Design and Implementation of a Novel Peer-To-Peer Information Retrieval Framework

G. SUDHA SADASIVAM[1]
V. KARTHIKEYAN[2]
P. RAJA[3]

[(1, 2, 3)]Department of Computer Science & Engineering
PSG College of Technology, Peelamedu
Coimbatore, TamilNadu 641004, India
[1]sudhasadhasivam@yahoo.com, [2]karthikasturi@gmail.com,[3] rajapcs@yahoo.com

Abstract. Peer-to-Peer (P2P) Information Retrieval framework consists of a peer to peer network of nodes, which voluntarily agree to share their resources by joining the network. While joining these nodes construct the active peer list. Each peer maintains a B+ tree containing IP hash values. The files are distributed over the peer to peer network based on the keywords. The files are initially uploaded into the target node based on the closest match between the hash values of the IP address of the node and the keywords used to index the file. While searching, the target node is identified by finding the closest match between the hash value of the keyword and the IP address hash from the B+ tree stored in the peers. After identifying the target node, the references to desired document is retrieved by searching a B+ tree indexed using keywords. The proposed framework uses Hadoop cluster to extract keywords from the files to be uploaded in the desired target node. Hadoop's MapReduce programming paradigm reduces the time for keyword extraction. As the framework maintains a B+ tree in the peers, it further reduces the search time and improves network bandwidth.

Keywords: P2P networks, information retrieval, Hadoop, B+ tree.

(Received December 15, 2008 / Accepted March 16, 2009)

1 Introduction

A P2P network is a network in which all the nodes have equal priority. A P2P computer network exploits diverse connectivity and the cumulative bandwidth of participating peers rather than using centralized resources wherein a relatively less number of servers provide the service. As nodes in the network are interconnected, there is no single point of failure. P2P networks are characterized by high processing power and storage without the overhead of high cost hardware. An overlay network is a computer network which is built on top of another network. Nodes in the overlay can be thought of as being connected by virtual or logical links. P2P networks can be viewed as overlay networks because they run on top of the Internet.

First generation P2P file sharing networks, such as Napster [3], relied on a central database to co-ordinate look ups on the network. Second generation P2P such as Gnutella [6], used flooding to locate files, searching every node on the network. This had a negative effect on the scalability of the system. Third generation P2P use Distributed Hash Tables (DHT) to look up files in the network. Some examples of 3G P2P networks include Tapestry, Chord, Pastry, and Content Addressable Networks (CAN). In Chord [7] the node keys are arranged in a circle. Identifiers and keys are assigned an m-bit identifier using consistent hashing. In this

framework, utmost O(log N) nodes have to be contacted to find the successor in an N-node network. In our proposed methodology as each peer maintains a B+ tree containing IP hash of peers, at most O($\log_M$N) searches are needed through the M-ary B+ tree in the peer. Thus the peer on which the desired file is placed can be directly identified. In Pastry[1] the key-value pairs of IP are stored in redundant P2P network of Internet hosts. The protocol is initialised by supplying it with the IP address of a peer already in the network. The routing table is then constructed dynamically. Like Chord and Pastry, CAN [8] is also scalable, fault tolerant and self organising. A CAN peer maintains a routing table that holds the IP address and virtual coordinate zone of each of its neighbor coordinates. A peer routes a message using greedy forwarding strategy to the neighbor peer that is closest to its destination peer. In all the DHT approaches mentioned above, utmost N nodes have to be contacted to locate the desired peer. In the proposed approach, as a map of IP hash values is maintained in each peer, the target node can be identified from any peer. Thus our proposed strategy minimises the network bandwidth. In Kademlia [5] the node ID provides a direct map to file hashes and that node stores information on where to obtain the file or resource. Like other DHT approaches, Kademlia contacts only O(log(N)) nodes during the search out of a total of N nodes in the system. Our approach contacts the destination node only once and search occurs within the peer to retrieve file reference using keyword indices in a B+ tree. The network bandwidth is efficiently utilised as the communication between peers is minimised. Section two describes the architecture of our proposed P2P information retrieval framework. Section three presents the implementation details of the system. Section four describes the experimental results.

