

Helix Vol. 8(5): 3884- 3887

Statistical Approach for Load Distribution in Decentralized Cloud Computing

[*1]Chandu Vaidya, [2]Aatur Nampalliwar, [3]Krunal Nampalliwar, [4]Rishabh Thakkar, [5]Sagar Bhagat

[1, 2, 3, 4, 5]Rajiv Gandhi College of Engineering Research, Wanadongri, Hingna Road, Nagpur

Email:chandu.nyss@gmail.com,aturnampalliwar@gmail.com,krunalnampalliwar@gmail.com,rishtkr@gmail.com,sagar.bhagat27@gmail.com

Received: 09th July 2018, Accepted: 14th August 2018, Published: 31st August 2018

Abstract

In this era of developing technologies, one of the most promising is cloud computing that has been functioning since years and used by individuals and large enterprises to provide different kind of services to the world. Cloud computing is attractive to business owners as it eliminates the requirement for users to plan ahead for pro-visioning hardware and allows enterprises to start from the small scale and increase resources only when there is a rise in service demand. One of the trends in cloud computing is the use of decentralized cloud, where resources are shared between multiple servers instead of one central server. The use of decentralized cloud can reduce task complexity and work efficiency per server by utilizing small amount of resources from various servers. With this emerging trend in decentralized cloud, various methodologies were introduced to implement it. One of the most important methodology that plays a vital role in decentralized cloud is Load distribution. Load distribution can be achieved in various ways and hence different approaches can lead to better ways to distribute load in cloud servers. So by researching various methodologies used in cloud computing, a similar but new load distribution algorithm can be implemented which will be based on current statistics of the server nodes by comparing statistics the resultant best node can be chosen to dispatch the file which is to be uploaded.

Keywords: Cloud Computing, Load Distribution, Resource Parameters, Statistical Comparison.

Introduction

The main purpose of this system is to deal with the decentralized cloud architecture where all the files are distributed to different resource nodes which are connected in the network. When different files are sent to resource nodes it may happen that a particular node is provided with abundant files more than it can handle, this will lead to loss of data and failure of system. There is a need of a mechanism which can decide which resource node should get the file and what resource handling capability each of the resource node has. Different resource nodes have their different resource values which defines what capability a resource node has.

In this proposed system, the cloud storage uses decentralized architecture which works in a distributed manner, that's why it is designed in peer to peer and a prototype system is developed. For example, a particular resource node may have the resource parameters as CPU usage, RAM and space available on the node etc. So these different resource parameters can be studied so that the load coming on a single resource node can be balanced and distributed by certain algorithm, by statistically comparing these resource parameters the best node to send the data can be selected, which will result in successful transferring of data from user to different resource node and will also reduce the delay in transfer if it selects a resource node which has low CPU utilization and more available space.

Problem Statement

In decentralized Cloud Computing, the data which is available has to be sent to different resource nodes to achieve decentralized nature, due to which the problem of load distribution exist because different resource nodes will have different data handling capabilities, which affects the efficiency and resource utilization ratio of overall system. It leads to delay in performing tasks, a statistical comparison of different resource parameters can be done and seamless load management of the servers can be achieved, which will ensure an efficient and fair allocation of computer resources.

Aim

The proposed system aims to develop a mechanism through which various optimizations in load distribution can be achieved by considering statistics of various parameters.

Objective

To develop a Decentralized Cloud platform in which statistical comparison of different resource parameters is to be done to achieve seamless load distribution among various servers. To achieve seamless load distribution among various servers by statistically comparing the different resource parameters (disk space, latency, etc) and finding an appropriate load distribution method, which will give best results.

Proposed Work

Workflow of Uploading a File

In order to upload a file to the cloud system one has to upload it through the file upload form. Once the

user selects and uploads the file, certain actions take place to store it. To attain the decentralized nature the file undergoes several stages before getting fully stored in several resource nodes. The following stages are explained in *Figure 1*.

