A novel Hash-Based File Clustering scheme for efficient distributing, storing and retrieving of large scale Health Records

Thanh Dat Dang
School of Computing and Communications, iNext
University of Technology Sydney, Australia
Thanh.D.Dang@student.uts.edu.au

Doan Hoang
School of Computing and Communications, iNext
University of Technology Sydney, Australia
Doan.Hoang@uts.edu.au

Priyadarsi Nanda
School of Computing and Communications, iNext
University of Technology Sydney, Austraslia
Priyadarsi.Nanda@uts.edu.au

***Abstract*— Cloud computing has been adopted as an efficient computing infrastructure model for provisioning resources and providing services to users. Several distributed resource models such as Hadoop and parallel databases have been deployed in healthcare-related services to manage electronic health records (EHR). However, these models are inefficient for managing a large number of small files and hence they are not widely deployed in Healthcare Information Systems. This paper proposed a novel Hash-Based File Clustering Scheme (HBFC) to distribute, store and retrieve EHR efficiently in cloud environments. The HBFC possesses two distinctive features: it utilizes hashing to distribute files into clusters in a control way and it utilizes P2P structures for data management. HBFC scheme is demonstrated to be effective in handling big health data that comprises of a large number of small files in various formats. It allows users to retrieve and access data records efficiently. The initial implementation results demonstrate that the proposed scheme outperforms original P2P system in term of data lookup latency.**

Keywords— clustering files; P2P systems; file distributions; hashing scheme; cloud based P2P systems

I. INTRODUCTION

Cloud computing has become increasingly popular as an efficient computing infrastructure for managing IT system [1] with rapid migration of both services and applications from users own infrastructures to cloud infrastructure. As a result, more and more data is generated and stored within IT systems. Electronic health record (EHR) systems store digitally healthcare information about an individual's lifetime with the purpose of supporting continuity of care, education and research, and ensuring confidentiality at all times [2]. The process of provisioning healthcare involves massive healthcare data which exists in different forms (structured files or unstructured data) on disparate data sources (relational databases, file servers, etc.) and in different formats (text, images, sensor data, XML files, relational database records). A patient often receives medical treatment from different health professionals in various organizations over his/her lifetime and their health data are stored in different HIS and can be shared to health care providers, insurance practitioners, researchers and family members. Efficient schemes for storing and retrieving EHRs are thus extremely important as they allow significant improvements in providing better healthcare services particularly in the big health data distributed cloud computing environments.

Various distributed storage systems such as Hadoop [3], P2P systems [4, 5] have been widely deployed in cloud environment. However, storing and retrieving a large number of files with various data sizes and structures pose a big challenge for current big data models. The work in [6] proved that Hadoop is not efficient in managing a large number of small files. Hadoop is designed to process large scale data in data mining and machine learning. It uses Hadoop Distributed File Storage (HDFS) (NameNode and DataNodes) as the primary storage. Users' request to access data from HDFS is first submitted to NameNode for namespace services and the data retrieval processes are performed at DataNodes based on metadata obtained from the NameNode. However, these systems that accommodate a large number of small files suffer heavy overhead on the NameNode due to frequent accessing of metadata more from the DataNodes. In fact, numerous searching and hopping from DataNote to DataNote present another inefficient data access pattern. Despite the fact that P2P systems and mechanisms are generally efficient for data lookup with low latency, they do not perform well in processing large-scale data analytic jobs compared to Hadoop [7]. Alternative infrastructures and access mechanisms are needed to handle this type of data and applications. P2P search is efficient and has been used in many real systems. However, for a system with very large number of files relative to the number of storage nodes and if the access pattern is skewed, the files will be distributed to their nearest hashed ID nodes evenly with some nodes heavily loaded and others extremely lightly loaded. Thus, data is spread to various nodes resulting in less efficient for retrieval and lookup process. Hence, distributing, storing and retrieving large scale of data efficiently present a real challenge for P2P systems.

