



A Short Survey of Key Distribution Techniques for Namespace Load Balancing

Ravneet Kaur¹, Shalini Batra²
¹Research Scholar, ² Assistant Professor
Computer Science and Engineering
Thapar University, Patiala
India

Abstract: With the enormous growth of volume of data, Namespace distribution and replication are becoming a major challenge as Namenode server needs to coordinate with data nodes for its status, job execution and data blocks operations. Usage of Namespace will resolve the issue of namenode single point of failure and namespace scalability. This paper discusses the key distribution techniques for storing large amount of data (petabytes) and load balancing on namespace server. A comparative analysis of majorly used Key distribution techniques has been provided and it has been analyzed that geographic based key distribution technique are the best candidate for load balancing on Namespace server as it can distribute the keys evenly and efficiently.

Keywords: Namespace; data; key distribution; load balancing; Geographic based distribution

I. Introduction

The exponential growth of internet based applications and web services in last few years have changed the techniques to store and analyze voluminous data. The enterprises are looking for reliable solutions for storing and processing such huge amount of data produced by these applications.

Hadoop is one such architecture ([1], [2], [3]) which is used for data intensive computing, used to process the data in parallel, across many nodes in a cluster. It can easily perform data processing tasks like searching and indexing as it has powerful distributed file system ([1], [2], [3]). The Hadoop Distributed File System (HDFS) [4] has namenode servers and data nodes [4] where namenode server maintains the metadata called namespace and the data nodes store and process the data [4]. Namespace has information about namenode servers, file, blocks, replica, data nodes and running jobs. HDFS is highly reliable as it replicates chunks of data to nodes in the cluster and namespace is cached in centralized namenode server. Major advantage of Hadoop is that it provides vertical scalability for namespace management. One of the important issues which need lot of consideration is that the namespace is growing and it requires huge amount of primary memory [5]. Namespace distribution techniques are required to address the issue of scalability [5] and single point of failure. Distributed Hash Tables (DHT) are used to distribute the namespace. DHTs [6] is a class of a decentralized distributed system that provides a lookup service similar to a hash table; (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes.

The goal of a DHT is to allow anyone to find the node that corresponds to a given key by using a flexible architecture to store large amount of data on namespace servers and balance the load on the nodes. There are few key distribution techniques which help to distribute a big namespace to multiple namenode servers.

II. Key Distribution Technique

Mainly there are three key distribution techniques:

Fixed Key Distribution: In Fixed distribution technique, the namespace object ids are mapped to namenode in a predefined range. Any key falling under a specified range will be managed by that namenode server irrespective of its location and load.

Let's consider a scenario where there are 10 nodes and 1 lac keys. In fixed distribution the keys are divided among ten nodes i.e. 10000 keys per namenode. A fix range of 10000 keys is assigned to each node. Whenever a new file is created in HDFS, its object ids are generated dynamically. These ids belong to some range where they are placed.

Table I: Keys values and node nos. for fixed key distribution:

Keys values	Node nos.
0-9999	NN0
10,000 -19999	NN1
20,000-29,999	NN2
30,000-39,999	NN3
40,000-49,999	NN4
50,000-59,999	NN5
60,000-69,999	NN6
70,000-79,999	NN7
80,000-89,999	NN8
90,000-99,999	NN9

In Table I, NN represents name nodes and values represent keys. It is fixed distribution where range is predefined for each node and the keys belonging to the range will go to that particular node. A logical ring is formed of these 10 nodes and keys are assigned to those nodes. Each node contains information about its successor node and its predecessor node. If there are N nodes and K keys, then each node is responsible for roughly K/N keys. Since there are 10 nodes and 1 lac keys, each node is responsible for $100000/10=10,000$ keys.

