International Journal of Advanced Research in

Computer Science Engineering and Information Technology

Volume: 2 Issue: 3 08-Apr-2014,ISSN_NO: 2321-3337

A SECURE HIGH PERFORMING CLOUD USING LOAD REBALANCING TECHNIQUE IN DISTRIBUTED FILE SYSTEM

Ajith Vishnu.P[1],*Dr.J.George Chellin Chandran*[2],*Prof.A.Usha Ruby,*[3]

PG Scholar, Dept of PG CSE, CSI College of Engg, Ketti, India[1].

Principal, CSI College of Engg, Ketti, India[2]

Research Scholar, Bharath University, Chennai, India[3]

ABSTRACT*- Map Reduce programming paradigm plays a vital role in the development of cloud computing application using the Distributed file system where nodes concurrently provide computing as well as storage functions. Initially a file is partitioned into number of chunks allocated into different nodes so that Map Reduce technique can be performed in the nodes. Since cloud computing is a dynamic environment upgrading, replacing and adding new nodes to the environment is a frequent concern. This confidence is obviously insufficient in a large-scale, failure-prone atmosphere since the central load balancer is put under significant workload that is linearly scaled with the structure of the system range, and may lead to a performance bottleneck the single point of failure. To overcome the failure in this paper, a fully distributed load rebalancing algorithm is presented to handle the load imbalance problem. The proposed algorithm is compared alongside a centralized approach in a production system and a competing distributed way out is available on hand in the literature. The simulation results point towards our proposal when compared with the existing centralized approach significantly outperforms the former distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead.*

Keywords- DHT, Centralized System, Load ImBalancing, Distributed System.

1. INTRODUCTION

The DHT plays a vital role in the existence of Distributed file system, the major functionality in the Distributed Hash Table is to balance the various nodes equally in the environment connected to the DHT. All DHTs take some effort in load balancing, generally this is done by randomizing the DHT address associated with each item with a "good enough" hash function and make each DHT node responsible for a balanced portion in the DHT address space. Similar to "Chord" it performs random hashing to nodes in a ring which means that each node will be responsible for a limited period of time when it is active using the ring address space whereas in random mapping of items only limited number of packets land in the ring interval owned by a single node. Existing DHT in Cloud Computing environment do not evenly partition the address spaces into which the key gets mapped into a larger portion of few machines. Therefore even if the keys are random and numerous some machines receive more than a fair share of information. To manage this problem, several DHTs use virtual nodes where each real machine pretends to be several distinct machines participating independently in the DHT protocol. The machine's load is thus determined by summing over the load of all virtual nodes, creating a tight attention of load by determining the average load. As a result, the Chord DHT based upon constant hash function will require numerous virtual environments to be operated for every node in this environment.

Every node will occasionally confirm its inactive virtual nodes, and may transfer to one of them a part of the distributed load in the system that has been updated. In view of the fact that only one virtual node is active; the genuine node need not pay the original Chord protocol's multiplicative enhancements made in space and bandwidth costs. Our elucidation allows a variety of nodes to move to various random addresses with the choice and also illustrate that we can carry out load balance in an arbitrary distribution of items, without using much cost in maintaining the load balance. Our proposal works through a kind of "work stealing" in which under loaded nodes migrate to portions of the address space engaged by too many items. The protocol is simple and practical, with all convolutions in its concert analysis. In this thesis, we are paying attention in studying the load rebalancing problem in distributed file systems focused for large-scale, dynamic and data-intensive clouds. Lastly by permitting nodes to choose random addresses in our item balancing protocol makes it easier for malicious nodes to interrupt the operation of the P2P network. It would be attention-grabbing to find counter-measures for this problem.

This paper is organized as follows. Section II: Related work, Section III. System Model, Section IV: Load balancing algorithm, Section V: Distributed files system, Section VI: Performance Evaluation and Section VII: Conclusion.

