Skip Suffix Array: A Partial Match Retrieval Method on Structured Overlay Networks

Ryohei Banno and Kazuyuki Shudo
Tokyo Institute of Technology, Tokyo, Japan
Email: banno@computer.org

***Abstract*—Structured overlay networks provide superior scalability and robustness suitable for large scale distributed systems. However, typical algorithms' ability to handle complex queries is limited; consequently, their range of applications is also limited. We propose a partial match retrieval method for structured overlay networks that we refer to as Skip Suffix Array (SSA). SSA integrates the concept of suffix arrays with Skip Graph, which is a distributed range-queriable data structure. In addition, we introduce a unique routing method for delivering partial match queries. SSA achieves lower communication cost and better load balancing compared to some existing methods for partial match retrieval, as well as its search time is suppressed to $O(\log N)$ where N is the number of nodes. Experimental results demonstrate that SSA can reduce the average path length and the average number of messages by more than 85% compared to existing methods, under experimental conditions with $10,000$ nodes and search words of length 8.**

***Index Terms*—Overlay networks, partial match retrieval, peer-to-peer networks, skip graph, suffix array**

I. Introduction

Structured overlay networks, a type of peer-to-peer network, are promising techniques to obtain scalability and robustness in large scale distributed systems. Structured overlay networks are attracting significant academic and industrial interest due to their applicability to data storage [1], publish/subscribe messaging [2] and Internet of Things (IoT) systems [3].

One of the challenges in structured overlay networks is handling complex queries. We focus on partial match retrieval which is required to realize flexible search in data storage and IoT systems, flexible filtering in publish/subscribe messaging, etc. Although typical algorithms for structured overlay networks, such as Chord [4] and Pastry [5], cannot achieve partial match retrieval, some existing methods can [6], [7]. However, these existing methods have some inefficiencies including large communication costs and load imbalance.

In this paper, we propose Skip Suffix Array (SSA), a method that enables partial match retrieval on structured overlay networks. SSA integrates the concept of suffix arrays [8], a data structure for string searches, with Skip Graph [9], a distributed range-queriable data structure. In addition, we extend a state-of-the-art routing algorithm for Skip Graph to deliver partial match queries to corresponding nodes efficiently. SSA achieves lower communication costs and better load balancing compared to existing methods. In addition, its search time is suppressed to $O(\log N)$, where N is the number of nodes.

This work was supported in part by SECOM Science and Technology Foundation, in part by New Energy and Industrial Technology Development Organization (NEDO), and in part by JSPS KAKENHI Grant No.19K20253.

The primary contributions of this paper can be summarized as follows.

- We show the basic design of the proposed SSA method that integrates suffix arrays and Skip Graph.
- We present a routing method to deliver partial match queries.
- We clarify the characteristics and effectiveness of SSA by analytical and experimental comparisons with existing methods.

Note that this paper is an extended version of our previous work [10], which reported the basic idea (in Japanese). We extended the previous version by improving the routing protocol to reduce the path length, and we redesigned and reconducted experiments. Specifically, the differences include: proposal of an improved routing method (Section III-D), quantitative evaluation that includes a comparison with our previous method and more existing methods (Section V).

The remainder of this paper is organized as follows. Section II introduces related studies on partial match retrieval on overlay networks. Section III explains the proposed method. Analytical comparisons with existing methods are discussed in Section IV. Section V presents experimental evaluation. Conclusions and suggestions for future work are given in Section VI.

II. Related work

Several existing methods attempt to provide partial match retrieval on structured overlay networks.

Preliminary to discussing those methods, we define partial match retrieval by assuming the following.

- Each node of an overlay network has one or more **labels**, which are to search.
- A label is a sequence of characters, denoted $a_1a_2...a_L$ where the length of the label is L.
- A set of substrings of a label lb is denoted $S_{lb} = \{a_ia_{i+1}...a_j \mid 1 \leq i \leq j \leq L\}$.

Definition 1. Assuming S is a set of finite strings, **partial match retrieval** is defined as follows: if a node issues a query with a search word $s \in S$, the query is delivered to all nodes which have a label lb such that $s \in S_{lb}$.

