International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

BALANCING BLOCKS FOR DISTRIBUTED FILE SYSTEM IN CLOUDS BY USING LOAD REBALANCING ALGORITHM

T.Sakthisri[1], S.Pallavi[2]
[1]*Anna University, Chennai, M.E Computer and Communication Engineering Final year, The Kavery Engineering College, Salem, India*
[2]*Assistant Professor/Computer and Communication Engineering, The Kavery Engineering College Salem, India*
E-mail id: sakthimecce@gmail.com, spallavicse@gmail.com

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 paper, 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

Keyword--**Load balance, Distributed file systems, Clouds**

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 Map Reduce programming paradigm [1], distributed file systems (e.g., [3], [4]), virtualization (e.g., [4], [5]), and so forth. These techniques emphasize scalability, so clouds (e.g., [6]) 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 Map Reduce 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 Map Reduce 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 this paper, the *load rebalancing problem* in distributed file systems specialized for large-scale, dynamic and data-intensive clouds. (The terms "rebalance" and "balance" is interchangeable in this paper.)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 [3], [4], resulting in the heterogeneity of nodes.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

Exploiting capable nodes to improve the system performance is thus demanded.

II. Our Proposal

The chunk servers in our proposal are organized as a DHT network; that is, each chunk server implements a DHT protocol such as Chord [18] or Pastry [19]. A file in the system is partitioned into a number of fixed-size chunks, and "each "chunk has a unique *chunk handle* (or chunk identifier) named with a globally known hash function such as *SHA1* [24]. The hash function returns a unique identifier for a given file's pathname string and a chunk index. For example, the identifiers of the first and third chunks of file "/user/tom/tmp/a.log" arerespectivelySHA1.

Each chunk server also has a unique ID. We represent the IDs of the chunk servers in V by 1n, 2n, 3n, · · · , nn; for short, denote the n chunk servers as 1, 2, 3, · · · , n. Unless otherwise clearly indicated, we denote the *successor* of chunk server i as chunk server i + 1 and the successor of chunk server n as chunk server 1. In a typical DHT, a chunk server i hosts the file chunks whose handles are within (i−1n , in], except for chunk server n, which manages the chunks whose handles are in (nn, 1n].To discover a file chunk, the DHT lookup operation is performed. In most DHTs, the average number of nodes visited for a lookup is O(log n) [18], [19] if each chunk server maintains log2 n *neighbors*, that is, nodes i + 2k mod n fork = 0, 1, 2, · · , log2 n − 1. Among the log2 n neighbors, the one i+20 is the successor of i. To look up a file with l chunks lookups are issued.

DHTs are used in our proposal for the following reasons:

A. The chunk servers self-configure and self-heal in our proposal because of their arrivals, departures, and failures, simplifying the system provisioning and management.
B. if a node leaves, then its locally hosted chunks are reliably migrated to its successor;
C. if a node joins, then it allocates the chunks whose IDs immediately precede the joining node from its successor to manage.

Our proposal heavily depends on the node arrival and departure operations to migrate file chunks among nodes.

2.1. Load rebalancing algorithm

2.1.2. OVERVIEW:

A large-scale distributed file system is in a *load-balanced state* if each chunk server hosts no more than A chunks. 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 of $(1 - \Delta L)$ A (where $0 \leq \Delta L < 1$).

2.1.2.1BASIC ALGORITHMS:

In the basic algorithm, each node implements the *gossip-based aggregation protocol* in to collect the 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. Using the gossip-based protocol, each node i exchanges its locally maintained vector with its neighbors until its vector has s entries. It then calculates the average load of the s nodes denoted by A_i and regards it as an estimation of A. The nodes perform our load rebalancing algorithm periodically, and they balance their loads and minimize the movement cost in a best-effort fashion.

2.1.2.2 EXPLOITIING PHYSICAL NETWORK LOCALITY

A DHT network is an overlay on the application level. The logical proximity abstraction derived from the DHT does not necessarily match the physical proximity information in reality. That means a message traveling between two neighbors in a DHT overlay may travel a long physical distance through several physical network links. In the load balancing algorithm, a light node i may rejoin as a successor of a remote heavy node j. Then, the requested chunks migrated from j to i need to traverse several physical network links, thus generating considerable network traffic and consuming significant network resources (i.e., the buffers in the switches on a communication path for transmitting a file chunk from a source node to a destination node). We improve our proposal by exploiting physical network locality. Basically, instead of collecting a single vector per algorithmic round, each light node i gathers NV vectors.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

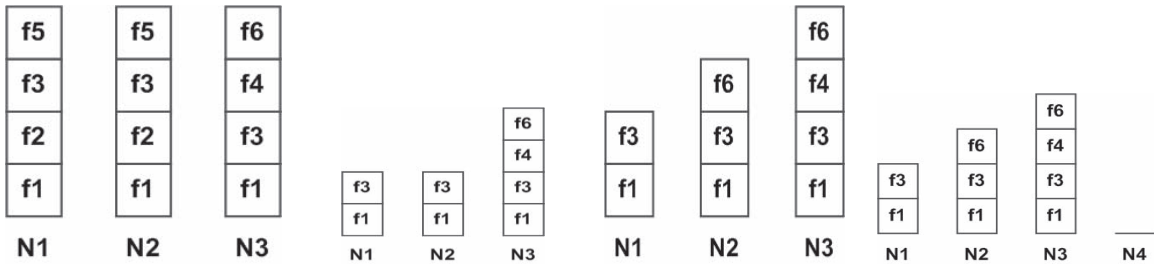

Fig. 1. An example illustrates the load rebalancing problem, where (a) an initial distribution of chunks of six files f1, f2, f3, f4, f5 and f6 in three nodes N1, N2 and N3, (b) files f2 and f5 are deleted, (c) f6 is appended, and (d) node N4 joins. The nodes in (b), (c) and (d) are in a load-imbalanced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

Each vector is built using the method introduced previously. From the NV vectors, the light node i seeks NV heavy nodes by invoking Algorithm 1 (i.e., SEEK) for each vector and then selects the physically closest heavy node based on the message round-trip delay.

