Performance Evaluation of Routing Algorithms for Distributed Key-Value Store Based on Order Preserving Linear Hashing and Skip Graph

Ken Higuchi
Graduate School of Engineering,
University of Fukui
Fukui, Japan
higuchi@u-fukui.ac.jp

Makoto Yoshida
Graduate School of Engineering,
University of Fukui
Fukui, Japan

Naoyuki Miyamoto
Graduate School of Engineering,
University of Fukui
Fukui, Japan

Kento Takehara
Faculty of Engineering,
University of Fukui
Fukui, Japan

Tatsuo Tsuji
Graduate School of Engineering,
University of Fukui
Fukui, Japan
tsuji@u-fukui.ac.jp

***Abstract*— In this paper, routing algorithms for the distributed key-value store based on order preserving linear hashing and Skip Graph are evaluated. In this system, data are divided by linear hashing and Skip Graph is used for overlay network. The routing table of this system is very uniform. Then, short detours can exist in the route of forwarding. In this paper, by comparison with an exact shortest route, it proved that the routing algorithm using detour is effective.**

Keywords—distributed key-value store; Skip Graph; linear hashing

I. Introduction

Managing a huge data is an important and difficult problem even now. A database system is a most popular solution to it. But the database system is difficult to handle data easily and quickly because it is too strict. This characteristic deteriorates the performance of it in the distributed environment. From this circumstance, distributed key-value store (Distributed KVS) has attracted attention, and is used for various services. The key-value store (KVS) is a scale-out easily store data model by limiting the format of the data in the key-value pair and limiting possible operations and functions. A distributed KVS is adaptable to large-scale data by storing data in a distributed manner to a plurality of nodes on the network.

Many existing distributed KVS such as Amazon Dynamo [1] and Apache Cassandra [2] use a consistent-hashing [3] for the key to divide and store date to nodes. Also, in order to reach the node to be processed in 1-hop forwarding, gateway nodes and client nodes have to hold the entire routing information to each node. Thus, it is necessary to share the routing information in a plurality of nodes and it requires expensive process when the routing information is changed. It cannot be said to be efficient to insert or delete computers (nodes) from it.

On the other hand, by using Distributed Hash Table (DHT) and Skip Graph [4], it is also conceivable to build distributed KVS. Chord [5] and Kademlia [6] are popular DHT. In these methods, queries are transmitted to the node in a multi-hop forwarding. Moreover, all nodes have the same function, and the node decides a node to forward in order to send the query to the target node of the query. That is, each node is not required to store entire routing information for all nodes, and has only a portion relating to itself. Entire routing information is held on a whole node set. In this method, when inserting or deleting a node, it is not necessary to update entire routing information in all nodes and it is only necessary to update the corresponding routing information in some nodes. Then, routing information is updated efficiently. However, on DHT systems that use the hash function to determine the data location, although excellent in load balancing, the order of the original key in nodes is often not preserved. Thus, for a range search, it is necessary to transfer the query to all nodes and the cost of the range search is very inefficient.

In contrast, Skip Graph[4] is an overlay network that is possible range search. Furthermore, there are some extensions of Skip Graph. Multi-key Skip Graph[7] is extended Skip Graph for efficient range retrieval. On the original Skip Graph, node can handle only one key, but on the Multi-key Skip Graph, node can handle multiple keys by using virtual nodes. By using this Skip Graph, the distributed KVS can be implemented. But the routing table becomes to be large because the routing table is necessary for each key. Then, the cost of the node insertion and node deletion is very high. Range-Key Skip Graph[8] is another extension of Skip Graph. It simplifies the routing table of Multi-Key Skip Graph. In addition, large-scale distributed KVS using Range-key Skip Graph has also been proposed[9]. However, it is not performed efficiency when node insertion and node deletion.

Distributed KVS based on Order Preserving Linear Hashing and Skip Graph[13][14] is another implementation of Distributed KVS. In this system, data is divided by order preserving linear hashing(OPLH) and its overlay network is Skip Graph. This system can easily insert and delete the node and the number of hops of the query forwarding is almost the same as that of [9]. Skip Graph in [13] is very uniform. By using this feature, detours exist in the route calculated by Skip Graph and short route can be re-calculated[14]. [14] proposed a routing algorithm using detours and evaluated by comparing other Skip Graphs. But it is not compared with the shortest route.

