International Journal of Scientific Engineering and Technology Research

ISSN 2319-8885
Vol.03,Issue.20
September-2014,
Pages:4304-4307

www.semargroup.org,
www.ijsetr.com

Load Rebalancing for Distributed File System in Clouds

GALWAH TALAL SAMI[1], DR.B.PADMAJA RANI[2]

[1]PG Scholar, Dept of CSE, JNTUH College of Engineering, Hyderabad, India, E-mail: galwatalal@gmail.com.
[2]Prof, Dept of CSE, JNTUH College of Engineering, Hyderabad, India.

Abstract: 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. However, in a cloud 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. Emerging distributed file systems in production systems strongly depend on a central node for chunk reallocation. This dependence is clearly inadequate in a large-scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size, and may thus become the performance bottleneck and the single point of failure. In this project, a fully distributed load rebalancing algorithm is presented 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 simulation results indicate that our proposal is comparable with the existing centralized approach and considerably outperforms the prior distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead. The performance of our proposal implemented in the Hadoop distributed file system is further investigated in a cluster environment.

Keywords: Load Rebalancing, Cost, Cloud Computing.

I. INTRODUCTION

Cloud Computing (or cloud for short) is a compelling technology. In clouds, clients can dynamically allocate their resources on-demand without sophisticated deployment and management of resources. Key enabling technologies for clouds include the MapReduce programming paradigm[1] , distributed file systems virtualization and so forth[2][3]. These techniques emphasize scalability, so clouds can be large in scale, and comprising entities can arbitrarily fail and join while maintaining system reliability. 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 word count 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., chunk servers). Each storage node (or node for short) then calculates the frequency of each unique word by scanning and parsing its local file chunks. In a distributed file system, the load of a node is typically proportional to the number of file chunks the node possesses[3] . Because the files in a cloud can be arbitrarily created, deleted, and appended, and nodes can be up-graded, replaced and added in the file system, the file chunks are not distributed as uniformly as possible among the nodes.

In this paper, 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” are interchangeable in this project.) 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 to normal applications. Moreover, as failure is the norm, nodes are newly added to sustain the overall system performance, resulting in the heterogeneity of nodes. Exploiting capable nodes to improve the system performance is, thus, demanded.

II. LOAD REBALANCING PROBLEM

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 algorithm, each module server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. This process repeats 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.

The load equalization service is sometimes provided by dedicated code or hardware, like a multilayer switch or name server. 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. We consider a large-scale distributed file system consisting of a set of chunk servers V in a cloud, where the cardinality of V is $|V| = n$. Typically, n can be 1,000, 10,000, or more. In the system, a number of files are stored in the n chunk servers. First, let us denote the set of files as F. Each file $F \in f$ is partitioned into a number of disjointed, fixed size chunks denoted by C_f . For example, each chunk has the same size, 64 Mbytes, in Hadoop HDFS [3]. Second, the load of a chunk server is proportional to the number of chunks hosted by the server [3]. Third, node failure is the norm in such a distributed system, and the chunk servers may be upgraded, replaced and added in the system.

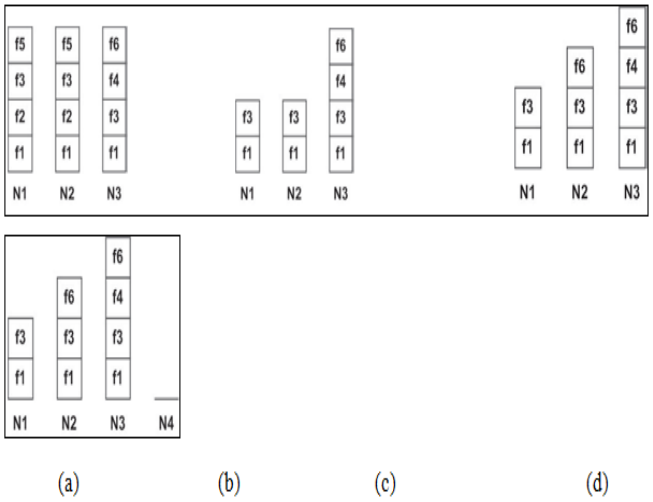

Fig.1. An example illustrates the load rebalancing problem, where (a) an initial distribution of chunks of six files f1, f2, f3, f4, f5, and f6 in three nodes N1, N2, and N3, (b) files f2 and f5 are deleted, (c) f6 is appended, and (d) node N4 joins. The nodes in (b), (c), and (d) are in a load-imbalanced state.

Finally, the files in F may be arbitrarily created, deleted, and appended. The next effect results in file chunks not being uniformly distributed to the chunk servers. Fig.1. illustrates an example of the load rebalancing problem with the assumption that the chunk servers.

Let $\mathcal{A}$ be the ideal number of Chunks that any Chunk server $i \in V$ is required to manage in any system-wide load-balanced state that is,

$$\mathcal{A} = \frac{\sum_{f \epsilon F} |C_f|}{n} \quad \textbf{(1)}$$

Then, our load rebalancing algorithm aims to minimize the load imbalance factor in each chunk server i as follows:

$$\|L_i - \mathcal{A}\| \quad \textbf{(2)}$$

where Li denotes the load of node i (i.e., the number of file chunks hosted by i) and $\|.\|$represents the absolute value function. Note that "chunk servers" and "nodes" are inter changeable in this project.

