DSMAV: An Improved Solution for Multi-Attribute Search Based on Load Capacities

ThanhDat NGUYEN, Claudiu Vasile KIFOR
Faculty of Engineering
Lucian Blaga University
Sibiu, Romania
ntdat@qnu.edu.vn, claudiu.kifor@ulbsibiu.ro

HoaiSon NGUYEN
Faculty of Information Technology
VNU University of Engineering and Technology
sonnh@vnu.edu.vn

***Abstract*— DHT (Distributed Hash Table) such as CHORD or PARTRY facilitates information searching in scalable systems. Two popular DHT-based approaches for range or multi-attribute search are to rely on attribute-value tree and a combination of attributes and values. However, tradeoff between a load balancing mechanism and query efficiency is a challenging task for such information searching systems. In this paper, we propose improved algorithms for a system called DSMAV in which information resources are distributed fairly among nodes and found based on multi-attribute queries in a small number of hop counts. Our system creates identifiers from resource names, each of which is a combination of attribute-value pairs (AV pairs). In order to overcome problem of load imbalance among nodes because of frequent appearance of AV pairs in many resource names, we control the quantity of information resources distributed to a node based on load capacity of each node. Moreover, periodical updating processes on a node enable our load balancing mechanism to be maintained accurately. As a result, our improvement evaluated by simulated results depicts a good degree of load balance as well as query efficiency.**

Keywords: DHT, multi-attribute search, load balancing, threshold values, load capacity.

I. Introduction

These days, information searching is a popular habit of many people. A person often update his or her knowledge by using connectable devices such as mobile phones, tablets and computers, or even smart televisions and refrigerators. Hence, information share and search systems with useful properties of scalability, efficiency and fault-tolerance are necessary for the requirement. Recently, DHT-based approaches such as CHORD [2], and PASTRY [3] offer promising solutions for the systems such as usage of Chord in Wireless Mess Network [1] or a heavily modified OpenChord version in Online Social Networks [12], in which information content can be distributed and searched exactly.

In particularly, a DHT-based CHORD network provides one operation [4]: given a value (i.e. piece of information or a representation of an information resource), it is mapped to a key by using a consistent hashing function. The value and key are sent to the node responsible for the key. In order to route messages, each node stores routing information of about O(log N) links to its neighbor nodes, where N is the number of nodes in the network and uses no more than O(log N) routing messages to find the location of any node responsible for a key. However, the operation is challenged by two problems. First, in DHT-based networks the node, which is responsible for a key, stores all values corresponding the key. If the key is frequently queried, the node may have a high load compared to others. This situation causes a load imbalance of DHT-based network. Second, the pure DHT-based information search system is challenged by complex queries in distributed manner such as range or multi-attribute queries.

In general, there are two typical solutions for the problems. The first approach is to base on a combination of AV pairs in order to limit the number of information resources in each node. In [5] [6], a content name is described by a set of AV pairs. One or several AV pairs are then combined to assign to DHT keys, which are used to locate nodes responsible for the content name. The second solution utilizes AV tree as a representation of an information resource [7] [8] [9] [10]. The main idea of the solution is that each strand of the AV tree is hashed to DHT keys. Information content is then distributed to target nodes that take responsibility for the keys. However, a tradeoff between load balance and query efficient is one of the challenges found in these solutions. Load imbalance caused by the existence of common AV pairs can be found in [7] [9], while a number of solutions [5] [6] [8] [10] suffer a high cost for maintaining load balance or executing queries.

In [11], we presented a solution for multiple-attribute search (SMAV). In the solution, information contents are distributed on the basis of DHT keys, which is created by one or a combination of AV pairs. The number of information contents stored in a node is limited by a pre-define value N_{max}. In order to find information contents, our system maintains key-to-CN and key-to-subkeys mappings. Hence, SMAV can keep a good degree of load balance. Yet, only one value N_{max} is applied to all nodes in a DHT network, where load capacity of each node is different in reality. Hence, the performance of SMAV depends on how N_{max} is chosen. The smaller N_{max}, the bigger the number of hop counts for forwarding query messages. In contrast, the bigger N_{max} the higher number of overloaded nodes.