2.1.2.3 TAKING ADVANTAGE OF NODE HETEROGENEITY

Nodes participating in the file system are possibly heterogeneous in terms of the numbers of file chunks that the nodes can accommodate. We assume that there is one bottleneck resource for optimization although a node's capacity in practice should be a function of computational power, network bandwidth and storage space.

In the distributed file system for Map Reduce-based applications, the load of a node is typically proportional to the number of file chunks the node possesses. Thus, the rationale of this design is to ensure that the number of file chunks managed by node i is proportional to its capacity.

2.1.2.4 MANAGING REPLICAS

In distributed file systems (e.g., Google GFS and Hadoop HDFS), a constant number of replicas for each file chunk are maintained in distinct nodes to improve file availability with respect to node failures and departures. Our load balancing algorithm does not treat replicas distinctly. It is unlikely that two or more replicas are placed in an identical node because of the random nature of our load rebalancing algorithm.

Given any file chunk, our proposal implements the *directory-based* scheme in to trace the locations of k replicas for the file chunk. Precisely, the file chunk is associated with k−1 pointers that keep track of k−1 randomly selected nodes storing the replicas.

III. IMPLEMENTATION

3.1 EXPERIMENTAL SETUP

I have implemented the proposal in Hadoop HDFS 0.21.0, and assessed our implementation against the load balancer in HDFS. The implementation is demonstrated through a small-scale cluster environment consisting of a single, dedicated name node and 25 data nodes, each with Ubuntu10.10]. Specifically, the name node is equipped with Intel Core 2 Duo E7400 processor and 3 Gbytes RAM. As the number of file chunks in our experimental environment is small, the RAM size of the name node is sufficient to cache the entire name node process and the metadata information, including the directories and the locations of file chunks.

In the experimental environment, a number of clients are established to issue requests to the name node. The requests include commands to create directories with randomly designated names, to remove directories arbitrarily chosen, etc.

Particularly, the size of a file chunk in the experiments is set to 16 Mbytes. Compared to each experimental run requiring 20 to 60 minutes, transferring these chunks takes no more than 328 seconds ≈ 5.5 minutes in case the network bandwidth is fully utilized. The initial placement of the 256 files chunks.

IV. EXPERIMENTAL RESULTS

Our proposal clearly outperforms the HDFS load balancer. When the name node is heavily loaded (i.e., small M's), our proposal remarkably performs better than the HDFS load balancer.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

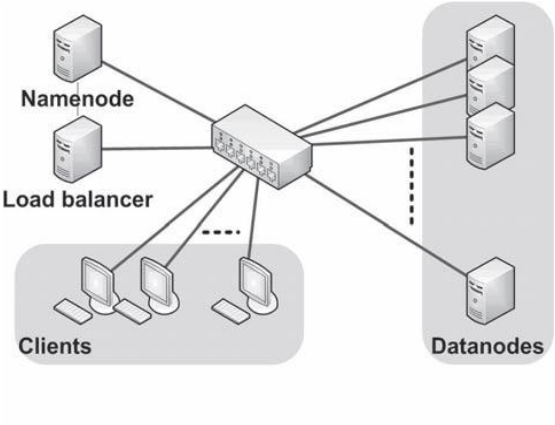

Fig. 2 shows the setup of the experimental environment

Our proposal clearly outperforms the HDFS load balancer. When the name node is heavily loaded (i.e., small M's), our proposal remarkably performs better than the HDFS load balancer. For example, if M = 1%, the HDFS load balancer takes approximately 60 minutes to balance the loads of data nodes. By contrast, our proposal takes nearly 20 minutes in the case of M= 1%. Specifically, unlike the HDFS load balancer, our proposal is independent of the load of the name node.

V. CONCLUSION

A novel load balancing algorithm to deal with the load rebalancing problem in large-scale, dynamic, and distributed file systems in clouds has been presented 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.

In the absence of representative real workloads 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. The synthesis 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. Our proposal is comparable to the centralized algorithm in the Hadoop HDFS production system and dramatically outperforms the competing distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead. Particularly, our load balancing algorithm exhibits a fast convergence rate. The efficiency and effectiveness of our design are further validated by analytical models and a real implementation with a small-scale cluster environment.

REFERENCES

[1] J. Dean and S. Ghemawat, "Map Reduce: 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, http://hadoop.apache.org/hdfs/.

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

[5] J. W. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," in Proc. 1st Int'l Workshop Peer-to-Peer Systems (IPTPS'03), Feb. 2003, pp. 80–87.

[6] S. Surana, B. Godfrey, K. Lakshminarayanan , R. Karp, and I. Stoica,"Load Balancing in Dynamic Structured P2P Systems," Performance Evaluation, vol. 63, no. 6, pp. 217–240, Mar. 2006.