This paper addresses the two major concerns from a novel perspective: clustering EHR files in a controlled manner into a defined number of nodes to minimize the number of steps in searching for a requested file and reduce the data lookup latency among peers. The paper proposes a novel Hash-Based File Clustering Scheme (HBFC) to distribute, store and retrieve EHR efficiently in cloud environments. The HBFC possesses two distinctive features: it utilizes hashing to distribute files into clusters in a controlled way and it utilizes P2P structure for data management. Specifically, we propose a scheme with two hashing rounds, one specifically for efficient data clustering and the other for leveraging the P2P search mechanism. Each hashing round has its own specific hash function and distinct ID space. The first round utilizes the "collision" property of the hash function in a controlled way to gather data into virtual clusters in a virtual ID ring (first ring), Virtual clusters since they are not the real nodes where file are found but they are the IDs where cluster of files are hashed to and the real cluster nodes are in the second ring.. The second round hashes the virtual IDs to hashed ID of real nodes of a P2P system and leverages its P2P properties. Our Hash-Based File Clustering (HBCS) scheme first reduces the searching space and then leverages the P2P searching mechanism to achieve data lookup efficiency. Due to the features of one way hash function, files associated with virtual ID are stored in the same or closest clusters. The requested files are mainly stored in these clusters. Besides leveraging P2P file search, we design efficient search schemes for files within a cluster. We are not aware of any existing methods that use "collision" to control the formation of clusters as designed in our scheme.

The rest of the paper is organized as follows. Section II discusses related work on various distributed models. Section III presents our HBFC scheme clustering and searching files. Section IV describes the design of proposed scheme. Section V presents the simulation results and evaluations of our proposed scheme. The conclusion will be drawn in section VI.

II. Related work

Related work associated with different distributed models has attracted recent research on improving the performance for Hadoop.. [6] analysed issues with small file problem on Hadoop and proposed a small file scheme for Hadoop using index strategy including merging, prefetching and caching. Although the metadata caching and index prefetching can reduce access latency and improve access efficiency, the global index file strategy results in additional overhead to NameNode. ETL (extraction, transformation and loading) technique which is responsible for the extraction of data from various heterogeneous data sources has been adopted to achieve processing the data in distributed environment more efficiently. [8, 9] worked on ETL technique to reduce the latency in querying data from databases.

Chen [7] presented BestPeer++, a system which delivers elastic data sharing services for corporate network applications in the cloud based on BestPeer – a peer-to-peer (P2P) system based data management platform. This model was deployed in cloud environment as a service that uses bootstrap peers as monitors, normal peer as database engine and access control. An adaptive query is proposed to switch between a P2P engine for small scale data to leverage fast performance on the local database engine and Map Reduce engine for large scale data in order to exploit the benefits of the Hadoop model. [10] proposed a hybrid P2P system as a combination of a structured P2P called *t-network* and an unstructured P2P called *s-network*. The *t-network* plays the role of a core transit network while the *s-network* stores data in the system and each *s-network* is attached to a peer in the t-network. The p_s parameter is defined to adjust the number of peers in each of the two networks. Altmann [11] presented a P2P file sharing topology based on social network allowing users to share resources with their friends or family. The number of peers accessing the resources based on their established relationship is, however, limited. [12] presented a data replication approach to avoid overload of some objects as hotspots by using multiple hash functions. These hotspots are replicated to different nodes dynamically resulting in improvements in access latencies and load balancing. In [13], authors proposed a model to distribute contents to users in an overlay network as a Constrained Stochastic Graph Process (CSGP). The CSGP is modelled by different architectures based on trees and meshes. The file distribution is only considered to deliver the content in the shortest possible time to users in the P2P systems while the searching task is not a main focus.