In this paper, we propose an improvement for SMAV [11] in order to facilitate implementation and enhance a degree of load balance as well as query efficiency in a DHT-based system. Our improved solution called DSMAV enables to define various threshold values for nodes so that the number of content names does not exceed load capacity of each node. Moreover, it considerably limits the number of mappings

978-1-5090-1801-7/16/$31.00 ©2016 IEEE

stored in nodes and utilize improved algorithms during information distribution and search. Finally, we also present the result of simulations in more detail with the consideration of load capacity of nodes. This paper is organized as follows. We present our improved solution in Section 2. In Section 3, we present and evaluate our work and show the effectiveness of our work by simulation results. Related works are introduced in Section 4. Section 5 is to conclude our paper.

II. System Model

A. Overview

Our DSMAV system is built on top of a ring-based DHT network and implements our improved algorithms of information distribution and search.

DHT algorithm is used to define an n-bit ID/key space from 0 to 2^n-1. IDs are assigned to nodes while keys are assigned to representations of information resources. Thereby, Node IDs and keys reside in the same identifier space and each of them is the result of a consistent hashing function such as SHA (Secure Hash Algorithm). The use of the hashing function is necessary for load balancing purpose of the DHT-based network. A DHT-based protocol such as PARTRY [3] or CHORD [2] is implemented in order to organize nodes and route messages. Nodes are therefore arranged into a virtual ring-shaped network, and each node is responsible for storing a portion of the n-bit key space from its counter-clockwise neighbor node to its own identifier [10].

In order to support multiple-attribute queries, the proposed system represents information resources such as data files, text, description of products, or pieces of information as content names. A content name (CN) is a representation of an information resource and is formed from a set of AV pairs. For example, a CN of a laptop can be represented as follows: (Manufacturer = “XXX”, Model = “YYY”, CPU = “2.0GHz”, RAM = “4.0 GB”, HDD = “256 GB”), where Manufacturer, Model, CPU, RAM, HDD are attributes, and Manufacturer = “XXX” is a pair of attribute-value. This representation is not useful only to describe semantic of information resources but also to support multiple-attribute queries. The result of a query involving various AV pairs (e.g. Manufacturer = “XXX”, RAM = “4.0 GB”) must include all content names that contain all the AV pairs in the query.

An information resource is represented by a CN, which is a set of Attribute-Value pairs, CN = $\{(a_1, v_1), (a_2, v_2)... (a_m, v_m)\}$. Each AV pair $\{(a_i, v_i)\}$ is hashed into a key k_i by using a consistent hashing function, such as SHA-1.

We maintain a load balancing and query efficiency through hierarchical key generation approach and a threshold value of each node. Keys are organized in a hierarchical structure consisting of multiple levels of key such as level-1 keys, level-2 keys… level-h keys. A level-h key is hashed from h pairs of common AV pairs of a CN. For instance, a level-2 key is created by combining two common AV pairs (a_1, v_1) and (a_2, v_2) into a hashing function $h(a_1, v_1, a_2, v_2)$. A level-h key is generated if the total of CNs holding an AV pair exceeds the load capacity of a node. Thereby, the AV pair becomes a common AV pair, and its corresponding key is also called a common key. Moreover, we propose a formula (1) of a threshold value to limit the number of CNs (N_{CN}) of each key in a node to the threshold value of the node.

For a key, its common status is controlled by two flags (Flag and ComFlag) that are kept in a mapping table of keys and CNs (key-to-CNs mapping) in the node responsible for the key (Table 1). A key’s ComFlag is used to check if the key has ever been common one or not while Flag is used to check if N_{CN} of the key exceeds the threshold value of a node or not. A new key’s ComFlag and Flag are always initialized by “False”. If N_{CN} of the key is over the threshold value of the node responsible for the key then the key’s flags are changed to “True”. If ComFlag is “True” then the key is always a common key even though the key’s Flag can be changed to “False” in the process of periodical updating a node’s threshold value. ComFlag with “True” also reveals that mappings between key and level-h keys (h>1) have been created for the key. We supplement ComFlag in order to maintain integrity and accurateness of mappings in whole DSMAV system.

Simply, a node distributes a CN to a few of nodes in DSMAV system through keys called distribution keys. First of all, each AV pair split from a CN is hashed to a key. If the key is a common one, it would be combined with other keys to create a new distribution key. Otherwise, it is used as a distribution key. Then, the CN can be distributed to the nodes that take responsibility for the distribution keys. This means that several replications of the CN can be created and stored in whole DSMAV system. Information of a distribution key and CN, as well as a distribution key and other keys are stored in mapping tables of each node. In order to search the CN, a node must create a query name that contains at least one key corresponding to an AV pair in the CN and sends to the node responsible for the key.