In this paper, we compare the routing algorithm in [14] with the shortest route by experiments. And the efficiency of the algorithm in [14] is proved.

This paper is organized as follows. Sec. II introduces the distributed KVS based on order preserving linear hashing and Skip Graph and its routing algorithms using detours. Sec. III shows the performance evaluation of these routing algorithms. And Sec. IV concludes this paper.

.

II. Distributed KVS Based on Order Preserving Linear Hashing and Skip Graph

A. Order Preserving Liner Hashing

Linear hashing[10] is a kind of dynamic hashing. It consists of a hash function, a bucket array (hash table), data buckets, and meta-information. Meta-information includes the hash level, the number of data buckets, the number of records, and a fixed threshold. The data bucket is labeled by the different non-negative integer that is less than the number of data buckets. These labels correspond to hash values, and each data bucket stored only the set of data whose hash values are its label. The bucket array keeps the addresses of data buckets in order of hash values. In general, the hash function is modulo by 2^i for hash level i.; i.e. the hash value is the i bits suffix of the bit pattern of the data.

In the process of data retrieval, the hash value of the target data is calculated by hash function (whose hash level is i), and the address of the data bucket storing the target data is searched from the bucket array. If the calculated hash value is not less than the number of data buckets, the data bucket corresponding to this hash value doesn't exist. In this case, the hash value is recalculated by the hash function whose hash level is $i-1$. Then the data bucket corresponding to recalculated hash value is searched for retrieval. When the ratio of the number of record to the number of the data buckets is greater than the threshold, the bucket array is expanded to have a reference to a new data bucket. The hash value of the new data bucket is the number of data bucket minus 1 and is greater than the other hash values. But if the number of data buckets is 2^i, the corresponding hash value of the new data bucket is already used. When the number of data buckets becomes to be greater than 2^i for hash level i, the hash level is increased by 1. Then, on the linear hashing whose hash level is i, the hash level corresponding to the data bucket is i or $i-1$.

Order-preserving linear hashing[11][12] (OPLH) is a linear hashing which uses special hash function. It employs a combination of division function and bit reversal function. This function outputs the reverse ordered bit string of the result of the division function. For examples in hash level is 3, the hash value of 010101 is 010 and the hash value of 011000 is 110.

On OPLH, the expansion method and the reduction method for the bucket array are the same as that of the linear hashing using modulo function. When hash level is incremented, the old hash values are added “0” as the prefix and the hash value of the new element of the bucket array is greater than the other hash values. Then the order-preserving linear hashing inherits the advantage of the traditional linear hashing with good response time for the range query.

B. Skip Graph

Skip Graph[4] is a distributed data structure, based on skip lists. It is a kind of structured overlay network, and it supports range retrievals. It is suited to peer-to-peer networks. Unlike DHT, Skip Graph doesn't use hash function to decide to data partitioning. On Skip Graph, data is clustered by the range of key similar to the sequence set of B$^+$tree and the order of keys is kept. Then, range retrievals are processed effectively.

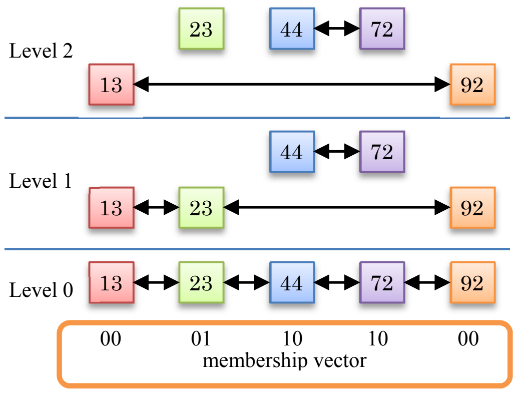

Figure 1. an example of Skip Graph

Fig. 1 shows an example of Skip Graph. In Fig. 1, the group of rectangles that have same number is a node and the number of the group of rectangles is the key of data that are stored by the node. Each rectangle represents an entry in the routing table and a link between a pair of rectangles expresses that it is valid entry. Each entry of the routing table belongs to one level of

Skip Graph. Each node has the randomized binary number which called *membership vector*. Membership vectors are used for the routing table of Skip Graph. In level 0 layer, nodes are sorted by keys and only adjacent nodes are connected. In other level i, rectangles are partitioned by the first i bit of the membership vector, only adjacent nodes in such partition are connected. Each valid entry of the routing table includes the node address and the key. Here, for the number of nodes *N*, the maximum level is $\lceil \log_2 N \rceil$.