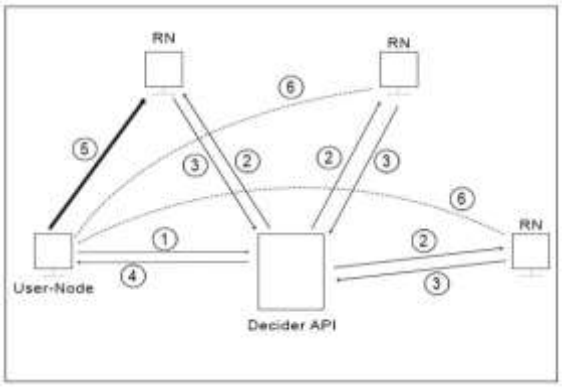

Figure 1: Flow Diagram of Uploading Module

Various Stages of File Uploading.

1) Splitting file

When a user issues a request of uploading a file the file undergoes pre-uploading stage. This pre-uploading stage consists of splitting of file into chunks. The client system i.e. the system by which the user uploads the file, provides the name of the file and the number of chunks to be split into.

The split mechanism works as follows:

Split (file, number of chunks)

1. Calculate file size
2. Calculate one chunk size (file size / number of chunks)
3. Open file
4. For each chunk
    a. Read file
    b. Write file into another file
5. Return chunk

From the above mechanism the file will be split into specified amounts of fixed sized chunks. Each chunk file name is suffixed with the part number it consists of. For example, file 'abc.pdf' of 20MB will be split into 'abc.pdf_1', 'abc.pdf_2', 'abc.pdf_3', 'abc.pdf_4' each of 5MB provided that the number of chunks to be split is 4.

2) Finding resource node

Once the file underwent pre-uploading stage and chunks are formed the client system requests the decider API to find address of resource node to which these chunked file will be uploaded to. The client system issues this request to decider API by providing the decider with current file name and chunk file size.

These two parameters are necessary for decider API to choose the best node for that particular chunk. This request is totally maintained by client system.

2) Broadcast request to all resource nodes

As soon as the Find resource request is received from Client PC, the Decider API sends a Broadcast message to all Resource nodes which are connected in the system. This request the resource node to share their current system statistics with the decider API.

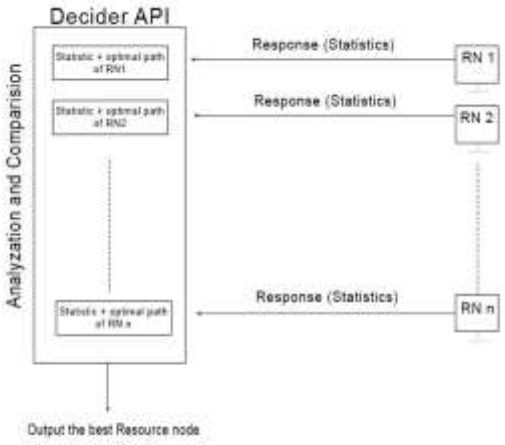

Figure 2: Architecture for Decider API

3) Response to broadcast with statistics

When the request packet is received from decider API, the resource node has to respond with the statistics of itself, these statistics includes - memory usage, CPU usage, disk space and latency. These statistics are calculated with using simple batch commands such as 'wmic' provided by the operating system. These batch commands are written in a batch file and are run on each resource nodes. These batch files are termed as Backend Connected Device Interface BCDI. This interface constantly updates the device specific statistical parameters into the system database so that the decider API can compute over it. This can be used as parameters to compare with another resource nodes and selecting the best one.

4) Return the best resource node

When the decider API receives the statistics from all the resource nodes, it will then calculate and analyzes the best suitable node for uploading a file chunk. The algorithm for selecting the best node, first checks whether the space required for the chunk file is available in resource nodes or not. Then it compares the latency of each resource nodes having the desired available space. The resource node with the smallest latency is selected as current best node for uploading a chunk file. Like-wise it calculates the same for all available chunks to be uploaded.

The algorithm of decider API works as follows:

1. List_latency = list of latency values of all resource nodes
2. List_storage = list of available storage value of all resource nodes
3. Decider (List_latency, List_storage, chunk size)
    1. Sort list_latency
    2. Key = list_latency[0]

3. Check list_storage [key] > chunk size ?
   a. If yes do: return address of key resource node.
   b. If no do : Decider (List_latency - List_latency[0], List_storage - List_storage[0], chunk size) until key count of List_latency = 1

This module returns the list of address of resource nodes back to client system.