If a key becomes a common key, the node responsible for the key can store more CNs than its load capacity. The node can be overloaded and several CNs can be lost because the overloaded node cannot store new CNs more. The node can cause problems of querying performance. Hence, in this article, we propose a formula of threshold values and improved algorithms of distribution and search based on the load capacity of every node. Our improved system can archive advantages as the follows:

A good degree of load balancing: A CN will be distributed calculatedly to a few of nodes. N_{CN} stored in a node is less than the node’s load capacity. The common AV pairs are stores in various nodes.

Efficient Search: Our improved system reduces significantly the lookup time by limiting the number of hop levels of a query. A query can find its all CNs at most three hops. Furthermore, because the number of overloaded nodes is diminished considerably, the rate of successful queries is high.

B. Formula of Dynamic Threshold Value

DSMAV system limits N_{CN} corresponding with each key at a node based on threshold values. A threshold value is specified for each node based on current status information of the node involving its *load capacity*, the number of *created keys* and N_{CN} stored in the node.

Assuming that N_p is the maximum number of CNs that a node is capable of storing (i.e. the load capacity of the node) and N_k is the number of keys created and stored in the node, we define a threshold value N_{th} for the node as follows:

$$N_{th} = \frac{N_p}{N_k} \quad (1)$$

When N_{CN} corresponding to a key stored in a node is over N_{th}, the key is considered as a common key. N_p can be specified based on the configuration of a node such as storage, bandwidth and the size of information resources.

By the use of a threshold value for a node, we can limit the total number of CNs stored in a node to be less than the load capacity of the node. For example, assume that a load capacity of a node A is 100; the number of created keys in node A is 10. Hence, node A's threshold value is 10. It means that each key managed by node A, can receive 10 CNs in maximum before the key becomes a common key.

The threshold value for a node may change frequently due to the change of the number of keys N_k stored in the node. Hence, we create flags to check the status of N_{th} corresponding to N_k and update N_{th} periodically based on the number of existed CNs corresponding to each key stored in a node. The flags are stored in mapping tables of each node.

C. *Updating Threshold Value Periodically*

In CHORD protocol, every node runs Stabilization process periodically in order to keep the node's successor pointers up to date. Basing on this progress of each node, we also update the threshold values and flags in the key-to-CNs mapping table in two steps.

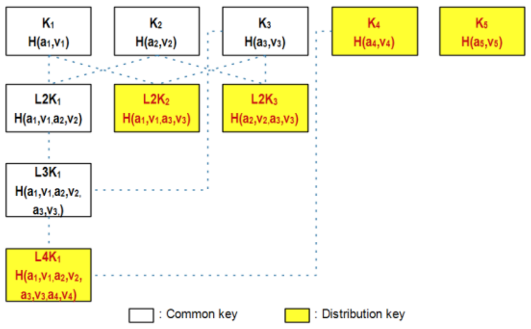
Step 1. A threshold value is updated based on formula (1). A periodical updating progress in a node is done as follows: Firstly, the updating node counts the number of keys N_k stored in the node. It also checks its key-to-CNs mapping table to find the maximum number N_{CNmax} of CNs corresponding to a key. Secondly, the node checks if $N_k * N_{CNmax}$ is over N_p or not. If yes, the threshold value is assigned by N_p/N_k. Since, the threshold value may be updated by a new value. Basing on the updated threshold value, step 2 is fulfilled.

Step 2. The flags in the key-to-CNs mapping table are updated. If N_{CN} of a created key is over the threshold value N_{th}, the key's Flag becomes "True". Otherwise, Flag is assigned by "False".

In a node, if a threshold value N_{th} is changed, the status of every common key can be changed. For example, assuming that a node's N_{th} is 10 and the node contains a common key k, which has been mapped to 14 CNs. After a periodical updating process, if N_{th} would be updated to 15, the common key k would become an uncommon one. Next, supposing that if N_{th} is updated to 12 after another periodical updating process, the uncommon key k becomes a common key again. This affects seriously the integrity of mappings as well as the query algorithm. Thereby, the common status of a key need to be kept correctly during updating a threshold value of each node.