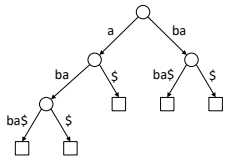
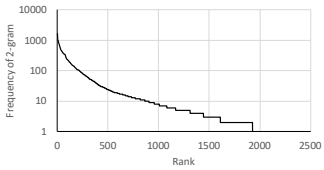
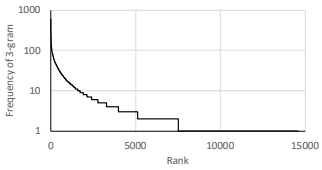


Fig. 1. Example of suffix tree



(a) 2-gram



(b) 3-gram

Fig. 2. Deviation of appearance frequencies of n-grams

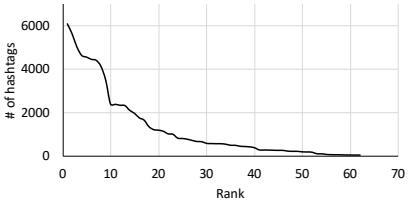


Fig. 3. Imbalance of matching hashtags among first characters

A. Existing methods

PIER [6], [11] is an approach to handle a partial match query as a combination of multiple exact-match queries. In a preprocess, each node divides its labels into n-grams. For example, a label "computer" is divided into seven substrings as 2-grams, i.e., "co", "om", "mp", "pu", "ut", "te" and "er". Subsequently, the node stores its identifier, e.g., IP address and/or the label, in a distributed hash table (DHT) [4], [5] with a key of each divided n-gram. When a node issues a partial match query, its search word is also divided into n-grams. For each n-gram, the issuer node retrieves a set of node IDs from the DHT using the n-gram as a key. Then, the node counts the number of appearances of each node ID in the obtained sets. If there are node IDs such that the number is the same as the number of n-grams of the search word, they are considered as nodes having matching labels. Finally, the issuer can deliver the query to the corresponding nodes.

DST [7] constructs a suffix tree, a data structure capable of partial match queries, over a DHT. Fig. 1 shows an example of a suffix tree for the string "ababa". Substrings are mapped to each edge such that a sequence of substrings from a root edge to a leaf edge represents a single suffix. The character "$" indicates the end of the suffixes. In DST, each root edge of the suffix tree is stored in the DHT with a key of the first character of its corresponding substring. Stored data includes identifiers of child edges. Edges besides root edges are also stored in the DHT with keys of IDs given by their own parent edge. On each leaf edge, IDs of nodes whose labels have the corresponding suffix are stored. When a node issues a partial match query, it first retrieves a root edge from the DHT by specifying the first character of the search word as a key. Thereafter, by retrieving the corresponding child edge on the DHT recursively, it can obtain IDs of nodes holding matching labels if they exist.

B. Inefficient aspects of existing approaches

Both PIER and DST have inefficiencies from some perspectives.

1) Load imbalance: One matter of concern is load imbalance. Namely, they are heavily affected by deviations in the appearance frequencies of keys.

In PIER, n-grams are used as keys in a DHT. Generally, the appearance frequencies of n-grams are largely biased. Fig. 2 (a) and Fig. 2 (b) show the deviation of the appearance frequencies of n-grams, using $10,000$ English hashtags obtained from Twitter[1]. Note that the average length of these hashtags was 10.154, whereas the number of 2-grams and 3-grams were $2,442$ and $14,587$ respectively. As can be seen, only a few n-grams appear hundreds or thousands of times, while most appear less than 10 times. Since all node IDs corresponding to an n-gram are stored on a single node in PIER, such deviation is directly reflected in the load of each node.

Similarly, DST also involves load imbalance. There is a limited number of root edges, and the appearance frequencies of their first characters are biased. Fig. 3 shows the number of corresponding hashtags for each initial character. Obviously, a node responsible for a character of high rank tends to have a heavier load because every query begins from one of the root edges.

2) High communication costs: PIER performs a lookup in a DHT for each n-gram of a search word. Thus, one partial match query requires communication cost proportional to the length of its search word. Likewise, DST also requires a large number of messages due to its mechanism, i.e., a lookup in a DHT occurs for each transition from a parent edge to its child edge.