In brief, these efforts focus mainly on structuring P2P systems to deal with the dynamic of nodes joining and leaving, the data distribution remains unchanged, relying on the traditional DHT P2P approach. Besides that, searching requested files is performed the same way as the original system. Our proposed scheme focuses on efficient distributing and retrieving data in two stages: file clustering and file searching within a cluster. Through the first hashing round, data is distributed into a certain number of defined clusters which have appropriate size such as number of nodes to serve for retrieving data efficiently. We propose to use the collision property of a hash function in a designed and controlled way to perform file clustering. Thus, files are hashed into defined clusters and the default searching mechanism in intra-cluster is employed for file searching.

III. The combination of clustering and searching files

In overlay sharing files systems, data lookup process is the most computationally expensive task while in file distribution systems, if the locations of data clusters are well designed, and the expensive task is the retrieval data. Good searching algorithms enable speeding up the data lookup process. Hashingis known as the fastest searching algorithm since its complexity is O(1). P2P systems utilize the hash functions to calculate data key in order to reach peers storing requested files with low latency. To benefit from hash function features, we utilize it in our proposed schemes in both clustering and searching files. On the one hand, considering that it costs only O(1) computation time, the searching performance of the system is not affected. On the other hand, storing files at the clusters in a controlled manner allows us to find requested files in a shorter time frame. Next, we will detail how to control the distribution of files into clusters.

In order to control clustering files, we treat the system's configuration as an adjustable parameter so that scaling up or down can easily be achieved when the system's configuration changes. Parameters defined are the number of files in the system, the expected average number of files stored in one cluster, and the number of nodes in the system. The idea of combining clustering files and searching files stems from two considerations: 1) if we can store most files of the system in defined clusters, the searching process is mainly performed at these clusters; 2) distributing a certain number of files in a controlled number of clusters enables retrieving data more efficiently compared to retrieving it among a large number of files in one cluster. Based on these, we adjust the parameters to control the hashing of files into clusters to achieve the expected file distribution.

Suppose that there are n_f files. The number of real nodes that form the P2P network is denoted by n_n. The number of defined cluster nodes is denoted by n_c. The number of files per node is denoted by f_c. The number of nodes which are not in defined cluster nodes is denoted by $\overline{n_c}$. The average number of files in $\overline{n_c}$ nodes is denoted by $\overline{f_c}$.

The expected number of nodes storing most of the files in the system is derived as

$$n_c = \alpha * n_n \quad (1)$$

where $0 < \alpha \leq 1$.

$$\overline{n_c} = n_n - n_c \quad (2)$$

The expected number of files distributed in n_c nodes is derived as

$$n_c * f_c = \beta * n_f \quad (3)$$

where $0 < \beta \leq 1$

As a result, the computation formula for f_c is derived as

$$f_c = \beta * \left({}^{n_f}/_{n_c} \right) \quad (4)$$

$$\overline{f_c} = \left(\frac{n_f - (\beta * n_f)}{\overline{n_c}} \right) \quad (5)$$

As can be seen from the formulas (1) and (3), we are able to control file distribution proportions given n_f and n_n as input parameters of the system. By turning the α and β parameters, we can control the percentage of clusters and the percentage of files distributed within these clusters. The first ring produces expected distribution rates as following;

1. $n_c = \alpha * n_n$ nodes that store most files of the systems. As a result, the retrieval process is focused on these nodes resulting in efficiency in lookup latency and access latency.

2. $n_c * f_c$ files are distributed across n_c nodes and f_c files per cluster results from fast look up process since we reduce the lookup space to f_c

3. $\overline{f_c} * \overline{n_c}$ files are left over in small number of clusters. $\overline{n_c}$ is designed to be much smaller than n_c.

IV. System overview

In a normal structured P2P system, the Distributed Hash Table (DHT) is built on top of an overlay network. Data lookup and insertion processes are efficiently supported by the hash table. Given the key of the file, the corresponding value of the file can be inserted and searched by hashing the key to a value with an appropriate hash function.