D. *Improved Distribution Algorithm*

Assuming that an information resource represented by a content name CN need to be distributed to DHT-based network through a node called a distribution node. The distribution node first generates keys k_i (i=1..m) from each pair $\{a_i, v_i\}$. Improved distribution process is fulfilled in two steps (Fig. 1):



K_1 $H(a_1,v_1)$
K_2 $H(a_2,v_2)$
K_3 $H(a_3,v_3)$
K_4 $H(a_4,v_4)$
K_5 $H(a_5,v_5)$
$L2K_1$ $H(a_1,v_1,a_2,v_2)$
$L2K_2$ $H(a_1,v_1,a_3,v_3)$
$L2K_3$ $H(a_2,v_2,a_3,v_3)$
$L3K_1$ $H(a_1,v_1,a_2,v_2,a_3,v_3)$
$L4K_1$ $H(a_1,v_1,a_2,v_2,a_3,v_3,a_4,v_4)$
: Common key
: Distribution key

Fig. 1. Generation of distribution keys and mappings.

Step 1: First of all, the distribution node sends messages to nodes responsible for each k_i and check the common status of k_i. Information of k_i can be found in the node responsible for k_i as in Table 1. If k_i.Flag is "False", k_i is a distribution key, otherwise, it is a common one. Next, the distribution node sends content messages comprising CN and k_i, and key k_i-to-CNs mapping information to each node responsible for each distribution key k_i. If each distribution key is a level-2 key then mapping information of a level-1 key and the level-2 key is kept in the node responsible for the level-1 key. Otherwise, all mapping information of the level-2 key and level-h keys (h>2) is kept in the node responsible for the level-2 key. Mapping information of a level-1 key and level-2 keys or a level-2 key and level-h keys (h>2) are store in the key-to-level-h-keys mapping table (Table 2).

TABLE 1: KEY-TO-CNS MAPPING TABLE

Distribution key	**Information Resource**	**Content number**	**Flag**	**ComFlag**
K1	CN1, CN2, …, CN10	10	True	False
K2	CN1, CN 2	2	True	False
…	…	…	…	…

A node that receives a content message stores content of the message in its database. Mapping information of key k and CN is stored in the key-to-CNs mapping table and N_{CN} of the key k is increased. If N_{CN} exceeds the node's threshold value, the key's Flag and ComFlag are assigned to "True".

TABLE 2: KEY-TO-LEVEL-H-KEYS MAPPING TABLE

Key	**Level-h keys (h>1)**
K1	$L2K_1$
K2	$L3K_1$, $L4K_1$, …
…	…

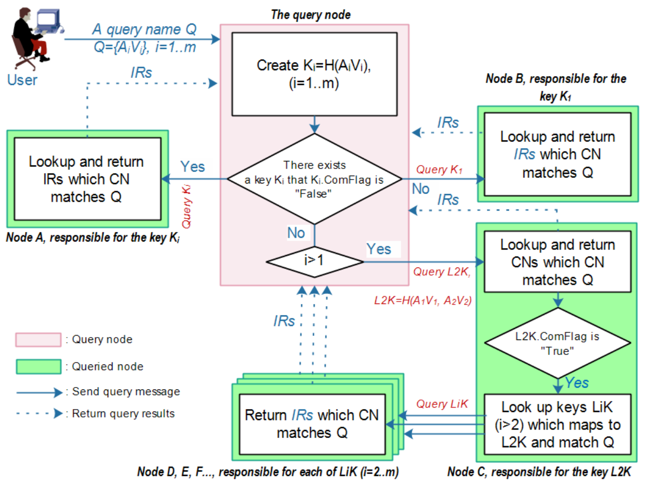
Step 2: If there exists a list of p_1 common keys (p_1<=m), there are two cases. In the first case (p_1>1), new level-h keys (h>1) are created by hashing h-1 common AV pairs with one of remaining common AV pairs from CN. Case 2 (p_1=1), if there exists a list of p_2 distribution keys (p_2=m-p_1, p_2>0), a new level-h key is created by hashing h-1 common AV pairs with one of remaining AV pairs, which N_{CN} is biggest. The first step is repeated with level-h keys until there is no common key.

Thereby, CNs and mappings are stored in the nodes that take responsibility for distributed keys.

As the Fig. 1 shown, K_4, K_5, $L2K_2$, $L2K_3$, $L4K_4$ are distribution keys, and K_1, K_2, K_3, $L2K_1$, $L3K_1$ are common keys. Level-3 key $L3K_1$ is created by using hashing function H((a1, v1),(a2, v2),(a3, v3)). Assuming that $L3K_1$ is a common key and N_{CN} mapped to K_4 is biggest compared with K_5. Then, $L4K_1$ is created by the function: H((a1, v1),(a2, v2),(a3, v3),(a4, v4)).

