Volume : 3 | Issue : 5 | May 2014 • ISSN No 2277 - 8179

Research Paper

Reallocation of Load in Nodes of Distributed File Systems

Engineering

KEYWORDS : Cloud computing, Distributed file system, Load rebalancing, Distributed hash table

Thushara Joy T — PG Student, Dept. of Computer Science, NSS College of Engineering, Palakad, Kerala, India

Sruthy Manmadhan — Asst. Professor, Dept. of Computer Science, NSS College of Engineering, Palakad, Kerala, India

ABSTRACT

Distributed file systems are the collection of nodes which does both computation and storage functions in terms of MapReduce programming paradigms. Each node contains files that are uniformly allocated. The nodes does not retain as the same as initial allocation. They can be upgraded, replaced, added according to the workload. Also the files in the node can be created, deleted and appended. Hence the load in the DFS gets imbalanced. To overcome this situation and to make uniformity among load in the nodes, a distributed approach is proposed. Since the centralised approach is insufficient.

1. INTRODUCTION

Cloud computing allocates large amount of resources. DFS are used for data-intensive applications. Files may comprise of several chunks. Large chunks are stored by distinct storage nodes. A large set of tasks can be performed in parallel over the nodes. The nodes are capable of computation and storage. By load balance, it means that the nodes maintain the same number of chunks. The load of a node is proportional to the number of file chunks the node stores. The node manages file operations like create, delete and append. In cloud computing the load balance among storage nodes is difficult to manage. The nodes can be upgraded, replaced and added. This causes the nodes to be imbalanced. The existing centralised approach is unable to accommodate a large number of file accesses.

This paper studies the load rebalancing problem in distributed file systems for large-scale, dynamic and data-intensive clouds. The study finds a method to allocate the chunks of files as uniformly as possible among the nodes such that no node manages an excessive number of chunks. Also figures out to reduce the movement cost (or network traffic). The load reallocation in storage nodes are done with help of distributed hash tables (DHTs). The storage nodes organise as a ring: each links to its immediate successor. Each file chunk is assigned with a unique ID. The node hosts chunks with numerically closest IDs in clockwise direction. The node joins by allocating chunks from its immediate successor. The node leaves by releasing its chunks to its immediate successor.

2. THE PROPOSAL ARCHITECTURE

Suppose the total number of nodes is N and set of files be F. Each file f is split into a number of disjointed, equally sized chunks C_f. As mentioned before, the load of node is proportional to the number of chunks the node possess. The load imbalances causes the chunks not uniformly distributed to the nodes. The study finds a method to allocate the chunks of files as uniformly as possible among the nodes such that no node manages an excessive number of chunks. Also figures out to reduce the movement cost (or network traffic).

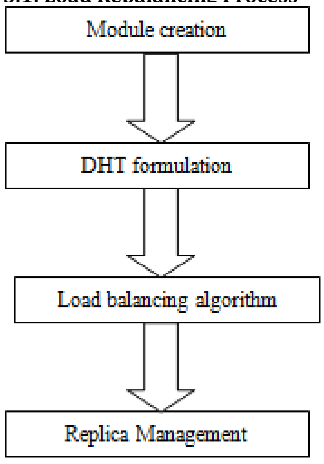
Reducing network traffic means reducing the number of chunks migrated to balance the loads of the nodes. Let M be the ideal number of chunks a node can manage in a load-balanced state. When the number of chunks in each file f is greater than M then the corresponding node is called heavy node. if the number of file chunks in F is much less than M then the node is called light node.

The nodes are structured as a network based on distributed hash tables (DHTs). The files comprises of several large chunks. Each file is partitioned into equally sized chunks. Each chunk is having unique chunk identifier named with a hash function. The hash function returns a unique identifier comprising file's pathname string and chunk index. Let the nodes be 1,2,3,...,n. Consider the successor of node i as node i+1 and the successor of node n as node 1if 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. Each node periodically informs the locally stored chunks information to gather the locations of chunks in the system.

There are four different modules in this system. They are depicted below.

- Module creation
- DHT formulation
- Load balancing algorithm
- Replica Management

Fig 3.1: Load Rebalancing Process



3. THE ALGORITHM

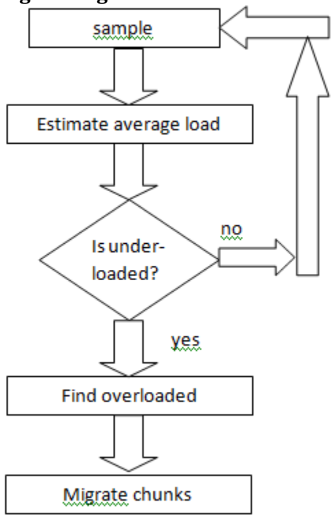
A distributed file system is said to be load-balanced when the node contains only M chunks. If a node i departs and rejoins as a successor of another node j, then we represent node i as node j+1, node j's original successor as node j+2, the successor of node j's original successor as node j+3, and so on. Foe each node i ϵ N, if node i is light, then it seeks a heavy node and takes over at most M chunks from the heavy node.

To release node j's load, node i requests minimum chunks from j. That is, node i requests M chunks from the heaviest nodej if j's load exceeds the threshold level. Node j still remains as heaviest node. So the lightest node i's depart and rejoins as j's successor (j+1). This repeats until j is no longer the heaviest. Hence all the heavy nodes in the system become light nodes. This can be depicted as follows:

- Interacting with the nodes to gather information regarding light and heavy nodes, including the locations of chunks

- Pair the light and heavy nodes (both selected randomly)
- Chunks migration
- Node notification
- Repeat till no progress, i.e. the reallocation terminates.

Fig 3.2: Algorithmic flow



4. CONCLUSION

These proposals balance the load of nodes and reduce the movement cost as much as possible. Using few storage nodes the proposal is tested and it outperforms the existing centralised approach. Here Hadoop HDFS is taken as the case study. A fully distributed load rebalancing algorithm is proposed with distributed hash tables (DHTs). This is a competing distributed solution.

REFERENCE

[1] Paul Krzyzanowski " Distributed File Systems", Rutgers University, October 28, 2012 | | [2] H.-C. Hsiao and C.-W. Chang, "A Symmetric Load Balancing Algorithm with Performance Guarantees for Distributed Hash Tables," IEEE Trans. Computers, 2012, http://doi.ieeecomputersociety.org/10.1109/TC.2012.13. | | [3]H.-C. Hsiao, H. Liao, S.-S. Chen, and K.-C. Huang, "Load Balance with Imperfect Information in Structured Peer-to-Peer Systems," IEEE Trans. Parallel Distrib. Syst., vol. 22, no. 4, pp. 634–649, Apr. 2011. | | [4] Shvachko K, Hairong Kuang, Radia S, Chansler R, "The Hadoop Distributed File System" Mass Storage Systems and Technologies (MSST), 2010 IEEE26th Symposium on Digital Object Identifier: 10.1109/MSST.2010.5496972, pp. 1 - 10 ,2010 | | [5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," COMMUNICATIONS OF THE ACM, Vol. 51, No 1, pp. 107-113, January 2008. | | [6] G Aggarwal, R Motwani, A Zhu" The load rebalancing problem" - Journal of Algorithms - Elsevier, 2006. | | [7]M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-Based Aggregation in Large Dynamic Networks," ACM Trans. Computer Systems, vol. 23, no. 3, pp. 219-252, Aug. 2005. | | [8] 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. | | [9] 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. | | [10] 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. | [11] 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. |