3) n-gram related issues: Since PIER utilizes n-grams, there could be false positive results. For example, the search word "sea" can match the label "ease". Such cases cause increased communication costs.

In addition, a search word must be equal to or longer than n of n-grams.

III. Skip Suffix Array

The proposed SSA partial match retrieval method can improve the inefficient aspects of the existing methods by integrating suffix arrays with Skip Graph.

A. Suffix array

A suffix array [8] is a sorted array of all suffixes of a string. For example, the suffix array of a string "computer" is shown in Table I. When a suffix array is given, we can find the position of an arbitrary substring efficiently by binary search on the suffix array.

B. Skip Graph

Skip Graph [9] is one of the algorithms for structured overlay networks, known to handle range queries effectively. Each

[1]The hashtags are obtained by using Twitter API on February 14th, 2019.

TABLE I
EXAMPLE OF SUFFIX ARRAY

Suffix	Position
computer	0
er	6
mputer	2
omputer	1
puter	3
r	7
ter	5
uter	4

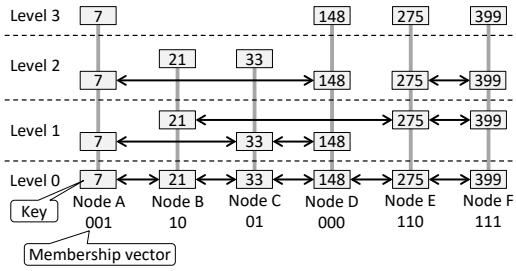


Fig. 4. Example of Skip Graph

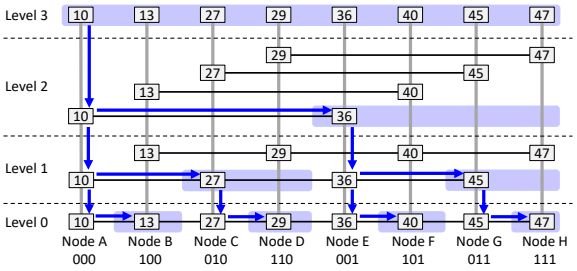


Fig. 5. Example of routing by SFB

node has a key and a random sequence called a membership vector. A membership vector consists of symbols contained in a finite alphabet Σ, which is typically $\Sigma = \{0, 1\}$. As shown in Fig. 4, Skip Graph creates a multiplex structure of a skip list [12]. Level 0 is a doubly linked list that consists of all nodes sorted in key order. In level i, each node composes a doubly linked list with nodes whose membership vectors have the same first i digits.

A node can issue a query by specifying a target key. The query is forwarded among nodes in the same manner as a skip list, i.e., it skips a long distance at higher levels and gradually moves down to level 0. The path length is $O(\log N)$ where the number of nodes is N. Note that the size of the routing table on each node is also $O(\log N)$.

Since nodes are sorted in key order, each node can also perform a range search; a query is delivered to nodes whose keys are included in the specified range. Methods for routing range queries in Skip Graph have been proposed [13], [14]. In SSA, we use SFB [14] due to its shorter path lengths and lower communication costs[2].

SFB utilizes a divide-and-conquer approach. Each node that receives a range query divides the target range into subranges based on the keys of its neighbor nodes within the range, and then forwards the query with each subrange to the corresponding neighbor node. As shown in Fig. 5, when node A receives a range query with a target range (depicted as a blue rounded rectangle at level 3) it first divides the range by the key of node E into two subranges. Subsequently, it forwards one of the two, the farther, to node E. Then, it repeats this process for the remaining subranges at lower levels recursively.

Hereafter, we refer to a key of Skip Graph as a "label" as described in Section I.

C. Topology of SSA

In SSA, we combine suffix arrays with Skip Graph to enable partial match retrieval.

First, each node divides its labels into suffixes. Then, every suffix is inserted in the Skip Graph as a virtual node. Although in the usual Skip Graph each node has its own membership vector, with SSA membership vectors are assigned for each physical node. Virtual nodes of the same physical node have the same membership vector. This enables the height to be suppressed, which is reflected in the size of routing tables, to $O(\log N)$ where N is the number of **physical** nodes [15].