E. *Improved Lookup Algorithm*

In the improved lookup algorithm, we use the flag ComFlag of each key in order to determine the number of key-to-key mappings of each key. If a key's ComFlag is "True" then the key has ever been a common one with key-to-level-h-key mappings kept in the node responsible for the key. Thereby, a node can find all mappings of a key. The lookup process is improved as follows (Fig. 2):



The query node
A query name Q
Q={AᵢVᵢ}, i=1..m
User
IRs
Create Kᵢ=H(AᵢVᵢ), (i=1..m)
Node B, responsible for the key K₁
IRs
Lookup and return IRs which CN matches Q
Lookup and return IRs which CN matches Q
Yes
There exists a key Kᵢ that Kᵢ.ComFlag is "False"
Query K₁
No
IRs
Query Kᵢ
Node A, responsible for the key Kᵢ
No
Yes
i>1
Lookup and return CNs which CN matches Q
Query L2K,
L2K=H(A₁V₁, A₂V₂)
IRs
: Query node
: Queried node
: Send query message
: Return query results
L2K.ComFlag is "True"
Yes
Query LiK
Return IRs which CN matches Q
Look up keys LiK (i>2) which maps to L2K and match Q
Node D, E, F..., responsible for each of LiK (i=2..m)
Node C, responsible for the key L2K

Fig. 2. Block diagram of lookup process. IR: Information Resource, LiK: level-i key

A node called a query node initiates a lookup process with a query name that includes a set of AV pairs, q = $\{(a_1, v_1), (a_2, v_2)\ldots (a_m, v_m)\}$. A query message includes information of a query key, a query name, and the location of the query node. The query node first creates query keys k_i (i=0…m) from each AV pair and send query messages to the nodes responsible for each key k_i. Next, the query node checks the common status of each key k_i based on flag ComFlag of each key. If there exists a list of keys whose ComFlag is "False", the query node choose randomly a key in the list and send a query message to the node responsible for the key.

If all ComFlags of query keys are "True", the lookup process is fulfilled in two steps. First of all, if there are at least two AV pairs, a level-2 key would be created by two AV pairs that corresponding to the first common keys $\{(a_1, v_1), (a_2, v_2)\}$. Second, the query node sends query messages to the nodes responsible for each of the first common key and the level-2 key. The result of the query name is information resources including CNs returned from the nodes that receiving query messages (Fig.2).

A queried node, which receives a query message, searches resource information in its database including mapping tables. First, it looks up CNs that match a query name in its key-to-CNs mapping table. Then, the queried node returns the CNs found to the query node. Next, the queried node checks the common status of the query key. If the key's ComFlag is "True", it will search the query key's mappings in the key-level-h-key mapping table (h>1). If there exist the mappings of the query key, the queried node forwards the query message to the nodes responsible for the level-h keys.

III. Evaluation

In order to evaluate effectiveness of our improved solution, a simulation program that simulates three solutions of information distribution and searches as the follows:

- A conventional solution: Each distribution key is created from each AV pair in a content name. There is no level-h key (h>1) created. Therefore, the content name is distributed to nodes responsible for only the distribution keys. A process of a multiple-attribute query is performed based on a query key created by random choosing an AV pair in a query name.
- SMAV solution: The distribution of information resource are based on a pre-defined threshold value N_{max}. Key mapping of level-h key and level-h+1 key is kept by the node responsible for the level-h key.
- Our improved solution, called DSMAV.

We implement Chord [2] protocol in order to organize 5,000 nodes into a DHT-based network as well as to route messages. Nodes/keys are mapped into the same identified space with 228 = 268,435,456 identifiers. Every node with a load capacity N_p fluctuating from 200 to 500 is responsible for keeping a portion of 20,000 content names that are formed from 150,073 AV pairs. Each content name is a set of from 5 to 10 AV pairs chosen randomly. For SMAV solution, a pre-defined threshold value N_{max} is 10, while a threshold value of each node in DSMAV is defined by the equation (1). In all three solutions, the load capacity of every node is the same. If the number of content names exceeds the load capacity of a node, new content names are rejected by the node. Besides, commonality of every AV pair depends on a parameter called a rank r. Each AV pairs are generated based on the Zipf distribution and the formula: $\frac{1}{r^{\alpha}}$, where α is a constant.