Theorem 1: The load rebalancing problem is NP-hard. **Proof:** By restriction, an instance of the decision version of the load rebalancing problem is the knapsack problem [16]. That is, consider any node $i \in V$. i seeks to store a subset of the file chunks in F such that the number of chunks hosted by i is not more than A, and the "value" of the chunks hosted is at least$\emptyset$, which is defined as the inverse of the sum of the movement cost caused by the migrated chunks. To simplify the discussion, we first assume a homogeneous environment, where migrating a file chunk between any two nodes takes a unit movement cost and each chunk server has the identical storage capacity. However, we will later deal with the practical considerations of node capacity heterogeneity and movement cost based on chunk migration in physical network locality. We implement the proposed system in this paper in four modules and when we implemented these modules The load of each virtual server is stable over the timescale when load balancing is performed. Load balancing is performed in proximity-aware manner, to minimize the overhead of load movement (bandwidth usage) and allow more efficient and fast load balancing.

III. MODULES

- Chunk creation
- DHT formulation
- Load balancing algorithm
- Replica management

A file is partitioned into a number of chunks allocated in distinct nodes so that Map Reduce Tasks can be performed in parallel over the nodes. The load of a node is typically proportional to the number of file chunks the node possesses. Because the files in a cloud can be arbitrarily created, deleted, and appended, and nodes can be upgraded, replaced and added in the file system, the file chunks are not distributed as uniformly as possible among the nodes. so we

allocate the chunks of files as uniformly as possible among the nodes such that no node manages an excessive number of chunks and after we partition the files we arrange the chunks files structured network based as distributed hash table (DHT) discovering a file chunk can simply refer to rapid key lookup in DHTs, given that a unique handle (or identifier) is assigned to each file chunk. DHTs enable nodes to self-organize and Repair while constantly offering lookup functionality in node dynamism, simplifying the system provision and management. The chunk servers in our proposal are organized as a DHT network. Typical DHTs guarantee that if a node leaves, then its locally hosted chunks are reliably migrated to its successor; if a node joins, then it allocates the chunks whose IDs immediately precede the joining node from its successor to manage

The hash functtion returns a unique identifier for a given files path name string and chunk index example : identifier of the first and third chunks of file ("user/jonh/tmp/a.log) are respictivly ("user/jonh/tmp/a.log ,0") and ("user/jonh/tmp/a.log ,2"). In our proposed algorithm, each chunk server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. A node is light if the number of chunks it hosts is smaller than the threshold. Load statuses of a sample of randomly selected nodes. Specifically, each node contacts a number of randomly selected nodes in the system and builds a vector denoted by V. A vector consists of entries, and each entry contains the ID, network address and load status of a randomly selected node. After that the replica management so we can replicate the uploaded file to server.

IV. CONCLUSION & FUTURE WORK

In this paper we eliminated the dependence on central node and balance the loads of nodes, we minimizing the movement cost as much as possible, while taking the advantage of physical network locality also exploits capable node to improve the overall system performance. In the absence of representative real workloads (i.e., the distributions of file chunks in a large scale storage system) in the public domain, we have investigated the performance of our proposal and compared it against competing algorithms through synthesized probabilistic distributions of file chunks. Emerging distributed file systems in production systems strongly depend on a central node for chunk reallocation and remarkably outperform the distributed algorithm in terms of load imbalance factor, movement cost. This dependence is clearly inadequate in a large-scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size, and may thus become the performance bottleneck and the single point of failure comparing with our algorithm; a fully distributed load rebalancing algorithm is presented to cope with the load imbalance problem. The load rebalancing problem in distributed file systems specialized for large-scale, dynamic and data-intensive clouds. Such a Large-scale cloud has thousands or tens of thousands of nodes in the future. We can also append some security to our data that uploaded and downloaded.

V. REFERENCES

[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.

[2] 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.

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

[4] VMware, http://www.vmware.com/, 2012.

[5] Xen, http://www.xen.org/, 2012.

[6] Apache Hadoop, http://hadoop.apache.org/, 2012.

[7] Hadoop Distributed File System "Rebalancing Blocks,"

[8] S. Mohammad, S. Breß, and E. Schallehn, "Cloud Data Management: A Short Overview and Comparison of Current Approaches," Proc. 24th GI-Workshop Foundations of Databases (Grundlagen von Datenbanken), 2012.

[9] Apache Hadoop, http://hadoop.apache.org/, 2013.

[10] P.Jamuna and R.Anand Kumar "Optimized Cloud Computing Technique To Simplify Load Balancing"International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11,November 2013.

[11] Kokilavani .K, Department Of Pervasive Computing Technology, Kings College Of Engineering, Punalkulam, Tamil nadu "Enhance load balancing algorithm for distributed file system in cloud" International Journal of Engineering and Innovative Technology (IJEIT) Volume 3, Issue 6, December 2013

[12] Hadoop Distributed File System "Rebalancing Blocks," http:// developer.yahoo.com/hadoop/tutorial/module2.html#rebalancing,2012

[13] ChahitaTanak, Rajesh Bharati "Load Balancing Algorithm for DHT Based Structured Peer to Peer System"International Journal of Emerging Technology and Advanced Engineering (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 3, Issue 1, January 2013)

[14] QuangHieu Vu, Member, IEEE, Beng Chin Ooi, Martin Rinard, and Kian-Lee Tan "Histogram-Based Global Load Balancing in Structured Peer-to-Peer Systems" IEEE transaction on knowledge and data engineering,vol. 21, no. 4, April2009.

[15] 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.

[16] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., 1979.

Author Profile:

Galwah Talal Sami, Al Mansour University Collage, B sc. Software Engineering on july 2006, Baghdad-Iraq, PG Scholar, Dept of CSE, JNTUH College of Engineering, Hyderabad, India,
E-mail: galwatalal@gmail.com.