2 System Architecture

The proposed framework consists of a P2P cluster and a Hadoop [2] cluster. The purpose of the Hadoop cluster is to extract keywords from a set of documents/files in an efficient manner. These keywords are used to index files for efficient searching. It uses a parallel programming paradigm called as MapReduce programming. Once the keywords are extracted, it is given to one of the peers in P2P cluster. The files are then uploaded to the target peer based on the closest match of its IP hash with keyword hash. The efficiency of the framework is attributed to the use of distributed hashing, Hadoop framework and B+ trees.

2.1 Hashing

Hashing is done for the IP addresses and the keywords. Secured Hashing Algorithm (SHA1) [4] is used to produce the 160-bit hash value. The 160-bit hash value is converted into 40-bit value for the convenience of maintaining the same in the B+ tree.

2.2 Hadoop

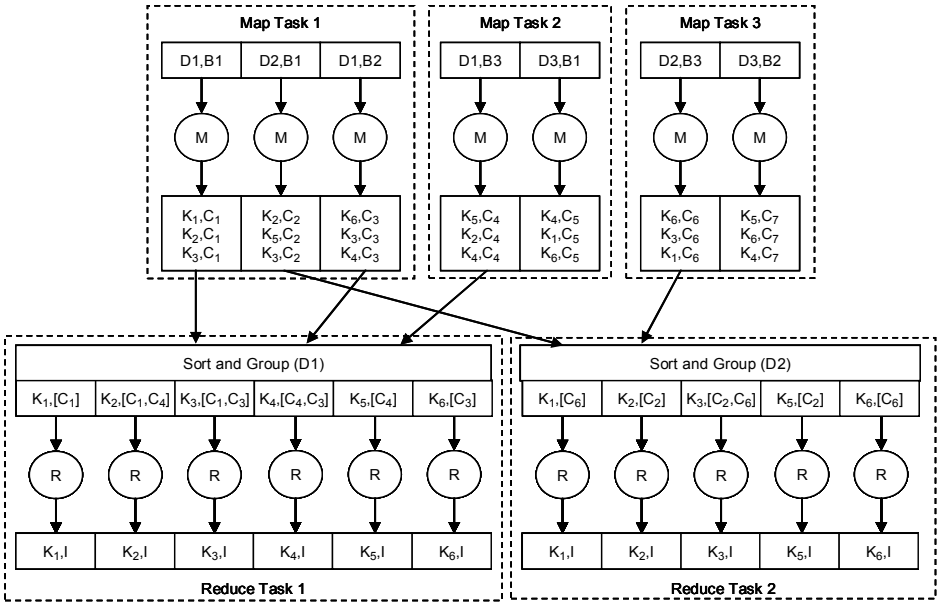
Hadoop[2] is a software platform specifically designed to process and handle vast amounts of data. It is based on the principle that moving computation to the place of data is cheaper than moving large data blocks to the place of computation. The Hadoop framework consists of the Hadoop Distributed File System (HDFS) that is designed to run on commodity hardware and MapReduce programming paradigm. HDFS is highly fault-tolerant and is designed to be deployed on low-cost commodity hardware. Hadoop is scalable, economical, efficient and reliable. Hadoop implements Map Reduce, using the HDFS. MapReduce divides applications into many small blocks of work that can be executed in parallel. HDFS creates multiple replicas of data blocks for reliability, placing them on compute nodes around the cluster. MapReduce can then process the data where it is located.

HDFS has a master/slave architecture. A HDFS cluster consists of a single NameNode and a number of DataNodes. The NameNode is a master server that manages the file system namespace and regulates access to files by clients. The DataNodes manage storage attached to the nodes that they run on. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients.

MapReduce is a programming paradigm that expresses a large distributed computation as a sequence of distributed operations on data sets of key/value pairs. The Hadoop MapReduce framework harnesses a cluster of machines and executes user defined MapReduce jobs across the nodes in the cluster. A MapReduce computation has a map phase and a reduce phase. The input to the computation is a data set of key/value pairs.