On Skip Graph, the route from node A (whose key is *a*) to another node B (whose key is *b*) is decided as followings. Here, $a < b$ is assumed. Firstly the highest key that is not more than *b* and its node are searched from the routing table of A. Let such key be *c* and let C be the node whose key is *c*. Next, node A forwards the message to node C and C forwards this message to next node similarly. By repeating this forwarding, the message is reached to node B. Here, this routing algorithm used routing tables and key (*b*) can decide the route for exact queries and range queries. In the followings, the node that is received query by client is called the *start node* and the node that stores result data for the query is called the *target node*.

C. Distributed KVS

Distributed KVS based on OPLH and Skip Graph was proposed[13]. In this system, data is partitioned by OPLH and Skip Graph is used for overlay network. In other words, buckets of OPLH correspond to nodes. But hash values are not only used for data partitioning but also membership vectors and labels of nodes. Then this system has following features.

- The set of labels of nodes is that of consecutive non-negative integers started from 0.
- In level 0 layer, the nodes are not sorted by their stored keys.
- In level i, the node labeled a is linked to the nodes whose labels are $a \pm 2^i$ if exist.

Fig. 2 shows an example of Skip Graph based of OPLH. In Fig. 2, the binary number is the label of the node and also the hash value. Its length is the hash level. Each node has the label with the hash level.

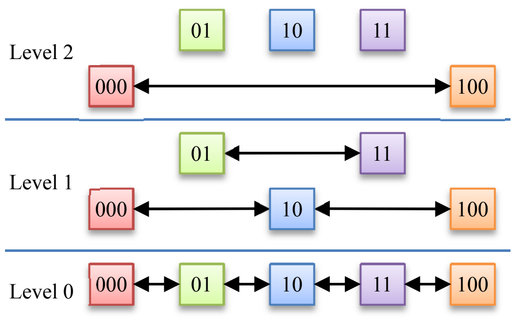

Figure 2. Skip Graph based on OPLH

D. Basic Routing Algorithm

In this system, the routing is similar to the binary search. In one routing, the message is not forwarded twice or more using same level of Skip Graph. Then, maximum number of hops of the message forwarding is not more than the maximum level of Skip Graph +1 ($\lceil \log_2 N \rceil$ for the number of nodes *N*). But for retrieval, the calculating the hash value is difficult on this system. In OPLH, the hash value is calculated by using hash level and the number of buckets. Then, if some node wants to calculate the correct hash value, the node has to use the hash level and the number of nodes. But these are global information and the node cannot use then in many cases. In order to solve this problem, this system uses the following revision.

- Let L_{max} is the maximum label in the routing table of the start node. $l = \lceil \log_2(L_{max} + 1) \rceil$ is used as the hash level.
- If the calculated hash value is more than L_{max} (it means that $L_{max} + 1$ is used as the number of nodes), the hash value is recalculate using hash function whose hash level is $l - 1$.

Here, this hash level l is called *improvised hash level* of the start node. But by using the improvised hash level and $L_{max} + 1$ as the number of nodes, the calculated target node is not the proper target node. In this situation, additional 1-hop is necessary (see [13] for details).

E. Detour

The routing algorithm described in Sec. II.D is based on Skip Graph. But the routing table of [13] is very uniform. By using this feature, there exists the shortest route[14]. It's called *detour*. Its concept is $a + 2^i + 2^{i-1} + \cdots + 2^{i-j} = a + 2^{i+1} - 2^{i-j}$ for node label a. Fig. 3 shows this detour. In Fig. 3, the red route (3 hops) is calculated by Skip Graph and the blue route is the short detour (2 hops). By using this detour, the number of hops from the start node to the target node can be reduced.