The hash value is the index of the file and all the hash values form the ID space. In DHT, peers are delivered in the ID space. Each peer is responsible for one partition of the ID space. Peers are connected by an overlay network through which the requests for data insertion and lookup are delivered. Basically, there is only one hashing round using the data key such as filenames or file content to obtain the value corresponding to the file. Files might be distributed relatively evenly based on hash functions in the original P2P systems. However, for a system with very large number of files relative to the number of storage nodes and if the access pattern is skewed, the load files will be distributed to their nearest hashed ID nodes may not be evenly with some nodes heavily loaded and others extremely lightly loaded. In other words, there may be some nodes storing large amount of files in the system while others have few files or no files. Consequently, load balancing is not achieved and data retrieval is not efficient. Firstly, peers still perform the lookup process in the same ID space but the destinations nodes tend to access several nodes in the systems. In addition, storage space of peers is not used efficiently causing not only unbalanced distribution but also lots of unused storage in peers.

We propose a new scheme namely the Hash-Based File Clustering scheme to address these issues. The basic idea is based around hash functions since they return the lookup results quickly and have lowest computational complexity of O(1). The new contribution of our design is based on both benefits and weakness of hash function features when applied to P2P systems. Specifically we propose to use the collision property of a hash function in a designed and controlled way to perform file clustering. Our scheme focuses on efficient distributing and retrieving data from P2P based on two hashing rounds in order to enhance the performance of P2P systems in two stages: file clustering and file searching within a cluster. Details of the design of each ring are presented in the next section.

A. *The design of HBFC*

In the HBFC, we use two different ID spaces to store the hash values of each hashing round. The first ID space has a small ID range representing file distributions on cluster nodes. Different files may be mapped to the same ID on the first ring as the result of hash collisions. We call a "collision" a "hit" and the number of hits per cluster can be controlled. We call this ring a virtual ID space; an ID on this ring represents a virtual cluster as it only contains a cluster of IDs, not a real cluster that contains files. This ID will be hashed into an ID of a real cluster in the second ring where files are found. The second ID space is designed with a large ID range as often the case in many existing P2P systems. Large P2P's ID space and proper hash functions allow keys to be hashed into unique IDs in an uniform distribution. An additional feature of our scheme is that only the cluster ID is used in the second hash round and the (file) data key is preserved so that searching for the data file within a file cluster can be performed. Figure 1 depicts the design of HBFC. As can be seen from the figure, there are two rings used to store hash values for each round. The first ring is designed as a virtual ring to gather hits from hash function (*h1*) when hashing different data keys to *s_id* in the first ID space. The aim is to control the expected number of files distributed into the same cluster. The second ring follows the original P2P's ring which hashes the *s_id* to *l_id* in the second ID space. Based on the feature of one way hashing that is the hash function only produces the same hash value for each data key. Therefore, if data keys are hashed with a value *s_id*, then this ID is definably hashed into the corresponding *l_id*. In order to obtain this design, we also modify the second hash function and attach the data keys for the second hashing round. By doing this, the requested files can be found at the destination node. The details of this design are presented in following subsections.

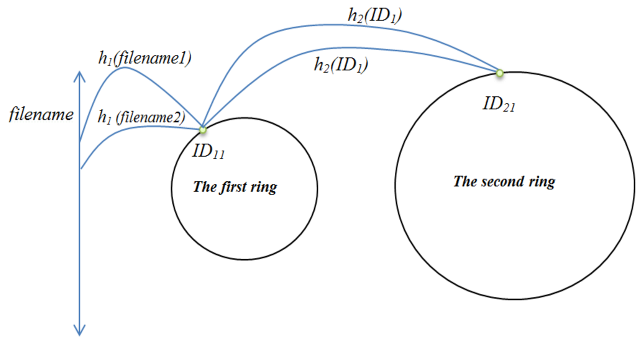

Fig. 1. The design of HBFC

A data item such as a file or a group of files is represented by a (key, value) pair. A key is a label or name of the data, such as a file name, while a value represents the content of the file. The pair (key, value) is inserted into the system by peers and key is used for lookup process in order to retrieve the corresponding value. To start searching a file, the peer first hashes the key to *s_id* which is a defined hit in the first ID space. Following that, *s_id* is used as the data key for the second hash function. It is noted that we attached the data key into s_id to keeep the identity of the data. In the second hashing round, the *s_id* is hashed to *l_id* which is an unique ID in the second ID space. In other words, hashing different data keys may produce the same ID, this means that these files are stored in the same cluster. The next subsections will detail the design of each ring.