In the map phase, the framework splits the input data set into a large number of fragments and assigns each fragment to a map task. The framework also distributes many map tasks across the cluster of nodes on which it operates (Figure 1). Each map task consumes key/value (K,V) pairs from its assigned fragment and produces a set of intermediate key/value (K',V') pairs. The framework sorts the intermediate data set by key and produces a set of (K',V'*) tuples. In the reduce phase, each reduce task consumes the fragment of (K',V'*) tuples assigned to it. For each such tuple it invokes a user-defined reduce function that transmutes the tuple into an output key/value pair (K,V).

The Hadoop MapReduce framework has a master/slave architecture (Figure 2). It has a single master server or jobtracker and several slave servers or tasktrackers, one per node in the cluster. The jobtracker is the point of interaction between users and the framework. Users submit map/reduce jobs to the jobtracker, which puts them in a queue of pending jobs and executes them on a first-come/first-served basis. The jobtracker manages the assignment of map and reduce tasks to the tasktrackers. The tasktrackers execute tasks upon instruction from the jobtracker and also handle data motion between the map and reduce phases.



D1, D2, D3 – Datanode B1, B2, B3 -Blocks K1, K2, K3 -Keywords
C1, C2, C3 –Count of the keyword I-Index M-Map R-Reduce

Fig. 1. MapReduce Programming

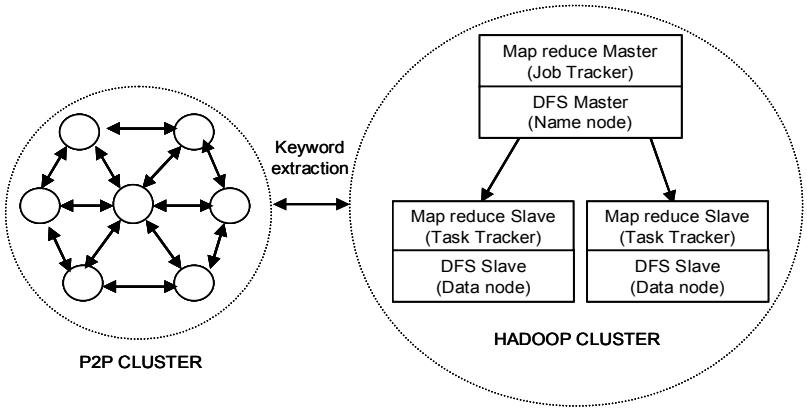


Fig. 2. Architecture of P2P Information Retrieval System

2.3 B+ Tree

The list of IP addresses of peers along with their hash values is maintained in the B+ tree. This data structure is also used to store keywords along with their references (file names) in the target peer. Each node of the B+ tree maintains the IP address of the peer along with its hash value as shown in Figure 3.

A B+ tree is a type of tree which represents sorted data indexed by a key for efficient insertion, retrieval and removal of records. It is a dynamic, multilevel index, with maximum and minimum bounds on the number of keys in each index segment. In a B+ tree, all the records are stored at the lowest level of the tree, namely the leaf node. The interior blocks contain only the keys.

The order of a B+ tree measures the capacity of nodes in the tree. In a B+ tree with M entries, order D is defined as D <= M <= 2 D, where M is the number of entries in each node. For example, if the order of a B+ tree is 3, each internal node can store 1 to 2 keys. The root can store 1 to 6 keys.

A search for a record R is performed by following pointers to the correct child of each node until a leaf is reached. Then, the leaf is scanned until the correct record is found. For example, to search for an IP with key value of 229, the first link from the root node followed by the second link from the next level node is followed. The bucket is then searched and the IP address of the target machine (IP8) is retrieved.