2. RELATED WORK

The challenge in load-balance is that it can fail in two ways. Firstly, the classic "random" partition of the address space amongst nodes is not completely balanced. A few nodes end up with a superior segment of the addresses and thus receive a larger portion of the randomly distributed data items. Secondly, a few applications may prevent the randomization of data items' addresses. For example, to maintain a range of searching in a database application the items may need to be placed in a specific order, or even at specific addresses, on the ring. In such cases, we may find the items unevenly distributed in address space, meaning that balancing the address space in the midst of nodes is not adequate to balancing the distribution of items among nodes. We give protocols to resolve both of the load balancing challenges just described.

2.1 Performance in a P2P System:

Our online load balancing algorithms are provoked by a new application domain for a variety of partitioning in peer-to-peer systems. P2P systems accumulate a relation over an outsized and vibrant set of nodes, and also maintain few queries in excess of this relation. A lot of modern systems, a technique known as Distributed Hash Tables (DHTs) are used in which hash partitioning ensure storage balance, and support point queries over the relation. There has been a significant recent attention in developing P2P systems that can support proficient range queries. In a P2P web cache, a node may request (pre-fetch) all pages with a specific URL prefix. It is well-known that hash partitioning is inefficient for answering such *ad hoc* range queries, motivating a search for new networks that allow range partitioning while still maintaining the storage space balance offered by normal DHTs.

2.2 Handling Dynamism in the Network:

The network splits the series of N_h to take over half the load of N_h, using the NBRADJUST operation. After this split, there may be NBRBALANCE violations stuck between two pairs of neighbors and in response, ADJUSTLOAD is executed, first at node N_h

and then at node N. It is easy to show (as in Lemma 3) that the resultant progression of NBRADJUST operations renovate all NBRBALANCE violations.

2.3 Node Departure:

Although in the network, each node manages data for a particular range. When the node fails or even when the node leaves the network, the data that is stored in that node becomes unavailable to the rest of the peers. P2P networks reunite this data loss in two ways: (a) Do nothing and permit the "owners" of the data deal with its availability. The owners will repeatedly survey the data to perceive its failure and re-insert the data into the network. (b) Preserve replicas of each node contends across multiple nodes. A Skip Net DHT organizes peers and data objects according to their lexicographic addresses in the form of a alternative of a probabilistic skip list. It supports logarithmic time range-based lookups and guarantees course vicinity. Mercury is more general than Skip Net since it supports range-based lookups on multiple-attributes. Our use of random sampling to approximate query selectivity constitutes a narrative donation towards implementing scalable multi-dimensional range queries. Load balancing is another important way in which Mercury uses Skip Net. While Skip Net incorporates a forced load-balancing mechanism, it is only constructive when part of a data name is hashed, in which case the part is unattainable for performing a range query. This implies that Skip Net supports load-balancing or range queries not both.

3. SYSTEM MODEL

3.1 Data Popularity

Unfortunately, in numerous applications, a particular range of standards might reveal a much superior recognition in terms of database insertions or queries than other ranges. This would cause the node accountable for the accepted range to become overloaded. One noticeable explanation is to agree on some way to partition the ranges in proportion to their reputation. As load pattern change, the system ought to shift nodes around as needed. We influence our approximate histograms to help execute load-balancing in Mercury. First, each node can use histograms to determine the normal load existing in the system, and so can determine if it is comparatively heavily or lightly loaded. Second, the histograms hold information with reference to which parts of the overlay are lightly loaded.

3.2 Load Balancing

We have exposed how to balance the address space, but sometimes this is not sufficient. Some applications, such as those aiming to support range-searching operations, necessitate specifying a particular, non-random mapping of items into the address space. In this section, we consider a vibrant protocol that aims to balance load for arbitrary item distributions. To do so, we must sacrifice the prior protocol's constraint of each node to a small number of virtual node locations—instead, each node is free to migrate anywhere. Our protocol is randomized, and relies on the fundamental P2P routing framework only able to contact "random" nodes in the system environment. The protocol is the following, to shape the performance of the environment, we need the concept of a *half-life* which is the time taken for half the items in the system to arrive or depart.