A DHT contains key and its value. This distribution will assign keys of specified range to different nodes and nodes will store the values for all the keys for which it is responsible. It adopts key lookup table to partition the files among different nodes. It is possible that multiple object ids, which are dynamically generated, may belong to a particular range so the namenode which is catering the namespace service is imbalancing the namespace load. This scenario is even worse when billions of keys are generated. It will generate load on that particular node while other nodes will sit idle leading to load imbalance among all the nodes. The ideal key distribution should distribute keys evenly among all the nodes.

This technique is simple and is good in a scenario where keys are small in number, leading to less uneven distribution of keys. Another important issue is that when a new node joins, a new range will be assigned to that namenode and key migration and shuffling will happen. Reshuffling keys again is a time consuming and costly process.

Uniform Key Distribution: In uniform distribution, modulus arithmetic operation is used. If there are N number of nodes and K are the number of keys then this key will reside on namenode $K \text{ modulo } N$. This technique distributes keys evenly but it relies on fixed initial number of namenode which may not be feasible in real life. Consider an example in which there are 32 nodes and 1 lac keys. In uniform distribution the keys are divided among 32 nodes using modulus. So keys are distributed using a circular fashion among namenode ids from 0-31. The HDFS object ids are distributed to the namenodes using modulo.

Table II: Key values and namenode ids for uniform key distribution.

NameNode ID	Keys	NameNode ID	Keys
NN0	Object ID modulo 32=0	NN16	Object ID modulo 32=16
NN1	Object ID modulo 32=1	NN17	Object ID modulo 32=17
NN2	Object ID modulo 32=2	NN18	Object ID modulo 32=18
NN3	Object ID modulo 32=3	NN19	Object ID modulo 32=19
NN4	Object ID modulo 32=4	NN20	Object ID modulo 32=20
NN5	Object ID modulo 32=5	NN21	Object ID modulo 32=21
NN6	Object ID modulo 32=6	NN22	Object ID modulo 32=22
NN7	Object ID modulo 32=7	NN23	Object ID modulo 32=23
NN8	Object ID modulo 32=8	NN24	Object ID modulo 32=24
NN9	Object ID modulo 32=9	NN25	Object ID modulo 32=25
NN10	Object ID modulo 32=10	NN26	Object ID modulo 32=26
NN11	Object ID modulo 32=11	NN27	Object ID modulo 32=27
NN12	Object ID modulo 32=12	NN28	Object ID modulo 32=28

NN13	Object ID modulo 32=13	NN29	Object ID modulo 32=29
NN14	Object ID modulo 32=14	NN30	Object ID modulo 32=30
NN15	Object ID modulo 32=15	NN31	Object ID modulo 32=31

In Table II, NN represents namenodes and the value represents the keys. It is uniform distribution where keys are evenly divided among namenodes and a logical ring of these 32 nodes is formed. Keys are assigned to those nodes. Each node contains information about its successor node and its predecessor node. If there are N nodes and K keys, then each node is responsible for roughly K/N keys. Here we have 32 nodes and 1 lac keys so each node is responsible for $100000/32$ key.

Keys are assigned to different namenodes according to modulo operation. In dynamic key allocation, the key ids are generated sequentially and are equally divided among namenodes. So, it can evenly distribute even billions of keys and there is no load imbalance [7] on particular node.

This technique is simple and is good in a scenario where keys are of any number. The issue of addition of new namenode will remain as in fixed distribution. The key migration and shuffling will happen. Reshuffling keys again is time consuming and costly. The uniform distribution [7] only solves the problem of skewed load distribution of keys. Other challenges like churning of namenode will exert pressure on the resources.

Random Key Distribution: In this technique, the keys are to be distributed randomly [8] to namenode servers. This technique is quite simple but has uneven distribution of keys leading to uneven load distribution. The issue of namenode churning requires key migration and is costly.

There is a need of a namespace distribution technique that ensures even distribution of keys, minimum migration and usage of resources while considering namenode churning and locality of keys. The locality of keys means that the keys of a particular geographic region will be available locally to that region and its replication are kept at different region to provide high availability.

