IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 3, Ver. I (May-Jun. 2014), PP 46-49
www.iosrjournals.org

Load Rebalancing Algorithm for Distributed File System In Clouds

Prof. V.R.Ghule [1] , Miss. Gayatri B. Pawar[2], Mr. Abhijit Shinde[3]
1 *Pune university, M.E. Computer engineering, Professor in Computer department in Smt. Kashibai navale college of engineering, pune, India.*
[2,3] *Pune university, B.E. Computer Engineering Final year, Smt. Kashibai navale college of engineering, pune, India.*

Abstract: *The rapid growth of the World Wide Web has brought huge increase in traffic to popular web sites. As a consequence, end users often experience poor response time or denial of service. A cluster of multiple servers that behaves like a single host can be used to improve the throughput and alleviate the server bottlenecks. To achieve such a cluster, we need robust routing algorithms that provide scalability, effective load balancing and high availability in a constantly changing environment. Due to an ever-increasing diversity of workloads and cluster configurations, it is very difficult to propose a single algorithm that performs best under all conditions. In this paper, we review a number of proposed load balancing algorithms for clusters of web servers. We focus on an experimental analysis of the performance under a number of well-known load balancing algorithms. We study the performance of the algorithms through a simulation to evaluate their performance under different conditions and workloads. The results of our study are reported in the paper.*

Keywords: *Load rebalance, Distributed file system, clouds, Hash table, Algorithms.*

I. Introduction

Internet server programs supporting mission-critical applications such as Network Load Balancing servers (also called *hosts*) in a cluster communicate among themselves to provide key benefits, including:

- **Scalability.** Network Load Balancing scales the performance of a server-based program, such as a Web server, by distributing its client requests across multiple servers within the cluster. As traffic increases, additional servers can be added to the cluster, with up to 32 servers possible in any one cluster.
- **High availability.** Network Load Balancing provides high availability by automatically detecting the failure of a server and repartitioning client traffic among the remaining servers within ten seconds, while providing users with continuous service.

Network Load Balancing distributes IP traffic to multiple copies (or *instances*) of a TCP/IP service, such as a Web server, each running on a host within the cluster. Network Load Balancing transparently partitions the client requests among the hosts and lets the clients access the cluster using one or more "virtual" IP addresses. From the client's point of view, the cluster appears to be a single server that answers these client requests. As enterprise traffic increases, network administrators can simply plug another server into the cluster.

financial transactions, database access, corporate intranets, and other key functions must run 24 hours a day, seven days a week. And networks need the ability to scale performance to handle large volumes of client requests without creating unwanted delays. For these reasons, clustering is of wide interest to the enterprise. Clustering enables a group of independent servers to be managed as a single system for higher availability, easier manageability, and greater scalability.

Advanced Server and Data center Server operating systems include two clustering technologies designed for this purpose: Cluster service, which is intended primarily to provide failover support for critical line-of-business applications such as databases, messaging systems, and file/print services; and Network Load Balancing, which serves to balance incoming IP traffic among multi-node clusters. We will treat this latter technology in detail here.

Network Load Balancing provides scalability and high availability to enterprise-wide TCP/IP services, such as Web, Terminal Services, proxy, Virtual Private Networking (VPN), and streaming media services. Network Load Balancing brings special value to enterprises deploying TCP/IP services, such as e-commerce applications, that link clients with transaction applications and back-end databases.

II. Our Proposal

The chunkservers in our proposal are organized as a DHT network; that is, each chunkserver 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 path name string and a chunk index. For example, the identifiers of the first and third chunks of file "/user/tom/tmp/a.log"arerespectivelySHA1.
Each chunkserver also has a unique ID. We represent the IDs of the chunkservers in by1n,2n,3n,···,nn; for short, denote then chunkservers as 1,2,3,···,n. Unless otherwise clearly indicated, we denote the successor of chunkserver i as chunkserver i+1 and the successor of chunkserver n as chunkserver1. In a typical DHT, a chunkserver I hosts the file chunks whose handles are within (i−1n,in), except for chunkserver n, which manages the chunks whose handles are in (nn,1n). To discover a filechunk, 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 chunkserver maintains log2n *neighbors*, that is, nodes i+2 k mod n fork =0,1,2,··,log2n−1. Among the log2n neighbors, the one i+20 is the successor of i. To lookup a file with l chunks lookups are issued
DHT's are used in our proposal for the following reasons:
A. The chunkservers self-configure and self-heal in our proposal because of the 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.
A large-scaled is tribute file system is in a load balanced state if each chunkserver hosts no more than A chunks. In our proposed algorithm, each chunkserver 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−ΔL) A (where0≤Δ<1).

2.1 Load Balancing Algorithm
2.1 Basic Algorithm And Overview :

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 neighbours until its vector has s entries. It then calculates the average load of the s nodes denoted by A. 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 Physical Network Locality :
A DHT network is an over lay 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 neighbours in a DHT over lay may travel along physical distance through several physical network links. In the load balancing algorithm, a light node I may rejoin as a successor of are 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 as in vector per algorithm i around, each light node i gathers NV vectors.

Each vector is built using the method introduced previously. From the NV vectors, the light node I seeks NV heavy nodes by invoking Algorithm1 (i.e., SEEK) for each vector and then selects the physically closest heavy node based on the message round-trip delay.