1) The first ring

The first ring is a virtual ring providing a small ID space for hash function in order to obtain the balance distribution when many data keys are hashed to one ID. Figure 2 depicts the design of first ring.

Let h_1 denotes the hash function of the first ring, the computation of h_1 is derived as follows.

$$h_1(key) = ID1$$

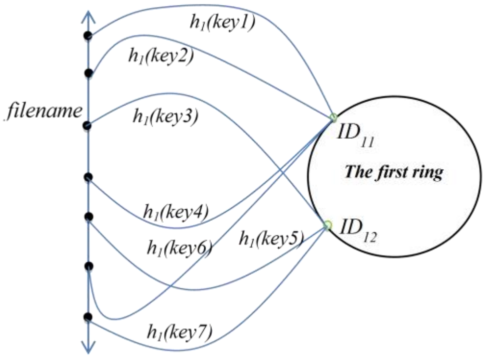

Fig. 2. The design of first ring

As can be seen from the figure, different data keys are hashed to the same ID. For example, *key1*, *key2*, *key4* and *key6* are hashed to ID_{11} while *key3*, *key5* and *key7* are hashed to ID_{22}. The hash values such as ID_{11}, ID_{22} then are used as data keys for the second hash function. As a result, we are able to achieve our design in distributions due to controlling a certain number of files stored in one cluster. Firstly, many data keys hashed to the same ID and that implies that multiple files are stored in the same cluster. In fact, the number of data keys mapped to the same ID is controllable. However, there is an issue of losing the identity of the requested files when we hash the data keys with two hashing round. Since ID_{11} are ID_{22} are virtual data keys or temporal keys, they do not provide the identification of requested files which are searched on the real nodes. Their values are only used to reach the nodes that stored requested files in the second ring representing real P2P nodes/systems. In order to mitigate this issue, we propose the attached keys within the hash function to search the files at real nodes in original P2P systems. Details of the attached key are presented in the next section.

2) The second ring

In the second hashing round, the second ring provides *a large ID space so that no collision of nodes is expected*. It follows exactly original P2P hash functions. The mapping 1-1 is performed between the small ID spaces to the large ID spaces. In this stage, ID obtained from the first hashing round is used as the data keys for the second hash function (h_2). Figure 3 depicts the design of second ring.

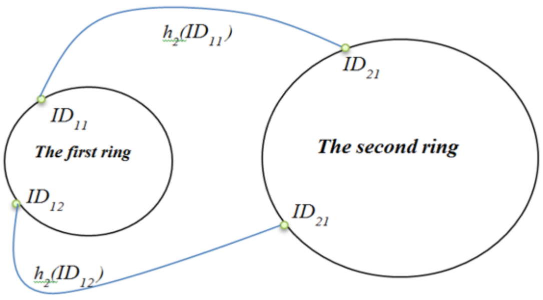

Fig. 3. The mapping 1-1 between the first ID space and the second ID space

In P2P systems, a requested file is hashed into the ID space in order to reach the peer that stores the intended file. In this process, a real node may not exist at the hashed ID (figure 4 below) and hence from this hashes ID, the lookup process is continuously performed in order to access closest real nodes. The range of lookup is determined by searching radius over the ring.