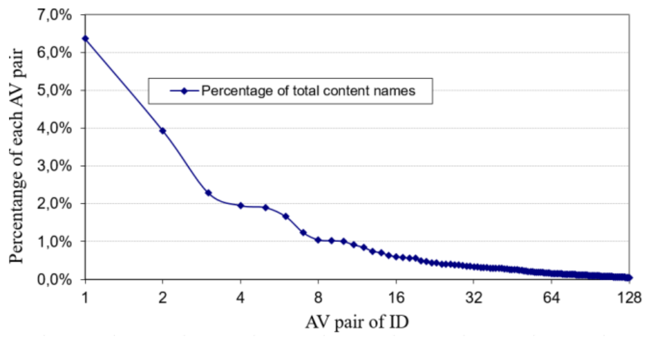


Fig. 3: The distribution of AV pairs in content names

Efficiency of our improved solution is evaluated based on a simulation experiment and comparable analysis between DMSAV with two the remaining solutions in two major aspects: a degree of load balancing and efficiency of information search algorithm that are discussed in the next sections.

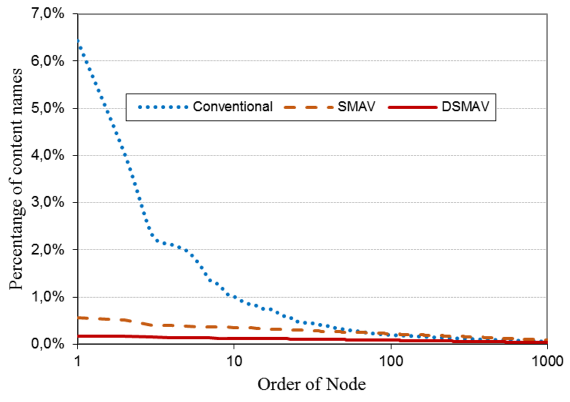


Fig. 4: Distribution of content names in 5,000 nodes

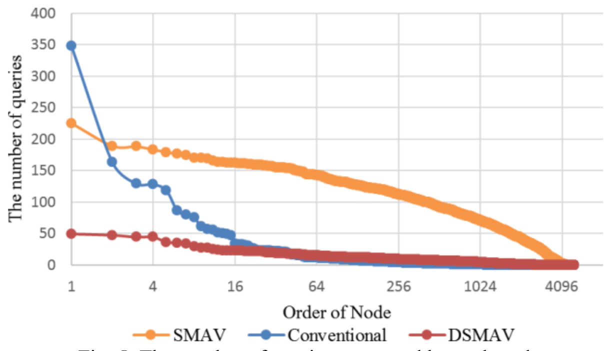


Fig. 5. The number of queries processed by each node

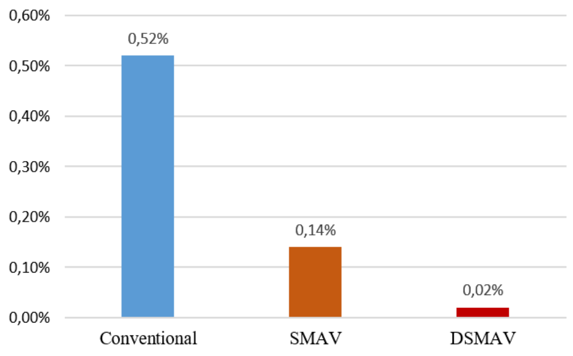


Figure 6: The rate of overloaded nodes

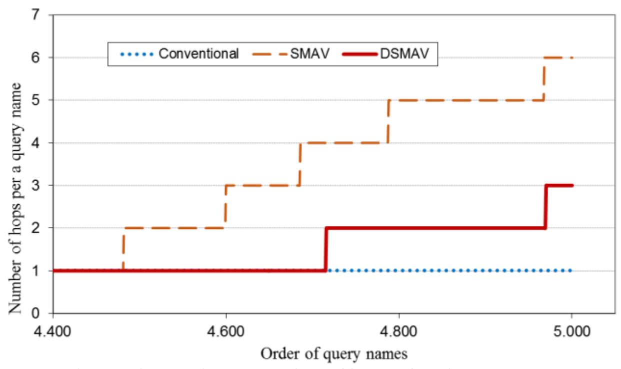


Fig. 7: The maximum number of hops of each query name

A. Load balancing

Load capacity of the system is evaluated based on the total of content names stored in each node, the total of queries processed by each node, and the number of overloaded nodes.