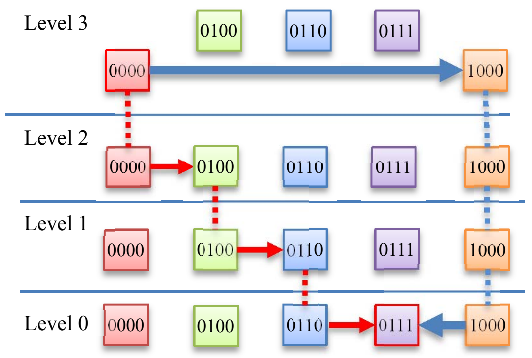

Figure 3. The detour from 0000 to 0111

In order to define the detour, some notations and definitions are introduced. Let $a_i \rightarrow a_j$ be the direct route from the node labeled a_i to the node labeled a_j, and the concatenation of direct routes is presented as $a_{n+1} \rightarrow a_n \rightarrow \cdots \rightarrow a_1 \rightarrow a_0$. This is route from the node labeled a_{n+1} to the node labeled a_0. Let $(a_{n+1}, a_n, \ldots, a_1, a_0)$ be the route $a_{n+1} \rightarrow a_n \rightarrow \cdots \rightarrow a_1 \rightarrow a_0$ calculated by Skip Graph. Here, $a_{i+1} \rightarrow a_i$ means that the node labeled a_{i+1} has the valid entry of the routing table to a_i at level i and the forwarding is occurred by using this entry. Here, n is the improvised hash level of the node labeled a_{n+1}. In some routes calculated by Skip Graph, it holds that $a_{i+1} = a_i$. In this case, the node labeled a_{i+1} doesn't forward to another node using level i entry of the routing table. Let $(b_n, b_{n-1}, \dots, b_1, b_0)$ be *a direction vector* of the route $(a_{n+1}, a_n, \dots, a_1, a_0)$ where for any $0 \le i \le n$, $b_i \in \{1, 0, -1\}$. Here if $a_{i+1} \to a_i$ is increasing direction then $b_i = 1$, if $a_{i+1} \to a_i$ is decreasing direction then $b_i = -1$, and if $a_{i+1} = a_i$ then $b_i = 0$. Then

$$a_{n+1} + \sum_{i=0}^{n} b_i \times 2^i = a_0$$

because of the features of the distributed KVS based on OPLH and Skip Graph in Sec. II.c. From above equation, even if the applying order of levels in the routing table, direction vector is not changed and the query can be forwarded from the start node to the target node (of course, the route is changed.)

The detour is found in the direction vector. If the direction vector has followings sub-part, the short detour exists. Here $b_{-1} = 0$ formally.

- There exist i and j $(i > j + 2, j \ge -1)$ such that $b_i = b_j = 0$ and for any k $(i > k > j + 1)$, it holds that $b_k = b_{k-1} \in \{1, -1\}$.

By changing as following respectively, the direction vector becomes the detour.

- $b_i = b_{i-1}$, $b_{j+1} = -b_{i-1}$ and for each k $(i > k > j + 1)$, $b_k = 0$.

But above detours is not always correct because such detour forwards via the node that does not exist. Then, the following condition is added to find the detour. Here, L_{max} is the maximum label in the routing table of the start node.

- $0 \le a_{i+1} + b_{i-1} \times 2^i \le L_{max}$ or $0 \le a_{i+1} - b_{i-1} \times 2^j \le L_{max}$.

Furthermore, since two or more detours exist in one direction vector, repeating to check of above mentioned conditions in the direction vector is necessary.

F. *Routing Alogorithm for the Detour*

In original Skip Graph, the route from the start node to the target node is decided by only key of the query condition. But in order to use the detour, the routing algorithm has to be changed as following.

1. Calculate the direction vector in the start node.
2. Calculate the detour and change the direction vector in the start node.
3. Select the leftmost element of the direction vector what the corresponding entry of the routing table is valid. Then, the destination node of the forwarding is decided by this element.
4. Forwarding to the destination node and change the corresponding element of the direction vector to 0.
5. Till reaching to the target node, repeat 3.and 4. in the node that received the forwarded message.

G. *Detour-Finding Alogorithm*

Here, one important matter is that "a detour makes new detour". In [14], this matter is not discussed. Consider the detour for the route from the node labeled 000000 to the node labeled 011011. The direction vector calculated by Skip Graph becomes $(0,1,1,0,1,1)$. $(1,0,-1,1,0,-1)$ is calculated directly by the definition of the detour. But, calculating from low level and using the modified direction vector, another detour can be calculated. Firstly, $(0,1,1,0,1,1)$ is changed to $(0,1,1,1,0,-1)$. Next, $(0,1,1,1,0,-1)$ is changed to $(1,0,0,-1,0,-1)$. Since the number of hops in $(1,0,0,-1,0,-1)$ is less than that of $(1,0,-1,1,0,-1)$, it is clear that detour calculating method is important. Fig. 4 shows these detours. Here, the blue route is $(0,1,1,0,1,1)$, the red route is $(1,0,-1,1,0,-1)$, and green route is $(1,0,0,-1,0,-1)$.