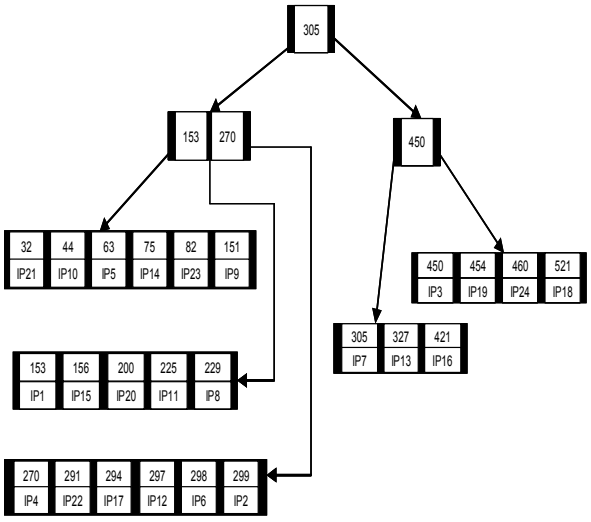


Fig. 3. Organisation of B+ tree

To perform insertion operation,

- The bucket where the new record is to be placed is determined.
- The record is added, if the bucket is not full.
- If the bucket is full, it is split.
- A new leaf is allocated and half the bucket's elements are moved to the new bucket
- The new leaf's smallest key and address is inserted into the parent.

A B+ tree of order B with N records offers the following advantages

- The space required to store the tree is O(N). Hence the entire tree can be loaded in the main memory.
- Inserting a record requires $O(\log_B N)$ operations in the worst case
- Searching for a record requires $O(\log_B N)$ operations in the worst case
- Removing a previously located record requires $O(\log_B N)$ operations in the worst case
- Performing a range query with K elements occurring within the range requires $O(\log_B N + K)$ operations in the worst case.

3 System Implementation

The proposed framework comprises up of 4 major components. They are the start up component, database distribution component, search component, add/delete peer component. These components are described below:

3.1 Startup Component

The functionality of the start up component includes the following

- Starting up the Hadoop cluster
- Identifying nodes that can participate in the P2P cluster.
- Determining the IP hash values for the peer nodes
- Forming the B+ tree.
- Uploading B+ trees in other peers.
- Starting the Web Server.

The IP addresses of all the active nodes in the cluster are identified. The hash value of the IP address is generated using SHA1 algorithm. The 160 bit hash value is converted into 40 bit value for ease of

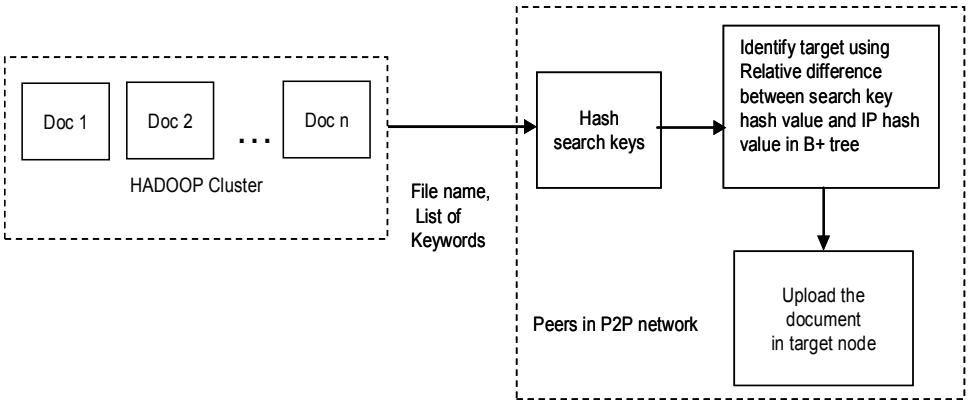


Fig. 4. Database Distribution Component

maintaining the same in B+ tree. The B+ tree is then constructed using IP address as the value and its hash value as key in the source node. As search request can be submitted to any peer, the B+ tree is uploaded into all peers.

3.2 Database Distribution Component

The main objective of this component is to upload files into the target peer to facilitate efficient searching. The sequence of operations performed by this module as shown in Figure 4 includes the following

- Prior to uploading a document to be searched, key words are extracted from it. These keywords are used for indexing the document. Keyword extraction is performed using MapReduce parallel programming paradigm in a Hadoop cluster. Weightage for each keyword is also calculated based on its importance and number of occurrences. Keywords can also be extracted from multiple files (Doc1 to n) in parallel.
- The keywords are hashed using SHA1 hashing function to obtain an 40-bit hash value.
- The keyword hash value is then compared to the IP hash values of the closest peers using the B+ trees in the peers. The peer having the greatest match to the keyword hash value is selected for distribution.
- The desired document is then uploaded into the desired peer.
- The B+tree maintaining the file references using keyword indices is updates in the target peer.
- This process is repeated for all the set of keywords available for the file.