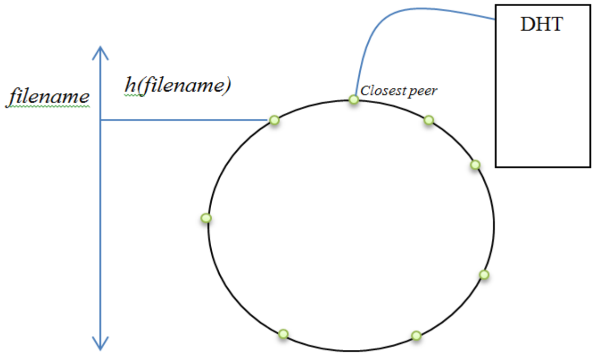

Fig. 4. Find closest node in P2P system

Despite the fact that our scheme is able to gather many files into one virtual node, we still have to deal with the problem finding individual file in a real node. We identify the lookup space of searching individual file in a real node to evaluate the HBFC.

The lookup space of closest node to the real node for an individual file may fall into the following scenarios: the real node may store a variable number of requested files as their IDs are closest to the real node ID; the node may store a random number of files and/or a variable number of small clusters not in the defined hits as the individual file IDs and the ID of these clusters are closest to the real node ID; the node may store individual files and/or files in small clusters and/or files in defined clusters. Generally, if the requested files fall within a defined clusters, HBFC will return results more efficiently due to the small lookup space. In other cases, HBFC cost approximately the same as the original P2P system since they execute the same lookup mechanism.

V. SIMULATION RESULTS

We run the HBFC on P2P systems with different parameters to validate our design. The results of HBFC are then compared with the original P2P system to determine the performance. For the virtual ring, the djb2 hash function [14] is used to hash data keys into the first ID space. we apply the HBFC in P2P system based on Chord [15, 16] and modified as our design scheme and compare its performance to the original P2P system which is also followed Chord without modifications. The experiment programs are coded using the Java programming language. The results of HBFC are compared to that of original P2P system in terms of scalability and system performance. Parameters of Chord's hash function are also processed as normal to hash only the *s_id* to the second ID space. However, we modify the process to attach the file's data key with the ID to allow the lookup process for the requested files in peers. The simulations are based on following parameters: the number of nodes, the number of files in the systems and the number of requested files.

The idea is to identify the performance efficiency of HBFC scheme in distributing and searching a number of files compared to original P2P systems. We run the simulations with different parameters. In the first simulation, we select a fixed number of nodes and number of files but input various number of requested files to evaluate the performance of the systems. In the second simulation, we increase the number of files in the systems with the same number of nodes and the same number of requested files. In the third simulation, we increase the number of nodes with the same number of files in the systems and the number of requested files.

Test case 1. The number of nodes and the number of files are fixed; the number of requested files is changed.

We initialized two sets of tests in which the number of files (10000000 files) and the number of nodes (5000 nodes) are unchanged. The number of requests in the system increases from 1500 to 10000 requested files. Figure 5 illustrates the execution time of data lookup process for different number of requested files. The x-axis represents the number of requested files, and the y-axis represents the execution time. As can be seen from the figure, the time cost of original P2P (T2) is approximately double compared to that of HBFC (T1). It demonstrates that within the same number of requests, the HBFC performs more efficient than original P2P since many files are distributed in one or several clusters.

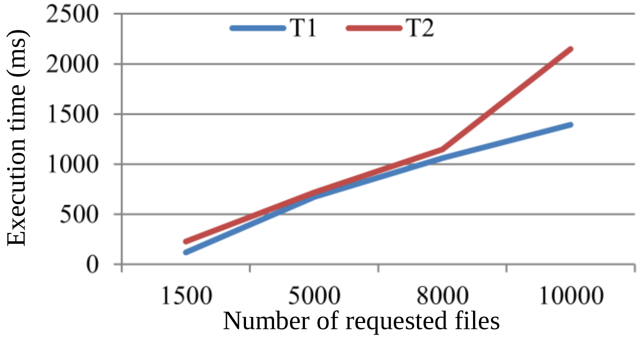

Fig. 5. Time cost of HBFC and original P2P for data lookup process within different number of requested files