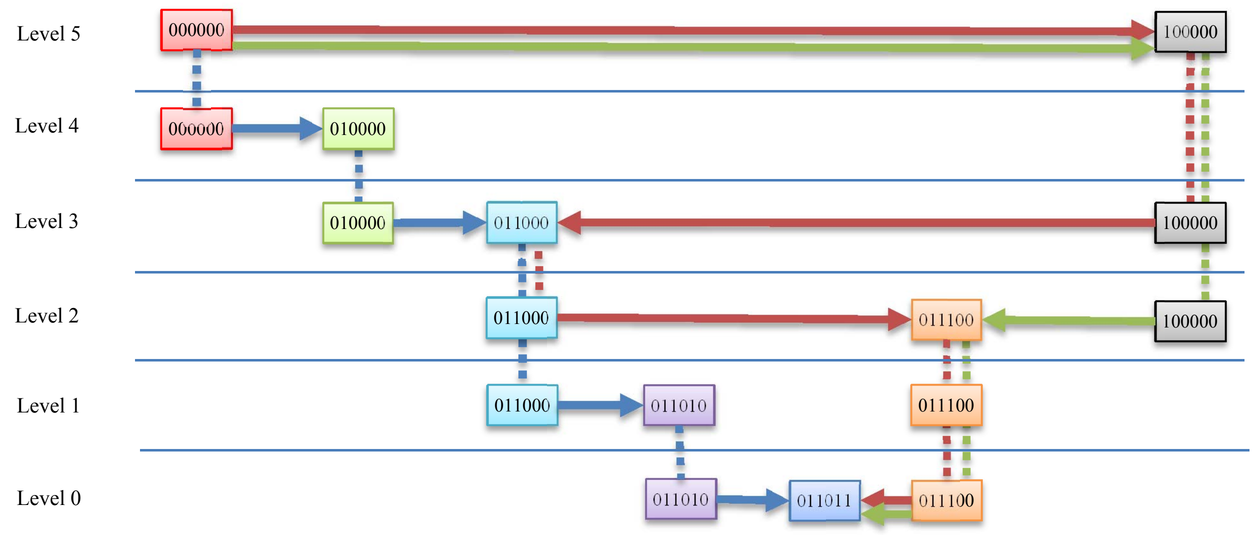

Figure 4. The detour from 000000 to 011011

The detour-finding algorithm of [14] uses this method. Here, this algorithm is $O(\log_2 N)$ because the length of the direction vector is $\lceil \log_2 N \rceil$ and the check to each level in the direction vector is only 1. But it is not proved that this route is shortest. If the system can uses very long time, the shortest route can be calculated. It is restricted problem of the shortest path problem of graph theory and $O(N \log_2 N)$. Furthermore, another problem exists in this algorithm. It is that the length of the direction vector is fixed to the improvised hash level of the start node. The global hash level is equal to the improvised hash level or the improvised hash level plus 1. Then, by changing the length of the direction vector to the improvised hash level plus 1 and verifying the correctness of the detour, shorter route may be founded. In next section, these 4 types of the detour-finding algorithms are compared and evaluated by experiments.

III. Experiments

A. Experiment Environment

To show effectiveness of the detour-finding algorithm, we evaluate the number of hops of the query forwarding and compare 4 types of finding algorithms shown in Table. I.

Table I. detour-finding algorithms for evaluation

Name	Explanations
LHSG	Original routing algorithm in [13] (not using detours)
DE0	Using directly the definition of the detour
DE1	Routing algorithms in [14] (using the modified direction vector to find detours)
DE2	Using the modified direction vector to find detours and changes the length of the direction vector
MIN	Shortest route found by breadth first search only using information in the start node.

Since we evaluate only the number of hops of the query forwarding, it is calculated on a simulator. The bit length of key is 32 and only exact retrievals are the target for evaluation, because a range retrieval is considered as the set of exact retrievals. In experiments, all condition (start node and key) are used for queries to evaluate. They are is not random choice and all combinations of all start nodes and all 32bit integers for key are used for query condition. By changing the number of node (from 10 to 10000), performance of detour-finding algorithms are evaluated.