3.3 Search Component

Searching is done on the peer in which the search request is received. The sequence of operations done by the search component (Figure 5) is listed as follows:

- The search request is converted into set of keywords by removing unwanted words from the search request.
- The sets of keywords are hashed to obtain an 40-bit value.
- The B+ tree maintained in the peer is searched to obtain the target peer in which the desired document is hosted.
- The database B+ tree of the target peer is then searched to retrieve file reference using keyword as an index.
- This process is repeated for all the keywords found in the search request. The resulting answer is sent to the peer which initially received the searching request.

3.4 Add/ Delete Peer Component

Whenever a new peer needs to join the P2P network dynamically, the following sequence of actions is carried out:

- The IP address of the new node is also added to the existing IP address table.
- The B+ tree is re-constructed by adding a new peer node to it after computing IP hash value of the newly added peer node.
- The changes made to the newly constructed B+ tree is communicated to the other peers.

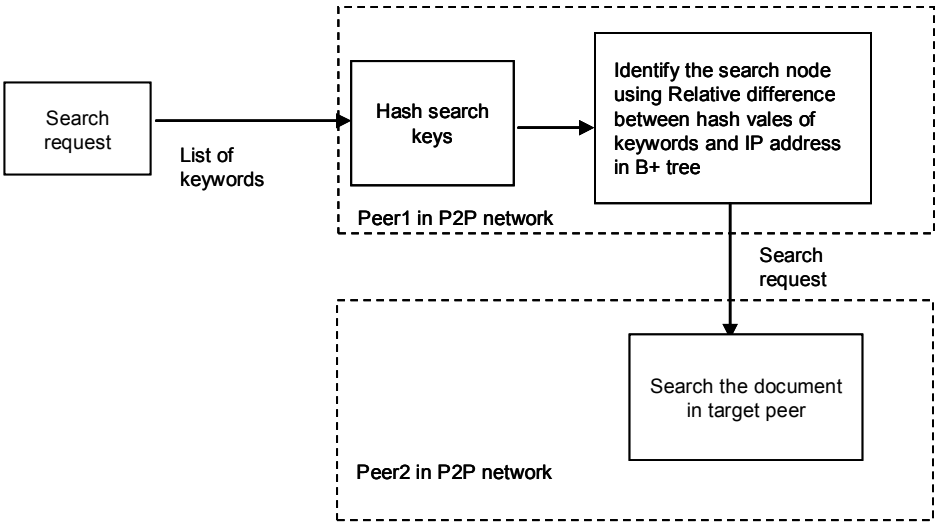


Fig. 5. Search Component

- Then the files closely related to the IP hash value of the new peer are relocated from the peer containing it to the new peer.
- The metadata entry for keywords in the peer is modified.

Whenever a peer leaves the cluster dynamically, the following operations are performed:

- The IP address of the peer node leaving the cluster is deleted from IP address table.
- The B+ tree is re-constructed by deleting the peer node from it after computing IP hash value of the newly added peer node.
- The changes made to the newly constructed B+ tree is communicated to the other peers.
- Then the files in the deleted peer are relocated from the peer containing it to the next closely related peer.
- The metadata entry of keywords in the new peer is changed.

4 Experimental Results

The P2P and Hadoop clusters were set up using ten P4 machines. Fedora ver 8 was used as the operating system and Hadoop ver 0.18 was used as the distributed file system. The block size in HDFS was set to 1 MB. Coding was done using java ver 1.5. The DLS framework thus implemented was tested as explained in the following paragraphs.