5) Direct data transfer between selected nodes.
When the best node is selected, a peer to peer connection is established between the user node and the resource node to transfer the data. A FTP connection is established between the resource node and client system to transfer the chunk file. The FTP transfer makes the system secure from outsiders as only the sending and receiving participates in transfer of file. Once the file is transferred the connection is closed and the address of that resource node is logged into client database.

Workflow of downloading a file

The download module will work in these following stages illustrated in *Figure 3* below.

1) Find resource node

Each file a user upload has a unique identifier termed as file_id associated with it. When the user issues a request to download a previously uploaded file, the client request the decider API to retrieve chunks of the file from the particular resource node using the file_id.

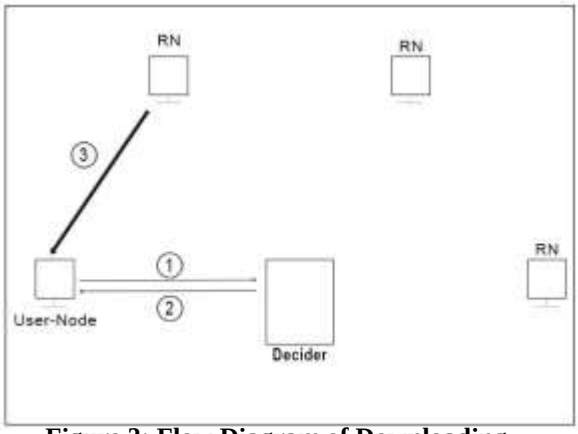

Figure 3: Flow Diagram of Downloading Module

2) Response with list of addresses

The decider API will then in turn will find the location of chunk file in the resource nodes connected to the backend. Once it finds all the chunk location it sends back the list of Addresses where the chunks are located to the client.

3) Direct Retrieval

The client will then issues the request to access the chunk file located at the resource system whose addresses are provided by decider API in previous stage. An FTP connection is established between the client system and resource node. Once the chunk file is received the FTP connection is terminated.

4) Merging Chunks

After all the chunks of the file are retrieved from above stages they undergo final stage. This final stage stitches all the chunks into one single file by removing the suffixes and passing the files into merging mechanism.

Pseudo Code

The pseudo code for implementing load balancing in decentralized cloud computing is explained below. The pseudo code consist of LOGIN,
UPLOAD, SPLIT, DECIDER, DOWNLOAD,MERGE functions to implement the corresponding modules explained above in this chapter.

```
IF LOGIN (username,
 Password)==SUCCESSFUL {
  UPLOAD (filename, file)
  {
   SPLIT ( file , no._of_chunks)
    FOR_EACH Chunk()
    {
     Select BEST_NODE =
     DECIDER(Chunk_size)
     PUSH Chunk to BEST_NODE
    }
  }
  Else
  {
   USER LOGIN FAILED
  }
}

DECIDER (Chunk_size)
{
 REQUEST STATISTICS from RESOURCE
NODES
CALCULATE LATENCY
SELECT MIN(LATENCY)
IF(STORAGE(MIN(LATENCY))>Chunk_size)
 {
Return(RESOURCE_NODE_ADDRESS)
 }
 else
 {
  Check for other RESOURCE NODES
 }
}

DOWNLOAD(file)
{
  REQUEST DECIDER for ADDRESS LIST
  Return All Chunk Files MERGE(All Chunk
  Files)
  Return(Merged file)
}
```


Results and Discussion

By implementing the modules discussed above, we achieved load distribution on various nodes present in a decentralized environment by comparing the statistics provided by them about various parameters. Statistical parameters like latency and storage space were used to determine the best node. Also, the collection of statistics was achieved by using console commands and compiling them into a batch script so that the process can be continuous.

When uploading a file, the Decider API is triggered and then it sends request packet to resource nodes to obtain the parameters and then chooses the best node based on latency and storage space of resource node to send the file. After choosing the best node, the file is split into a chunk and then sent to the chosen node, this process continues until all the chunks have been uploaded.

All chunks are assigned with a numerical suffix denoting the sequence number of that file, this helps in recognizing which part of the file the particular chunk belongs to. When the chunk is uploaded to a resource node, the address of the node is added to the user records so that it would be helpful to download that chunk again.

