Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Science and Engineering

Elixir Comp. Sci. & Engg. 71 (2014) 25090-25092

ISSN: 2229-712X

Reducing movement cost and performing fast convergence in DFS using cloud

M.Mangayarkarasi and R.Shanthi
Department of CSE, Alpha College of Engineering, Chennai, T.N, India.

ARTICLE INFO
Article history:
Received: 16 February 2014;
Received in revised form:
6 June 2014;
Accepted: 20 June 2014;

Keywords
Cloud Computing,
Distributed Hash Table,
Chunk Servers,
Distributed File System,
Load Balancing Technique.

ABSTRACT
Distributed file systems (DFS) is one of the important building blocks for cloud computing environment which supports Map Reduce programming pattern where nodes at the same time serve both computing as well as the storage functions. A file is partitioned into variety of chunk units allotted in different nodes in order that the Map Reduce tasks can be carried out in parallel over the nodes. However, in cloud environment, files can be dynamically performs all the operation like creation, deletion, and modification. These consequences in load imbalance on the storage resources; that is, the different file modules are not distributed as consistently as possible among the nodes. A fully distributed load balancing algorithm is offered to manage with the load imbalance problem and thus it increases the overall performance of the system.

Introduction

Cloud Computing, uses the internetworking and central distant servers to maintain and to provide storage for various data and applications. Cloud computing allows users and various organization to make use of the applications without installation and provide access to their personal files at any computer through internet access. This technology allows user for much more well-organized computing by centralizing all data storage, data processing and bandwidth. The perfect example of cloud computing is Yahoo, Gmail, or Hotmail etc. Cloud computing mainly works on three major sections: "application/software" "storage" and "connectivity." Each sections serves a different purpose and offers different products for the organization and also the cloud users access their information from anywhere at any time around the world. Load balancing, is where the network of computers distribute their workloads uniformly across multiple-resources, such as computers, network links, CPU, etc. Load rebalancing goal is to optimize resource usage, maximize the throughput, minimize the response time, and avoid overload of the resources. The key things to consider while developing such algorithms are as follows: estimation of load, comparison of load, stability of different system, performance of system, interaction between the nodes, nature of work to be transferred, selecting of nodes and many other ones.

Related Work And Existing System

Distributed file systems (DFS) in clouds environment mainly rely on central nodes to manage all the metadata details about the files (i.e data about the data) in the file systems and as well to provide a sense of balance in the load of nodes based on that metadata details. In the Existing methodology, which uses a master/slave architecture which has various groups consists of a single NameNode, a master in the master/slave architecture server that manages all the file system namespace and controls access to files by the clients. With single NameNode, there are many number of DataNodes which maintain the storage that is present in the nodes that they run on. This master/slave architecture represents a file system namespace and allows client information to be stored in files. The NameNode is responsible for executing and performing various file system namespace operations like fileopen, fileclose, and filerename in the directories. The NameNode performs the mapping of various blocks to DataNodes. The DataNodes also perform block create, delete, and replicate the files upon instruction that is provides by the NameNode. The Metadata information performs consistently and synchronously updating all the copies of the files which may corrupt the rate of namespace transactions per second. When the NameNode resumes, it selects the latest and most recent consistent to use. If the NameNode system gets corrupted and failed means no automatic resuming is possible and failover of the NameNode application to another node is not carried out, therefore manual intervention is necessary.

The centralized methodology makes the design and implementation of a distributed file system so simple and ease. When the large number of storage nodes, files and the accesses to files increases very frequently;the central nodes become a huge bottleneck, as they are not capable to perform the large number of file accesses due to user applications. Thus, depending on the central nodes to attempt the load imbalance problem make worse their heavy loads.

Proposed System

In the proposed methodology, the rebalancing of load is studied using the DFS in the cloud environment which is scalable and dynamic in nature. (The terms “rebalance” and “balance” is indistinguishable). Such a large-scale cloud has hundreds or thousands of resources and applications to be processed (and may reach tens of thousands in the future).The main goal is that the traffic of the network need to be reduced as much as possible, thus also to increase the bandwidth and the access of the files or applications from DFS more efficiently. Our proposal heavily depends on the node arrival and departure operations to migrate file modules among nodes. Node A goes into the system is partitioned off into variety of fixed-size modules, and every module contains a distinctive module handle (or module symbol) named with a globally performed hash perform like SHA .The hash perform returns a novel identifier for a given files pathname string and a module index.