Test case 2. The number of nodes and the number of requested files are fixed; the number of the files is changed

In this experiment, we initialized two sets of tests in which the number of requests (1500 requests) and the number of nodes (5000 nodes) are unchanged. The number of files in the system is increased from 10000000 to 30000000 files. Figure 6 illustrates the execution time of data lookup process for different number of files. The x-axis represents the number of files, and the y-axis represents the execution time. As we can observe, In fact, along with the increasing of the number of files in the system, the time costs in searching files remain approximately the same for both schemes namely T1 (123 ms), T2 (229 ms) respectively. However, T2 (229 ms) is still twice longer than T1 (123 ms) despite the fact that data lookup space is increased in terms of the number of files. It is concluded that the HBFC is more efficient than original P2P system in terms of scalability.

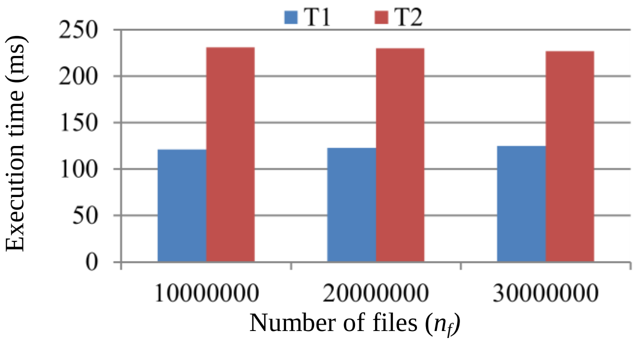

Fig. 6. Time cost of HBFC and original P2P for data lookup process within different number of files in the system

Test case 3. The number of the files and the number of requested files are fixed; the number of nodes is changed.

In the third case, we initialized two sets of tests in which the number of requests (1500 requests) and the number of files (10000000 files) are unchanged. The number of nodes in the system is increased from 5000 to 10000 nodes. Figure 7 illustrates the execution time in data lookup process for different number of nodes. The x-axis represents the number of nodes, and the y-axis represents the execution time. Generally, the execution time grows linearly with the increase of nodes in the system but in our proposed scheme there is a slight increase from 120 ms to 166 ms compared to original P2P system from 230 ms to 349 ms. Similarly, T2 (322 ms) is still approximately twice longer than T1 (154.5 ms) in this case.

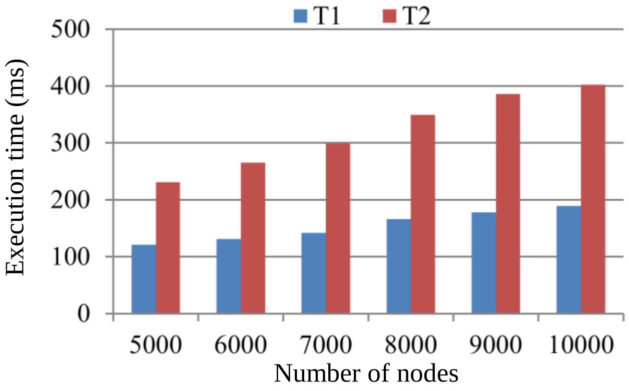

Fig. 7. Time cost of HBFC and original P2P for data lookup process within different number of nodes in the system

Overall, it is demonstrated that our proposed scheme is more efficient than the original P2P in all cases. Specifically, when the size of system is increased together with the increase in the number of requests, the executing time of original P2P system increases rapidly in comparison with that of HBFC.