3.3 DHT Implementation

The storage nodes are prearranged as a network based on *distributed hash tables* (*DHTs*), DHTs facilitate nodes to self-organize and repair while frequently offering lookup functionality in node dynamism, simplifying the system provision and management. The chunk servers in our suggestion are structured as a DHT network. Typical DHTs pledge that if a node leaves, then its locally hosted chunks are dependably migrated to its successor; if a node joins, then it allocates the chunks whose IDs instantly precede the joining node from its successor to manage. Now we portray the application of this idea to DHTs. Let h0 be a universally agreed hash function that maps peers onto the ring. Similarly, let h_1,h_2,…h_d, be a series of universally agreed hash functions mapping items onto the ring. To insert an item x using d hash functions, a peer calculates $h_{1(x)}, h_{2(x)}, \dots .h_{d(x)}$. Then, d lookups are executed in parallel to and the peers $p_{1\,,}p_2$,…p_d responsible for these hash values, according to the mapping given by h0.

1. ***Chunk creation***

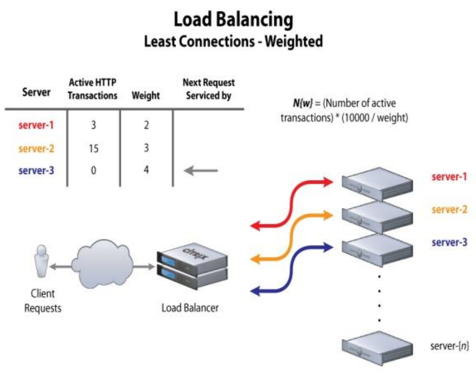
A file is partitioned into a number of chunks allocated in diverse nodes so that Map Reduce Tasks can be performed in parallel over the nodes. The load of a node is classically comparative to the number of file chunks the node possesses. Because the files in a cloud can be arbitrarily created, deleted, 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. Our objective is to allocate the chunks of files as uniformly as possible among the nodes such that no node manages a disproportionate number of chunks.

2. Replica Management

In distributed file a constant number of replicas for each file chunk are maintained in distinct nodes to improve file accessibility with respect to node failures and departures. Our current load balancing algorithm does not extravagance replicas distinctly. It is unlikely that two or more replicas are placed in an indistinguishable node because of the random nature of our load rebalancing algorithm. More particularly, each under loaded node samples a number of nodes, each selected with a probability of 1/n, to share their loads (where n is the total number of storage nodes).

4. LOAD BALANCING ALGORITHM

In our projected algorithm, each chunk server would firstly estimate whether the node is under loaded (light) or overloaded (heavy) without global knowledge. A node is said to be *light* if the number of chunks it hosts is smaller than the threshold value. The load status sample of randomly selected nodes is given below.


Load Balancing
Least Connections - Weighted
N(w) = (Number of active transactions) * (10000 / weight)
server-1
server-2
server-3
server-{n}
Client Requests
Load Balancer
citrix

Server	Active HTTP Transactions	Weight	Next Request Serviced by
server-1	3	2	
server-2	15	3	
server-3	0	4	←

Fig.1Load Balancing

Distinctively, each one of the node interacts to a number of arbitrarily selected nodes in the scheme 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 in a large-scale distributed file system. Fig. 1 shows the total number of messages generated by a load rebalancing algorithm; A is in a large-scale distributed file system.

4.1 Load-balanced state:

If each one of the chunk server do not host not more than 'Am' chunks. In our projected algorithm, each chunk server node 'I' firstly estimate whether it is under loaded (light) or overloaded (heavy) exclusive of global knowledge. 'L_j' of 'A' from 'j' is used to relieve the load of 'j' node 'j' may possibly still remain as the heaviest node in the system after it has migrated its load to node 'i'. In such cases, the current least-loaded node, say node 'I' departs and then rejoins the system as a successor of 'j'. That is the new node 'I' becomes node 'j+1', and j's original successor 'i' thus becomes node 'j + 2'. Such a process repeats iteratively until 'j' is no longer the heaviest. Then, the same process is executed to release the extra load on the next heaviest node in the system. This process repeats until all the heavy nodes in the system become light nodes.

4.2 Others:

We will offer a rigorous performance analysis for the effect of varying $\boldsymbol{n_V}$ in Appendix E. Specifically; we discuss the tradeoff between the value of $\boldsymbol{n_V}$ and the movement cost. A larger $n_V$introduces more overhead for message exchanges, but results in a smaller movement cost.