4.1 Suitability of Hadoop Clusters for Keyword Extraction

Hadoop framework was used to perform keyword extraction. The suitability of Hadoop for our DLS framework was verified using the following measures:

4.1.1 Keyword Extraction Efficiency

As Hadoop uses MapReduce programming, parallel extraction of keywords can be done from files. The effect of the number of keywords extracted to index files using Hadoop clusters was studied. Experimental results of the study are illustrated using Figure 6. As Hadoop forms a complete data and compute cluster, the time taken to extract keywords from files of size 1KB is almost a constant. It varies from 1 msec to 2 msec when the number of keywords extracted varied from 5 to 20. It is also noticed that the number of peers no not have an effect on the efficiency of keyword extraction.

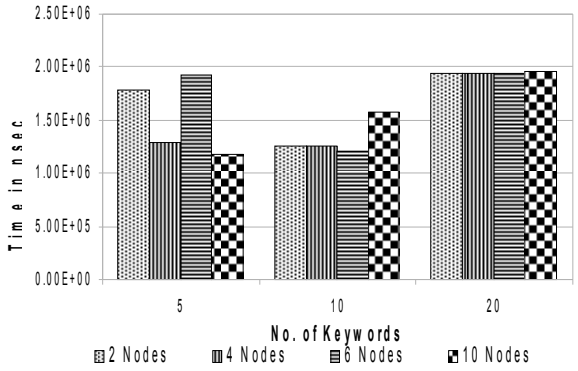


Fig. 6. Keyword Extraction Efficiency

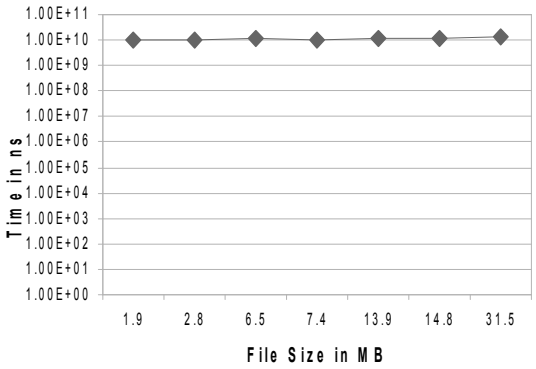


Fig. 8. Keyword Extraction from Files of Different Sizes

4.1.2 Parallel Extraction of Keywords from Multiple Files

HDFS facilitates the extraction of keywords from a number of files in parallel. The size of the files taken was 1 MB. From graph shown in Figure 7, it can be seen that the time taken to extract keywords from multiple files remains constant. Time taken to extract keywords from a single file (1 MB) is approx 10 sec, whereas time taken to extract keywords from 7 files is only 20 sec.

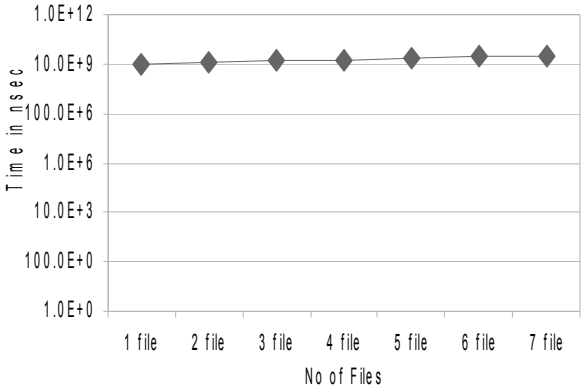


Fig. 7. Keyword Extraction from Multiple Files

4.1.3 Using files of different sizes

Testing was done on files of different sizes from which the 20 keywords were extracted. As keyword extraction can be performed on blocks in parallel, the time for extraction thus remains constant (Figure 8).

4.2 Setup Time

Experimental results (Figure 9) illustrate the time taken to set up a P2P cluster consisting of different number of nodes. The time taken to set up a cluster of 2 – 10 nodes varies from 2.5 s to 8.5 s.