B. Results of Experiments

Table II shows the average of the calculation times in case that the number of nodes is 2000. Fig. 5 shows the average of the number of hops in exact retrievals. MIN cannot be calculated over 2000 nodes because the calculating time is too long. Here, Level is the number of levels of Skip Graph. It means the upper bound of the number of hops. Therefore, the number of hops of the valid route decided by the start node is not more than LEVEL and not less than MIN. Then, the position of LHSG shows that LHSG is effective. Furthermore, in all situations, DE0 is less than LHSG and DE1 is less than DE0. And DE1 is equal to DE2 and is approximately equal to MIN. Then, it can be said that the improvised hash level is enough to use for the length of the direction vector. Furthermore, even if the number of nodes is less or equal to 2000, DE1 can find the shortest route in most case. From Table II, MIN needs long time to calculate and DE1 is shorter than DE2. Then it can be said that DE1 is best algorithm to calculate the route.

Table II. the average of the calculation time (2000 nodes)

Algorithm	time(sec)
LHSG	6.382×10^{-7}
DE0	8.302×10^{-7}
DE1	8.713×10^{-7}
DE2	9.127×10^{-7}
MIN	2.377×10^{-4}

IV. Conclusions

By experimental results, it expected that the detour-finding algorithm in [14] can be calculated effectively good route. But in some case, it cannot find the shortest route. It is future work. And other future works, we need to plan a routing algorithm suitable for replication.

References

[1] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshal, P., and Vogels, W., “Dynamo : Amazon's highly available key-value store”, ACM SIGOPS Oper. Syst. Rev. Vol41, No.6, pp.205-220, Dec., 2007

[2] Lakshman, A., Malik, P., “Cassandra - A Decentralized Structured Storage System”, ACM SIGOPS Oper. Syst. Rev. Vol42, No.2, pp.35-40, Apr., 2010.

[3] Karger, D. R., Lehman, E., Leighton, F. T., Panigrahy, R., Levine, M. S., and Lewin, D. “Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web”, ACM Symposium on Theory of Computing, pp.654-663, 1997.

[4] Stoica, I., Morris, R., Karger, D.,Kaashoek, F. and Balakrishnan, H. “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, ACM SIGCOMM, pp.149-160, Oct., 2001.

[5] Maymounkov, P. and Mazieres, D. “Kademlia: A Peer-to-peer Information System Based on the XOR Metric”, 1st International Workshop on Peer Systems (IPTPS’02), Mar., 2002.

[6] Aspnes, J., and Shah, G. “Skip Graphs”, ACM Transactions on Algorithms, Vol.3, No.4, pp.37:1-37:25, Nov., 2007.

[7] Konishi, Y., Yoshida, M., Takeuchi, S., Teranishi, Y., Harumoto, K., Shimojo, S., “An Extension of Skip Graph to Store Multiple Keys on Single Node”, Jornal of IPSJ, Vol.49, No.9, pp.3223-3233, Sep, 2008

[8] Ishi, Y., Teranishi, Y., Yoshida, M., Takeuchi, S., Shimojo, and Nishio, S., “Range-Key Extensions of the Skip Graph”, Proc. of IEEE 2010 Global Communications Conference (IEEE GLOBECOM 2010) , pp.1-6, Dec., 2010

[9] Ishi, Y., Teranishi, Y., Yoshida, M., and Takeuchi, S., “An Implementation of Large Scale Distributed Key-value Store with Range

Search Feature Based on Range-key Skip Graph" , Journal ot IPSJ, Vol.53, No.7, pp1850-1862, Jul., 2012.

[10] Litwin, W., "Linear hashing: new tool for file and table addressing", Proc. of 6th Conference on Very Large DataBases, pp.212-223, 1980.

[11] Robinson, J. T., "Order preserving linear hashing using dynamic key statics", Proc. of the 5th ACM SIGACT-SIGMOD symposium on Principles of database systems, pp.91-99, 1985.

[12] Higuchi, K., Tsuji, T., "A Distributed Linear Hashing Enabling Efficient Retrieval for Range Queries", Proc. of IEEE SMC 2010, pp. 838-842, Oct., 2010.

[13] Yoshida, M., Higuchi, K., Tsuji, T., "An Implementation and Evaluation of Distributed Key-Value Store Based on Order Preserving Linear Hashing and Skip Graph", Jornal of IEICE, Vol.J89-D, No.5, pp.742-750, 2015.