Table III: Comparative Analysis of Fixed, Uniform and Random Key distribution:

Sr. No.	Characteristics	Fixed distribution	Uniform distribution	Random distribution
1.	Distribution Technique	Specified fixed range	Distribution based on modulo	Distribution based on random function
2.	Load Balancing	Uneven	Uniform	Uneven
3.	Size(No. of keys)	Small no. of keys	Any no. of keys	Any no. of keys
4.	Locality of keys	Can be fixed	Not supported	Not supported
5.	Namenode Churning	Shuffling of keys is required	Shuffling of keys is required	Shuffling of keys is required
6.	Namenode count	Fixed	Fixed	Dynamic
7.	Keys Count	Fixed range	Dynamic	Dynamic

There is a definite need of an overlaying network structure for namenodes placed geographically which has above mentioned characteristics and is highly scalable. The ideal technique would be to consider locality of data i.e. data is available within local domain. So, geographic based distribution may be developed for the distribution of keys.

Geographic Based key distribution: Geographic hash table hashes keys into geographic coordinates, and stores a key-value pair at the name node server geographically nearest the hash of its key ([9], [10]). It uses an efficient consistency protocol to ensure that key-value pairs are stored at the appropriate nodes and it distributes load throughout the network using a geographic hierarchy. A data object is associated with a key and each node in the system is responsible for storing a certain range of keys. By hashing keys, GHT spreads storage and communication load between different keys evenly throughout the namespace.

III. Conclusion and Future Scope

The three key distribution techniques are discussed above i.e. the keys are assigned to namenode servers according to these techniques and a comparative analysis of all the techniques has been provided. DHT

approaches is used to distribute and balance the load of namenode servers with help of key distribution techniques as it resolves the issue of single point of failure and provides high availability. Further, it is analyzed that structured P2P DHT approach is most suitable for load distribution of namenode.

In future, geographic based key distribution may be developed to enhance the performance of distributed namespace. It will decrease the usage of bandwidth to other sites and maintains the data locality for namespace. At the same time it should have characteristics like uniform distribution, self-healing, self-organizing, self-repairing, scalable, reliable, and decentralized. A new replication technique for distributed namespace may be developed to guard namenode against failure.

References

- [1] Tom White, Hadoop: The Definitive Guide, O'Reilly Media, June 2009.
- [2] Jason Venner, Pro Hadoop, APress, June 2009.
- [3] Apache Hadoop Project: <http://hadoop.apache.org>
- [4] Konstantin Shvachko, Hairong Kuang, "The Hadoop Distributed File System" Mass Storage Systems and Technologies (MSST), IEEE 26th Symposium 2010.
- [5] Konstantin V. Shvachko, "HDFS scalability: the limits to growth", Usenix Vol. 35, No. 3, May 2010, www.usenix.org/publications/login/2010-4/openpdfs/shvachko.pdf.
- [6] www.en.wikipedia.org/wiki/Distributed_hash_table
- [7] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, "Dynamo: Amazon's Highly Available Key-value Store", in ACM symposium on Operating systems principles, Volume 41, Issue 6, Pages 205-220, December 2007.
- [8] Zhou, Shuheng; Ganger, Gregory R.; and Steenkiste, Peter Alfons., "Balancing locality and randomness in DHTs" (2003). Computer Science Department. Paper 2238, <http://repository.cmu.edu/compsci/2238>.
- [9] M. Elena Renda, Giovanni Resta, and Paolo Santi, "Load Balancing Hashing in Geographic Hash Tables", IEEE Transactions On Parallel And Distributed Systems, Vol. 23, No. 8, August 2012.
- [10] Sylvia Ratnasamy and Brad Karp, Li Yin and Fang Yu, Ramesh Govindan, Scott Shenker, "GHT: A Geographic Hash Table for Data-Centric Storage" in proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, pages 78-87, September 28, 2002