Procedure 1 ADJUSTLOAD (Node Ni) fOn Tuple Insertg

```
1: Let L($N_i$) = x 2 ($T_m$; $T_m$+1].
2: Let $N_j$ be the lighter loaded of $N_i$-1 and $N_i$+1.
3: **if** L($N_j$) _ $T_m$ +1 **thenf**DoNBRADJUSTg
4: Move tuples from $N_i$ to$N_j$to equalize load.
5: ADJUSTLOAD($N_j$)
6: ADJUSTLOAD($N_i$)
7: **else**
8: Find the least-loaded node Nk.
9: **if** L($N_k$) _ $T_m$+2**then** fDoREORDERg
10: Transfer all data from $N_k$ to N = $N_k$_1.
11: Transfer data from Ni to $N_k$, s.t. L($N_i$) = $d_x$=2e    and L($N_k$) = $b_x$=2c.
12: ADJUSTLOAD (N)
13: fRename nodes appropriately after REORDER.g
14: **end if**
15: **end if**
```

Example1: In the setting above, the maximum load is at most log (log n) = log (d+O) with high probability. Our proof (not included for reasons of space) uses the layered induction technique from the seminal work because of the variance in the arc length associated with each peer; we must modify the proof to take this into account. The standard layered induction uses the fact that if there is k bins that have load at least k,

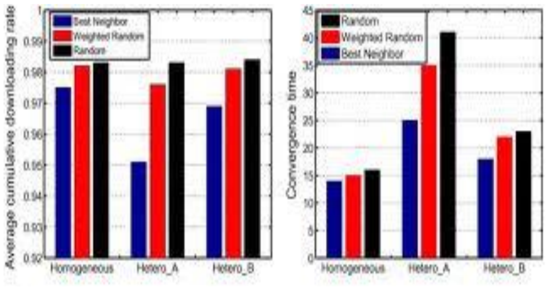
Example2:long distance links are constructed using the harmonic distribution on node-link distance. Value Link denotes the overlay when the harmonic distribution on value distance.

Given the capacities of nodes (denoted by {$\beta 1$, $\beta 2$, · · · , βn}), we enhance the basic algorithm in Section III-B2 as follows: each node i approximates the ideal number of file chunks that it needs to host in a load balanced state as follows:
$Ai = \gamma\beta i$,
Note that the performance of the Value Link overlay is representative of the performance of a plain DHT under the absence of hashing and in the presence of load balancing algorithms which preserve value contiguity.
As follows:-