Tele:
E-mail addresses: mmangai1191@yahoo.com

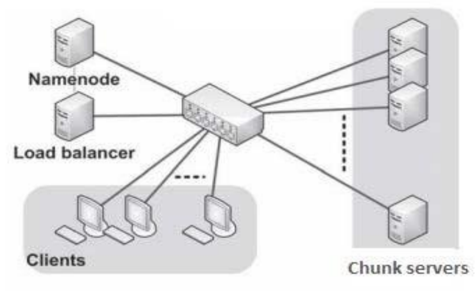

Fig 1: Implementation of data distribution

A node is light if the number of modules it hosts is smaller than the threshold as well as, a heavy node manages the number of modules greater than threshold. A large-scale distributed file system is in a load-balanced state if each module server hosts no more than A modules. In our proposed technique, each module server node I first calculate whether it is under loaded (light) node or overloaded (heavy) node without any information about the global knowledge. This process repeats and continues until all the heavy nodes in the system become light nodes. In Proposed system, file downloading or uploading with the help of the centralized system. Centralized system will be sharing the file (uploading and downloading). First of all we are going to notice the lightest node to require the set of modules from heaviest node. Thus we will do the method while not failure. Load equalization may be a technique to distribute employment across several computers or network to realize most utilization of resources economical output, reducing latency, and take away overload. During this project we have a tendency to use Load rebalancing formula. Then identical method is dead to unleash the additional load on following heaviest node within the system. Then we are going to once more notice the heaviest and lightest nodes, such a method repeats iteratively till there's not the heaviest.

System Description

DHT Formulation

DHT (Distributed Hash Table) is a primary technique used in Distributed File System. DHT is a hash table that requires key, values and a hash function. The hash function maps the key to a location where the values are stored. The chunk server is structured as a DHT network. To find where the file chunks are located in the chunk server, the DHT lookup operation is performed.

Creation of Chunks

Through chunk server, we try to make the file into various modules and distribute them as uniformly as possible among all the resources or chunk servers so that, no chunk servers handle an excess number of chunk of files or modules of files. The file is broken into number of modules and chunks which are placed in various chunk servers so that the map reduce task can be carried out simultaneously in all the chunk servers. A hash function such as SHA1 is used for each chunk to provide a unique ID for each and individual chunks, example the ID for second and fifth module of files“/user/mm/java.log” are SHA1(/user/mm/java.log, 1) and SHA1(/user/mm/java.log, 4). We can get the space for chunks from GoogleAppEngine, which makes efficient storage node for the real time applications.

Load Balancing Algorithm

Each chunk server approximately calculates whether the server is lightly loaded or heavily loaded. Load balancing technique distributes the workload across many chunk server to attain maximum throughput, maximum resource utilization. In this algorithm, each chunk server implements the gossip- based aggregation protocol which collects the status of the load by communicating with the nearby chunk servers.

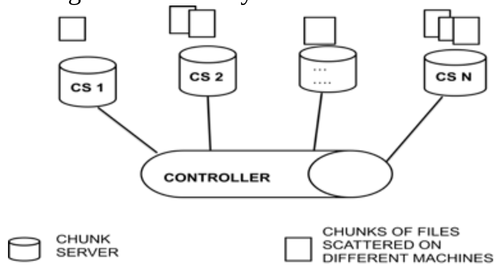

Fig 2: Files Splitted in Chunk Server

Manage Replicas

Replication is well known approach to attain high levels of availability and minimal access times for distributed environment. In DFS, a number of replicas for each file chunk are retained in distinct chunk servers to improve file availability with respect to node failures. The light weighted node will manage the replicas. In more particular, each light-weighted node makes sections of a number of nodes, each selected by means 1/n probability. So that sharing the load makes more efficient manner.

Experimental Result

The proposed methodology clearly performs better than the existing load balancer. When the NameNode is loaded heavily (i.e.,small N’s), our proposed techniques efficiently performs much better than the existing load balancer. For example, if N=1%, the load balancer takes about 70 mins to balance DataNodes load. By distinguish, our proposal takes nearly 25 mins in the case of N=1%. Specifically, in contrast the existing load balancer, our proposal is self-sufficient of the load of the NameNode. Our proposed techniques balance the load more efficiently and perform successful load balancing with fast convergence.

Conclusion

The load-balancing algorithm to deal with the load in large-scale, dynamic, and distributed file system in clouds has been offered in this paper. Our proposal strives to balance the loads of nodes and reduce the demanded movement cost as much as possible, while taking advantage of physical network locality and node heterogeneity. Our load-balancing algorithm exhibits a fast convergence rate. The fusion workloads stress test the load-balancing algorithms by creating a few storage nodes that are heavily loaded. The computer simulation results are encouraging, indicating that our proposed algorithm performs very well.

Reference

[1] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Proc. Sixth Symp. Operating System Design and Implementation (OSDI ’04), pp. 137-150, Dec. 2004.org/, 2012.

[2] K. McKusick and S. Quinlan, “GFS: Evolution on Fast-Forward,”Comm. ACM, vol. 53, no. 3, pp. 42-49, Jan. 2010.

[3] 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.

[4] A. Rowstron and P. Druschel, “Pastry: Scalable, Distributed Object Location and Routing for Large- Scale Peer-to-Peer Systems,” Proc. IFIP/ACM Int’l Conf. Distributed Systems Platforms Heidelberg, pp. 161-172, Nov. 2001.

[5] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, “Load Balancing in Structured P2P Systems,” Proc.

Second Int’l Workshop Peer-to-Peer Systems (IPTPS ’02), pp. 68-79, Feb. 2003.
[6] 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.
[7] 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.
[8] 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.
[9] G.S. Manku, “Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables, ”Proc. 23rd ACM Symp. Principles Distributed Computing197-205,July2009.