2.1.3 Node Heterogeneity On Distributed File:

Nodes participating in the file system are possibly heterogeneous in terms of the numbers off i1 chunks that the nodes can accommodate. We assume that the reason bottleneck resource for optimizatio
although a node's capacity in practice should be function of computational power, network bandwidth and storage space. In the distributed file system for MapReduce based applications, the load of a node is typically proportional to the number off i chunks the node possesses. Thus, the rationale of this design is to ensure that the number off i chunks managed by node i is proportional to its capacity.
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 and nature of our load rebalancing algorithm. Given

any file chunk, our proposal implements the *directory based* scheme into trace the locations of replicas k 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 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.

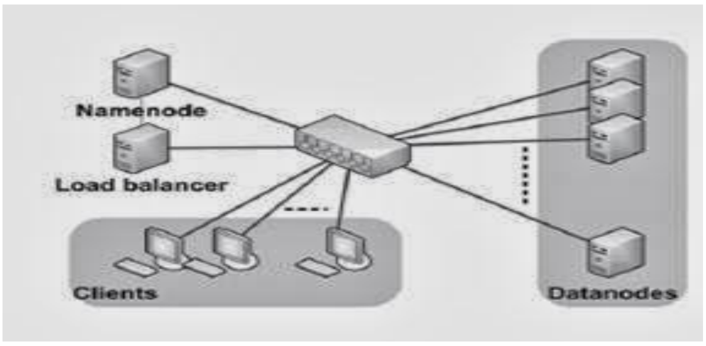

Fig 1. Shows the architecture of experimental setup .

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.

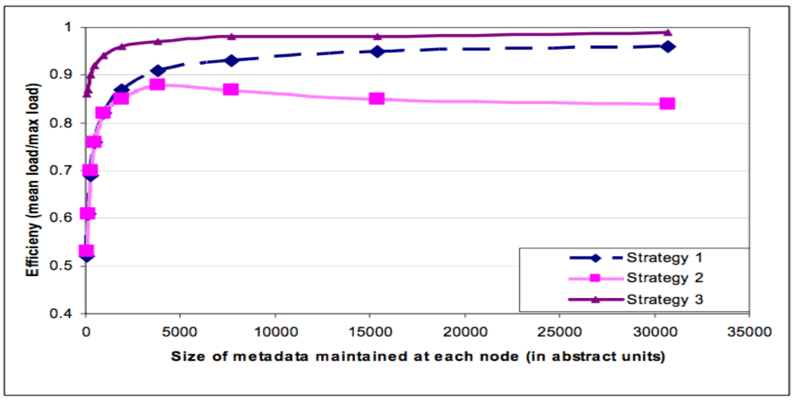

Fig 2 shows size of metadata according to the efficiency of the load.

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.

Without load balancing, some processors in the system are idle while other processors having many ready tasks waiting to run when the average processor utilization is just under 100%. Load balancing functions make use of the processor resource more efficiently and produce great improvement under this condition. When the processor utilization is higher than 100%, the processors are always busy. Periodic load balancing cannot bring improvement in overall performance, but can improve the performance for
some kinds of tasks at the expense of other kinds of tasks.

When the processor utilization is low, the choices of destination priority array on the destination processor do not influence the performance improvement from load balancing significantly. Load balancing leads to improvement in overall performance regardless. When the system is heavily loaded, the choices of the destination priority array have significant impact.

References:

[1]. L. M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," IEEETrans.SoftwareEng., vol. 11, no. 5, pp. 491–496, May 1985.

[2]. L. M. Ni, C.-W. Xu, and T. B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," IEEETrans.SoftwareEng., vol. 11, no. 10, pp. 1153–1161, Oct. 1985.

[3]. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in Proc.19thACMSymp.OperatingSystemsPrinciples(SOSP'03), Oct. 2003, pp. 29–43.

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

[5]. G. Copeland, W. Alexander, E. Boughter, and T. Keller, "Data Placement in Bubba," in Proc.ACMSIGMOD'88, June 1988, pp. 99–108.

[6]. P. Scheuermann, G. Weikum, and P. Zabback, "Data Partitioning and Load Balancing in Parallel Disk Systems," in Proc.7thInt'lConf.VeryLargeDataBases(VLDB'98), Feb. 1998, pp. 48–66.

[7]. S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," in ACM/IEEEConf.HighPerformanceNetworkingandComputing(SC'06), Nov. 2006.

[8]. F. B. Schmuck and R. L. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," in Proc.USENIXConf.FileandStorageTechnologies(FAST'02), Jan. 2002, pp. 231–244.

[9]. Lustre, http://www.lustre.org/.

[10]. Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Supporting Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems," IEEETrans.ParallelDistrib.Syst., vol. 22, no. 4, pp. 580–593, Apr. 2011.

[11]. B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable Performance of the Panasas Parallel File System," in Proc.6thUSENIXConf.FileandStorageTechnologies(FAST'08), Feb. 2008, pp. 17–33.

[12]. W. Ligon and R. B. Ross, BeowulfClusterComputingwithLinux. MIT Press, Nov. 2001, ch. PVFS: Parallel Virtual File System, pp. 391–430.

[13]. A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spy-glass: Fast, Scalable Metadata Search for Large-Scale Storage Systems," in Proc.7thUSENIXConf.FileandStorageTechnologies(FAST'09), Feb. 2009, pp. 153–166.

[14]. B. Fan, H. Lim, D. Andersen, and M. Kaminsky, "Small Cache, Big Effect: Provable Load Balancing for Randomly Partitioned Cluster Services," in Proc.2ndACMSymp.CloudComputing(SOCC'11), Oct. 2011.