In our simulation network, 69,507 different AV pairs are found in 20,000 content names. The popularity of the each AV pair is different, some of the AV pairs occur at a high frequency. For example, as Fig. 3 shown that there are just over 128 AV pairs that occur with a frequency of well over 30 times per pair. Especially, we can see in Fig. 3 that an AV pair repeated by the highest frequency is 4,425 times, accounting for over 6.3% of total AV pairs. For the conventional solution, it is clear that the distribution of CNs stored in each node is similar to the popularity of AV pairs. More notably, over 10% of CNs are kept by only 10 nodes. Therefore, it is not surprising to find around 0.52% of nodes overloaded in Conventional solution compared to SMAV (0.14%) and DSMAV (0.02%) (Fig. 6). In the case of SMAV and DSMAV solutions, all CNs are distributed more equally. Only at most 0.6% of total CNs are kept by a node for SMAV while 0.2% of that for DSMAV (Fig. 4). However, while the distribution rate of CNs in SMAV can be changed depending on how N_{max} is pre-defined, DSMAV shows a more efficient distribution of CNs based on the formula (1). 1,133 threshold values N_{th} were updated periodically in the process of distributing CNs to nodes. Compared to SMAV and Conventional solutions, DSMAV solution creates a distribution of queries more equally (Fig. 5). Moreover, the number of queries carried out by each node is much lower compared with SMAV. This is reasonable because the number of level-h keys (h>1) created by SMAV is bigger than that created by DSMAV, 46,202 and 23,548 keys respectively. Fig. 5 also shows that the maximum number of queries carried out by each node in DSMAV solution is nearly 50 and much smaller than that in SMAV and Conventional solutions, 225 and 350 respectively.

B. Query Efficiency

The efficiency of information search in our simulation is evaluated based on the number of hop counts found for each query name in three solutions. The smaller the number of hop counts, the lower the query cost.

As presented in Fig. 7, the most striking feature is that the number of hop counts in DSMAV solution is not bigger than 3. The reason is because the number of nodes, which are responsible for forwarding query messages to other nodes, are limited considerably. Therefore, the query cost based on hop counts in DSMAV is lower than that in SMAV, 3 hops compared to 6 hops respectively. In the case of the conventional solution, the number of hop counts carried out by each query is only 1. Therefore, the cost of query time for the solution seem to be the best. Yet, a very low rate of successful queries caused by overloaded nodes is the biggest disadvantage that the Conventional solution has to pay (57.1%). In contrast, the number of successful queries in SMAV and DSMAV is very high, over 99% and nearly 100% respectively.

IV. RELATED WORKS

The problem of information distribution in a scalable system so that information can be found based on multi-attribute queries while ensuring a good degree of load balance as well as a low query cost has been getting an attention of researchers. Overall, two typical approaches are introduced in recent years.

The first significant approach is to base on AV trees to represent information resources. Balazinska and co-authors [7] present a system called INT/Twine to distribute and query intentional names to a DHT-based system. A noteworthy point in their system is that information contents are described as AV trees. Each unique prefix subsequence of AV pairs corresponds to a strand of an AV tree. The strands are then mapped to DHT keys, which are used to locate all nodes responsible for the information contents. Similarly, Garces-Erice and authors in [8] also introduce an AV tree-based approach in order to describe and store data in DHT networks. The authors' approach is to generate multiple indexes of data that contain key-to-key (or query-to-query) mappings and then distribute data to only one (or few) of the nodes responsible for a unique DHT key created from a part of an AV tree. In [9] a Resource Category Tree (RCT) is proposed in order to organize resources. Sun H. and the colleagues also describe resources as a distributed tree in which each node is a specific range of primary attribute values. Their solution allows performing commonly used queries including range queries or and multi-attribute queries.

Another notable approach for this research problem is to utilize a combination of AV pairs to create DHT keys. Gao and Steenkiste [5] have proposed a Content Discovery System (CDS) that enables to perform content registration and flexible queries based on the approach. Their method is to represent contents based on a combination of attributes and values and maintain a load balance by a logical matrix. Particularly, each set of AV pairs is a representation of a content name. AV pairs are assigned to DHT keys in order to locate the nodes responsible for storing content names. Thereby, a content name can be found based on only a DHT key chosen randomly in a set of candidates. Moreover, the authors resolve a problem of load imbalance because of an appearance of common AV pairs by constructing logical matrixes of nodes named Load Balancing Matrix (LBM) that responsible for common keys. In [6], Arakawa and co-authors also second a combination of AV pairs in their solution. The authors combine several AV pairs split from a content name to produce a content distribution key. The number of content names in a node is always limited to a constant value, and the number of AV pairs in a combination is just small enough. Their solution can ensure a load balance because of a presence of common keys.