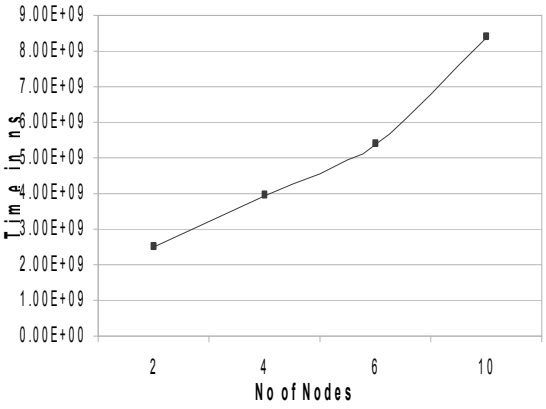


Fig. 9. Performance of Start Component

4.3 Performance of Add/Delete Peer Component

Graph in Figure 10 demonstrates the time taken to add a new peer dynamically to the cluster. This time includes the time for reconstructing B+tree, updating the B+ trees in all other peers, extraction of keywords and storing the files in the target directory. The time taken to add a new peer directly depends upon the number of peers already in the network and the number of keywords used for indexing the files. As the number of keywords increases, the time for storing also increases.

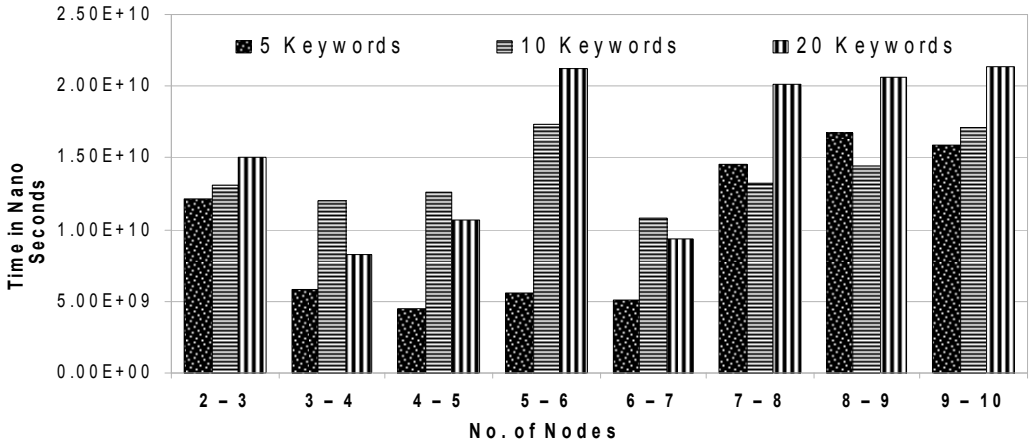


Fig. 10. Performance of Add/Delete Peer Component

As the number of peers increases, the time to update the B+tree increases. Nevertheless, the time taken to add a new peer in the P2P cluster varies from 5 sec to 20 sec in our experimental setup. This time is inclusive of the time for keyword extraction.

4.4 Performance of Data Distribution Component

The time for storing the file in the target peer is computed to evaluate the performance of the data distribution component. It depends on the number of keywords used to index the file as shown Figure 11. As the number of indices increases, redundant copies of the same file have to be made in the peers. We store the files in different peers one after another. The performance can be enhanced by using multicasting.

4.5 Performance of Search Component

The efficiency of the search component is shown in Figure 12. The search efficiency remains a constant at approximately 9 msec. The efficient search performance is attributed to the use of B+ trees and search distribution. Further a B+ tree is a complete M-ary tree with height that is roughly $\log_M N$ instead of $\log_2 N$ (in a binary tree). Here M is the number of internal (non-leaf) nodes and N is the total number of nodes. Consider that that the values of N and M are 220 and 20 respectively. In the case of a B+ tree, the height is $\log_{20} 220$ (less than 5) when compared to a B tree where the height is 20 (ie $\log_2 220$). Thus, the search efficiency is significantly improved. It remains almost a constant (Figure 12).