VI. Conclusions

This paper discussed a novel approach to distribute and search data in P2P systems based on the specific utilization of hash functions. We proposed the HBFC to cluster most of the files in the system into defined nodes and enhance the searching process. Taking advantage of the hash functions, searching and accessing files are performed mainly on the defined number of peers rather than in the whole networks. The low computational complexity of hash functions enables us to achieve the performance efficiency in searching for requested files. Hence, the expected file distributions is achieved by hashing data keys with selected hash function to collided ID of the first designed ID space and then storing these files at a certain number of clusters through the one-to-one mapping between the first ID space and the second ID space. The simulation results demonstrate better feasibility, efficiency of the proposed scheme than the default schemes in original P2P systems. More importantly, the proposed scheme allows us to cluster files in a controlled way to suit the specifications of a real system.

References

[1] L. Schubert and K. Jeffery, "Advances in clouds," Report of the Cloud Computing Expert Working Group. European Commission, 2012.

[2] I. Iakovidis, "Towards personal health record: current situation, obstacles and trends in implementation of electronic healthcare record in Europe," International Journal of Medical Informatics, vol. 52, pp. 105-115, 1998.

[3] W. Tantisiriroj, S. W. Son, S. Patil, S. J. Lang, G. Gibson, and R. B. Ross, "On the duality of data-intensive file system design: reconciling HDFS and PVFS," presented at the Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, Seattle, Washington, 2011.

[4] 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," SIGOPS Oper. Syst. Rev., vol. 41, pp. 205-220, 2007.

[5] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," SIGOPS Oper. Syst. Rev., vol. 44, pp. 35-40, 2010.

[6] B. Dong, Q. Zheng, F. Tian, K.-M. Chao, R. Ma, and R. Anane, "An optimized approach for storing and accessing small files on cloud storage," J. Netw. Comput. Appl., vol. 35, pp. 1847-1862, 2012.

[7] G. Chen, T. Hu, D. Jiang, P. Lu, K. L. Tan, V. Hoang Tam, and S. Wu, "BestPeer++: A Peer-to-Peer Based Large-Scale Data Processing Platform," in Data Engineering (ICDE), 2012 IEEE 28th International Conference on, 2012, pp. 582-593.

[8] C. R. Valencio, M. H. Marioto, G. F. Donega Zafalon, J. M. Machado, and J. C. Momente, "Real Time Delta Extraction Based on Triggers to Support Data Warehousing," in Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2013 International Conference on, 2013, pp. 293-297.

[9] P. Xishui, Z. Xuezhong, S. Hongmei, Z. Runshun, and Z. Tingting, "Enhanced data extraction, transforming and loading processing for Traditional Chinese Medicine clinical data warehouse," in e-Health Networking, Applications and Services (Healthcom), 2012 IEEE 14th International Conference on, 2012, pp. 57-61.

[10] Y. Min and Y. Yuanyuan, "An Efficient Hybrid Peer-to-Peer System for Distributed Data Sharing," Computers, IEEE Transactions on, vol. 59, pp. 1158-1171, 2010.

[11] J. Altmann and Z. B. Bedane, "A P2P File Sharing Network Topology Formation Algorithm Based on Social Network Information," in INFOCOM Workshops 2009, IEEE, 2009, pp. 1-6.

[12] M. Yuqi, Y. Cuibo, M. Tao, Z. Chunhong, Z. Wei, and Z. Xiaohua, "Dynamic Load Balancing with Multiple Hash Functions in Structured P2P Systems," in Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on, 2009, pp. 1-4.

[13] D. Carra, R. Lo Cigno, and E. W. Biersack, "Stochastic Graph Processes for Performance Evaluation of Content Delivery Applications in Overlay Networks," Parallel and Distributed Systems, IEEE Transactions on, vol. 19, pp. 247-261, 2008.

[14] D. J. Bernstein. (2015). D. J. Bernstein. Available: http://cr.yp.to/djb.html. [Accessed: 02- Dec- 2015].

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," presented at the Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, San Diego, California, USA, 2001.

[16] JChord. (2016). JChord. Available: https://code.google.com/archive/p/joonion-jchord/downloads. [Accessed: 01- Jan- 2016].