Although mentioned solutions support range queries or multiple- attribute queries, they still have their own drawbacks. The first problem is load imbalance between nodes because of common keys. In the case of INT/Twine [7] and RCT [9], the appearance of common AV pairs is one of the major causes that result in a load imbalance between nodes. Moreover, sending the same information to various nodes as in INT/Twine system causes information redundancy. Another important problem is to concern with the costs of information distribution and query. In DHT-based systems such as CDS [5], Data Indexing [8], and [6], the problem of load imbalance is improved better. However, the high number of query steps, as well as an inefficient load balance mechanism raise costs for query execution or maintenance of load balance in the systems.

V. Conclusion And Future Works

In this paper, we proposed an improved solution for multiple-attribute search system in DHT-based networks. We have formed a formula that determines the threshold value of every node based on N_{CN}, created keys, and load capacity of the node. Although a periodical change of a threshold value can affect the common status of a key that can lose mapping information of a key, supplementing the flag ComFlag and improving distribution and lookup algorithms help our system to overcome the obstacle. Thereby, our system can keep integrity and accurateness of mappings. Moreover, the result of simulation shows that DSMAV system can archive a good degree of load balancing even though the distribution of AV is not equal. The result of a query name can be found in only three hops. Therefore, our improvement allows reducing the lookup time efficiently. However, the results of the paper are only evaluated on the basis of conventional, SMAV and DSMAV solutions, hence a comparative analysis between DSMAV and other solutions such as INT/TWINE and CDS is necessary. For the future work, we will implement algorithms of the solutions to our simulations in order to compare query and load balancing efficiency of DSMAV to the solutions. Moreover, we will also implement DSMAV solution in a tested system and apply into a knowledge management system.

References

[1] P. P. M., Krishna, M. V., Subramanyam, & K. S. Prasad, "Investigation of Chord Protocol in Peer to Peer-Wireless Mesh Network with Mobility", in *WAS., ET., International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, 934-938, 2015.

[2] I. Stoica, R. Morris, D. Karger, M. Kaashoek and H. Balakrisnan, "Chord: A Scalable peer-to-peer lookup service for Internet applications," in *In Proceedings of ACM SIGCOMM'01*, Aug. 2001.

[3] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *In Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, Nov. 2001.

[4] R. Vijayalakshmi and S. M. Kumarasamy, "LOAD BALANCING WITH PARTIAL KNOWLEDGE OF SYSTEM IN PEER TO PEER NETWORK," *International Journal of Advances in Engineering & Technology,* Vols. 3(2), 422, 2012.

[5] J. Gao and P. Steenkiste, "Design and Evaluation of a Distributed Scalable Content Discovery System," *IEEE Journal on Selected Areas in Communications,* Jan. 2004.

[6] Y. Arakawa, H. Minami, M. Matsuo, M. Yamaguchi and H. Saito, "DHT based Peer-to-Peer Search System using Pseudo-candidate Key Indexing," *IEICE Transactions on Communications,* Vols. vol. J88-B, no. no. 11, pp. pp 2158-2170, Nov. 2005.

[7] M. Balazinska, H. Balakrishnan and D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery," in *In Proceedings of International Conference on Pervasive Computing*, 2002.

[8] L. Garces-Erice, P. Feebler, s. E. Bier, G. Purvey-Keller and K. Ross, "Data Indexing in Peer-to-Peer DHT Networks," in *In Proceedings of 24rd International Conference on Distributed Computing Systems*, 2004.

[9] H. Sun, J. Huai, Y. Liu, R. Buyya and ., "RCT: A distributed tree for supporting efficient range and multi-attribute queries in grid computing," *Future Generation Computer Systems,* no. 24(7), pp. 631-643, 2008.

[10] N. Hoaison, T. Yasuo and Y. Shinoda, "D-AVTree: DHT-Based Search System to Support Scalable Multi-Attribute Queries," in *IEICE Transactions on Communications*, 2014.

[11] N. HoaiSon, N. ThanhDat and P. ThiHue, "SMAV: A solution for multiple-attribute search on DHT-based P2P network," in *In Advanced Technologies for Communications, IEEE*, 2009.

[12] A., Disterhoft, & K., Graffi. "Protected chords in the web: secure P2P framework for decentralized online social networks", in *Peer-to-Peer Computing (P2P)*, IEEE International Conference on (1-5). IEEE. 2015