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, "1");
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
result += ParseInt(v);
Emit(AsString(result));
```

5. DISTRIBUTED FILE SYSTEM

We have given more than a few provable proficient surveys on load balancing for distributed file's protocols and distributed data storage in P2P systems. More details and analysis can be found in a thesis. Our algorithms are simple, and easy to implement in the distributed files so an obvious next research step should be a practical evaluation of these schemes. In addition, several concrete open problems follow from our work. Firstly, this should be possible to further improve the consistent hashing scheme as discussed at the end of our range search data structure. Distributed hashing does not easily generalize to more than one order. For example (Fig.2) when storing music files, one might want to index them by both artist and song title, allowing lookups according to two orderings. Since our protocol rearranges the items according to the order, doing this for two orders at the same time seems to be difficult. A simple, but inelegant solution is to rearrange not the items themselves, but just store pointers to them on the nodes. This requires a more storage capacity and computing power.



Network Setting Network Setting

Fig.2 The average downloading rate and Convergence time

This makes it possible to maintain two or more orderings at a single time. Finally, by permitting nodes to decide subjective addresses in our item balancing protocol for distributed files makes it easier for malevolent nodes to interrupt the operation of the P2P network. It would be interesting to find counter-measures for this problem.

6. PERFORMANCE EVALUATION

We run a varying number of players. The players move through the virtual world according to a random way point model, with a motion time chosen consistently at random from seconds, a destination is chosen uniformly at random, and a speed chosen uniformly at random from (0, 360) pixels per second. The size of the game world is scaled according to the number of players. The dimensions are 640n -480n, where n is the number of players. All results are based on the average of 3 experiments, where each experiment lasts up to 60 seconds. The experiments include the bent of log n sized LRU cache long pointers. In the HDFS load balancer and 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.

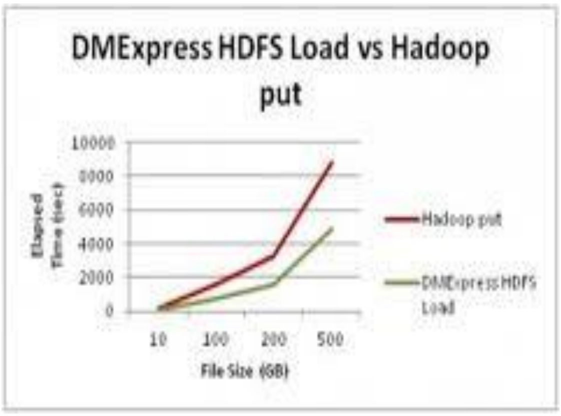


Fig.3 HDFS

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 in the name node. In particular, approximating the unlimited scenario is expensive, and the use of b(log2) nc virtual peers as proposed in introduces a large amount of topology maintenance trace but does not provide a very close approximation. Finally, we observe that while we are illustrating the most powerful instantiation of virtual peers, we are comparing it to the weakest choice model further improvements are available to us just by increasing d to 4.

7. CONCLUSION AND FUTURE ENHANCEMENT

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

The datum passing through the network is passed as the plain text so the encryption and decryption techniques can be included in the proposed system to increase the network security. The speed of the system might reduce due the increase in network traffic due to the various these cryptographic techniques can be reduces. Steps can be taken to include two or more cloud environment to the existing system.

REFERENCE

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek,F. Dabek, and H. Balakrishnan, "Chord: a Scalable Peer-to-Peer LookupProtocol for Internet Applications,"*IEEE/ACM Trans. Netw.*, vol. 11,no. 1, pp. 17–21, Feb. 2003.
[2] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed ObjectLocation and Routing for Large-Scale Peer-to-Peer Systems," *LNCS 2218*, pp. 161–172, Nov. 2001.
[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman,A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo:Amazon's Highly Available Key-value Store," in *Proc. 21st ACM Symp.Operating Systems Principles (SOSP'07)*, Oct. 2007, pp. 205–220.
[4] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "LoadBalancing in Structured P2P Systems," in *Proc. 2nd Int'l Workshop Peerto- Peer Systems (IPTPS'02)*, Feb. 2003, pp. 68–79.
[5] D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms forPeer-to-Peer Systems," in *Proc. 16th ACM Symp. Parallel Algorithms and Architectures (SPAA'04)*, June 2004, pp. 36–43.
[6] D. DeWitt, R. H. Gerber, G. Graefe, M. L. Heytens, K. B. Kumar,and M. Muralikrishna. Gamma -a high performance dataflow database. In *Proc. VLDB*, 1986.
[7] D. DeWitt and J. Gray. Parallel database systems: The future of highperformance database processing. *Communications of the ACM*,36(6), 1992.
[8] H. Feelifl, M. Kitsuregawa, and B. C. Ooi. A fast convergencetechnique for online heat-balancing of btree indexed database overshared-nothing parallel systems. In *Proc. DEXA*, 2000.
[9] P. Ganesan, M. Bawa, and H. Garcia-Molina.Online balancing ofrange-partitioned data with applications to p2p systems.Technical Report http://dbpubs.stanford.edu/pubs/2004-18, Stanford U., 2004.
[10] P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule themall: Multi-dimensional queries in p2p systems. In *WebDB*, 2004.
[11] S. Ghandeharizadeh and D. J. DeWitt.A performance analysis of alternativemulti-attribute declustering strategies. In *Proc. SIGMOD*,
1992.
[12] N. J. A. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman.Skipnet: A scalable overlay network with practical locality properties. In *Proc. USITS*, 2003.
[13] D. R. Karger and M. Ruhl. Simple efficient load-balancing algorithmsfor peer-to-peer systems.In *Proc. IPTPS*, 2004.