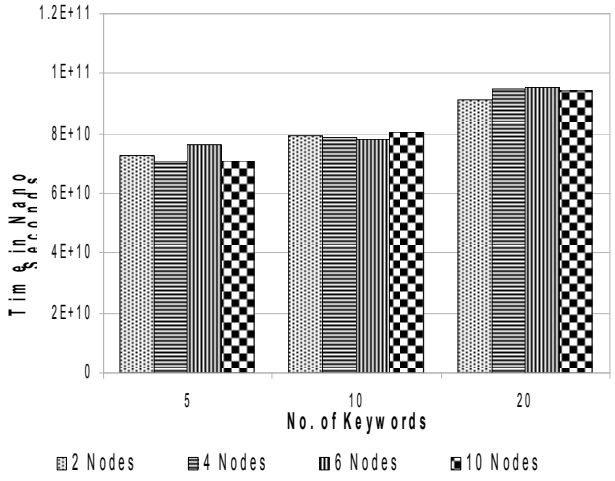


Fig. 11. Performance of Data Distribution Component

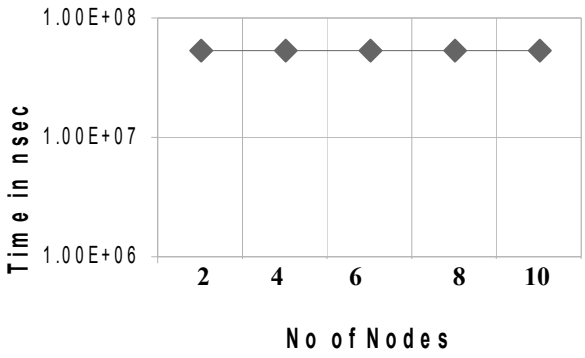


Fig. 12. Performance of Search Component

5 Conclusion

The proposed framework is specifically created to host the digital library application that searches efficiently for files in the 3G P2P network that uses DHT approach. The digital library framework takes advantage of the features given by peer to peer network and the Hadoop framework. It searches for a files indexed by keywords in the target peer only. As B+ tree is implemented for storing and searching the files using keywords with hash values the network bandwidth is efficiently used and hence the searching time is reduced. It is in the order of msec. Storing time is relatively greater than search time as it involves both keyword extraction and file transfer. Keyword extraction is done on multiple target files using Hadoop framework. Hadoop's map-reduce programming strategy improves the efficiency of keyword extraction. The proposed framework can be extended for search on files containing media content also. The fault tolerance in the proposed framework can be enhanced by using virtual machines and by maintaining links to redundant copies.

Acknowledgements

The authors would like to thank Dr.Rudramoorthy, Principal, PSG College of Technology and Mr. Chidambaran Kollengode, Director, Cloud Computing Group, Yahoo Software Development (India) Ltd, Bangalore for providing the required facilities to complete the project successfully. This project is carried out as a consequence of the Yahoo's University Relation Programme with PSG College of Technology.

References

[1] Antony, R.; Peter, D., "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems", *Lecture Notes in Computer Science*, *Available at http://www.cs.rice.edu/~druschel/publications/Pastry.pdf*, 2001.

[2] Apache, Hadoop Documentation, *Available at http://hadoop.apache.org/core/docs/r0.17.2/*, 2002.

[3] Carlsson, B.T and Gustavsson, R. The Rise and Fall of Napster - An Evolutionary Approach, *Proceedings of the 6th International Computer Science Conference on Active Media Technology*, 2001.

[4] Matthew J. B. and Robshaw, MD2, MD4, MD5, SHA and other hash functions, *Technical Report TR-101, RSA Laboratories,* 1995.

[5] Petar,M. and David, M. Kademlia, A peer-to-peer information system based on the XOR metric, *Available at http://www.scs.cs.nyu.edu/~dm/papers/maymounkov:kademlia.ps.gz*, 2002.

[6] Ripeanu, M., Ian Foster and Iamnitchi A., Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design, *IEEE Internet Computing*, 6(1), pp. 120-127, 2002.

[7] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for Internet applications, *IEEE/ACM Transactions on Networking*, 2003.

[8] Sylvia, R., Paul, F., Mark, H., Richard, K. and Scott, S. A Scalable ContentAddressable Network, *SIGCOMM'01,* San Diego, California, USA, 2001.

[9] Sudha Sadasivam ,G., Renuga, R., P2P Information Retrieval Framework for Digital Library System, *Journal of Applied and Theoretical Information Technology,* 2008