[14] Higuchi, K., Yoshida, M., Miyamoto, M., Tsuji, T., "A Routing Algorithm for Distributed Key-Value Store Based on Order Preserving Linear Hashing and Skip Graph", In Roger Lee (Eds.), *Applied Computing & Informaiton Technology (Studies in Computaitonal Inteligence Vol. 619)*, pp.127-140, Springer, 2016.

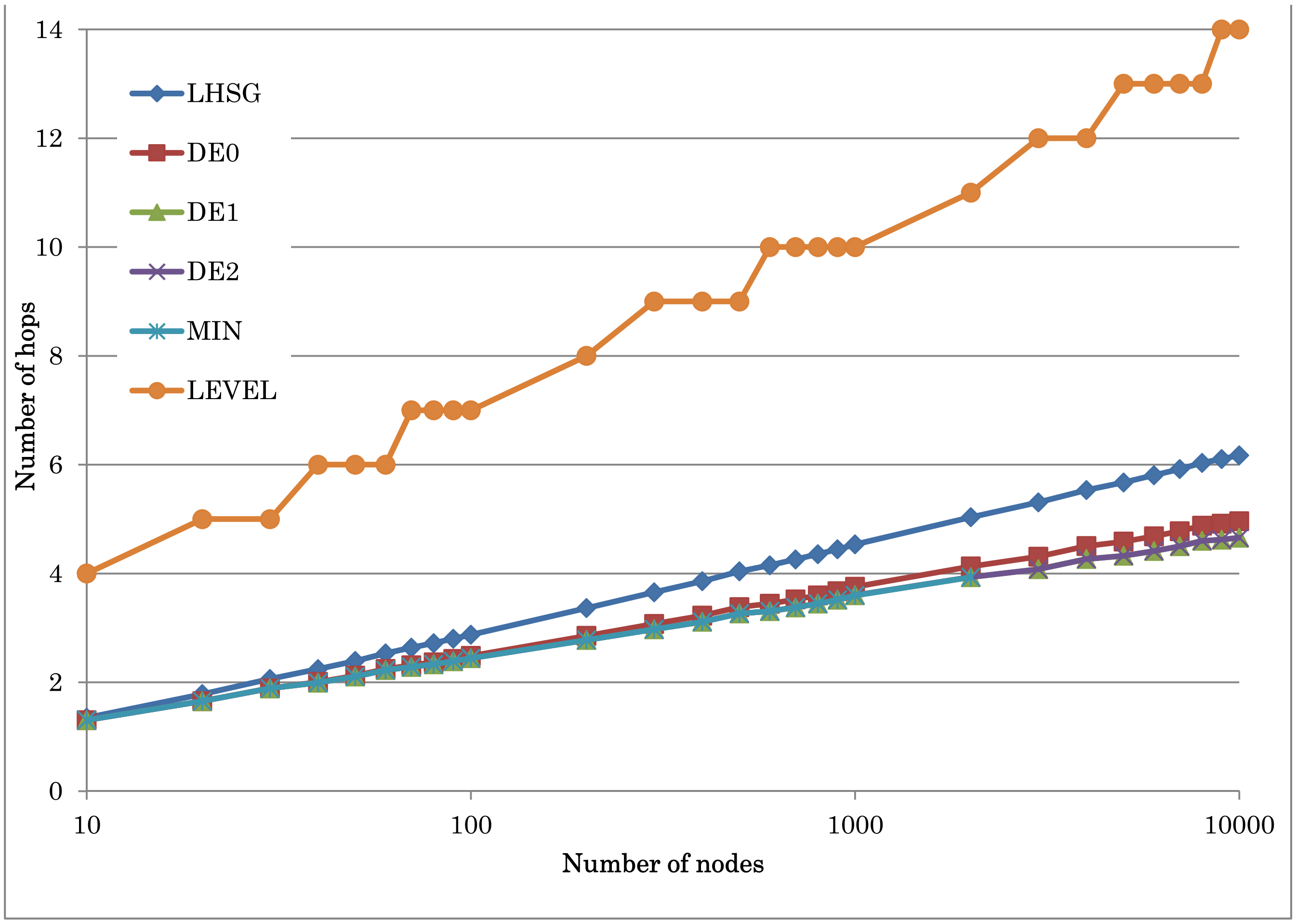

Figure 5. The average of the number of hops for the exact retrieval