To retrieve back the file i.e. to download the file, the addresses of the chunks are recalled and downloaded to the user node via a peer to peer connection and then merged according to the sequence number they had been provided with. After merging the file, the whole file is obtained. Since the chunks are stored instead of files, if a node is attacked only the chunk will be obtained which will be of no use to the attacker.

References

[1] Jiyi Wu, Lingdi Ping, Xiaoping Ge, Ya Wang, Jianquing Fu, "Cloud storage as the Infrastructure of Cloud Computing", Intelligent Computing and Cognitive Informatics (ICICCI) IEEE, 2010, 10.1109/ICICCI.2010.119.

[2] Qi Zhang, Lu Cheng, Raouf Boutaba "*Cloud computing: state-of-the-art and research challenges",* Journal of Internet Services and Applications,2010,https://doi.org/10.1007/s13174-010-0007-6.

[3]Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, Ivona Brandic, "*Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility",* Journal of Future generation computer systems, 2009.

[4] Mohammad Sajid, Zahid Raza, "*Cloud Computing: Issues & Challenges",* International Conference on Cloud, Big Data and Trust, 2013.

[5] Ayob Sether, "*Cloud Computing Benefits",* SSRN, 2016.

[6] Tanupriya Choudhury , Vasudha Vashisht , Himanshu Srivastava, "*A Secure Decentralized Cloud Computing Environment over Peer to Peer*", International Journal of Computer Science and Mobile Computing, 2013.

[7] Nitin Agrawal Cristian,Ungureanu," *Decentralized cloud storage*", 2011. https://patents.google.com/patent/US20110238737A1/en?q=decentralized&q=cloud&q=storage.

[8] Geetika Tewari Lakshmanan, Yuri G. Rabinovich, Robert Jeffrey Schloss," *Decentralized load distribution in an event-driven system*", 2011. https://patents.google.com/patent/US20110047555A1/en?q=decentralized&q=cloud&q=storage.

[9] Saurabh Chandra, Venkat Sunder Raman Rangasamudram Komaleeswaran."*Decentralized cloud workflow*, 2016, https://patents.google.com/patent/US20160105487A1/en?q=decentralized&q=cloud&q=workflow&oq=decentralized+cloud+workflow.

[10] Mark S. Diggs, David E. Merry, Jr.," *Interface for enabling a host computer to retrieve device monitor data from a solid state storage subsystem",* 2014,https://patents.google.com/patent/US7962792B2/en

Dharmesh kashyap, Jaydeep Viradiya,"*A survey of various load balancing algorithms in cloud computing*", International Journal Of Scientific & Technology Research Volume 3, 2014, ISSN 2277-8616.

[11] Ratan Mishra and Anant Jaiswal.," *Ant Colony Optimization: A solution of Load Balancing in Cloud",* International Journal of Web & Semantic Technology, 2012.

[13]Shounak Chakraborty, Ajoy Kumar Khan," *A study of load distribution algorithms in distributed scheduling*", IJRET: International Journal of Research in Engineering and Technology, 2013.

[14]Sau-Ming Lau, Qin Lu, Kwong-Sak Leung, "*Adaptive load distribution algorithms for heterogeneous distributed systems with multiple task classes",* Journal of Parallel and Distributed Computing, 2006.

[15] Dinesh Babu L.D., P. Venkata Krishna, "*Honey bee behavior inspired load balancing of tasks in cloud computing environments",* Applied Soft Computing, 2013.

[16] Chandu Vaidya, Prashant Khobragade, Ashish Golghate "*Data Leakage Detection and Security in Cloud Computing*", Global Research and Development Journals, 2016.

[17] Chandu Vaidya, Prashant Khobragade, "*Data Security in Cloud Computing*", International Journal of Advanced Technology in Engineering and Science, 2015.

[18] Tejashri Khandve, Megha Talekar, Sheetal Dhiwar, Madhuri Patil, "*Security in Cloud Computing*", International Journal of Advanced Research in Computer Engineering & Technology, 2015.

[19] Rongzhi Wang, *"Research on data security technology based on cloud storage",* Procedia Engineering, 2017.

[20] Cong Wang, Qian Wang," *Towards secure and dependable services in cloud computing*", IEEE, 2015, 10.1109/TSC.2011.24.