[2]In our previous work [10], we used MRF [15]. The difference between SFB and MRF is disscussed in [14].

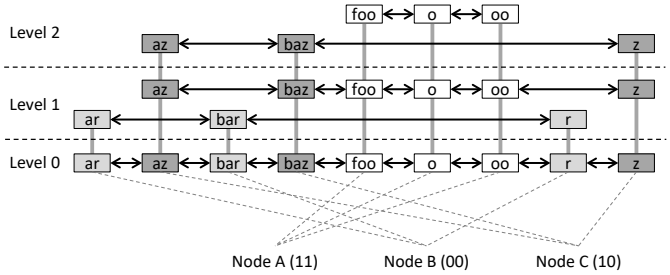


Fig. 6. Example SSA topology

Fig. 6 shows an example of SSA topology. In this topology, there are three physical nodes, A, B, and C labeled "foo", "bar", and "baz", with membership vectors 11, 00, and 10 respectively. As can be seen, the level 0 list forms a suffix array such that all physical nodes having a label that corresponds to one substring are placed continuously. Therefore, we can perform partial match retrieval efficiently by range search. For example, if a partial match query is issued with the search word "ba", it is handled as a range query where the range includes keys starting with "ba". This query is eventually delivered to two virtual nodes, i.e., "bar" of node B and "baz" of node C.

Note that since SSA utilizes the topology maintenance mechanism of Skip Graph, methods aiming at the improvement of the topology can be applied [16]–[18]. By using these methods according to the characteristic of each application such as the locality of node placement, better performance e.g., shorter path lengths might be obtained.

D. Routing in SSA

As mentioned previously, we use SFB [14] in SSA. Although SFB achieves a shorter average path length than other routing methods [13], [15], utilizing it in SSA can cause inefficient routing paths because SFB does not assume the use of virtual nodes.

Considering virtual node VN_1 attempts to forward subrange SR_1 to virtual node VN_2 and subrange SR_2 to virtual node VN_3, the possible undesirable cases are described as follows.

(i) VN_2 or VN_3 is of the same physical node as VN_1. For example, if nodes A and G in Fig. 5 are of the same

physical node, the physical node receives the same query twice.

(ii) VN_2 and VN_3 are of the same physical node. For example, if nodes C E in Fig. 5 are of the same physical node, the physical node receives the same query twice.

(iii) A virtual node of the physical node of VN_2 is contained in SR_2. For example, if nodes B and F in Fig. 5 are of the same physical node, the physical node receives the same query twice.

To avoid these cases, we propose Multi-Key SFB (MK-SFB) which extends SFB by aggregating the query processing and forwarding as like Multi-key Skip Graph [15]. MK-SFB is designed to work effectively under the assumption that each physical node may have multiple keys (i.e., multiple virtual nodes).

When a node receives a range query, the node first confirms whether the range contains virtual nodes of the same physical node. If such virtual nodes exist, it divides the range by the keys of the virtual nodes and delegates subranges to the nodes as internal processes of the physical node. This avoids case (i).

Then, each of the above virtual nodes divides the subrange it is responsible for into smaller subranges according to SFB, but not yet forwards the subranges. Inside the physical node, subranges to be forwarded to the same physical node are aggregated and forwarded collectively to the physical node. This avoids case (ii).

In addition, when each virtual node divides a subrange, it can consider "sibling" virtual nodes of its neighbor virtual node. For example, let nodes B and F be of the same physical node in Fig. 5. Since node B is a neighbor of node A, node A can obtain the label of the physical node of node B. Therefore, node A can easily know the existence of node F by considering the label's suffixes. In this case, node A divides the subrange into four smaller subranges by nodes F, E, C, and B. As a result, case (iii) can be avoided. Although this technique relies on the characteristic of SSA, i.e., a key of a virtual node is a suffix of the key of its physical node, it can also be applied to other cases that do not have such characteristic by exchanging the information of child virtual nodes among connected nodes.

IV. Analytical comparison

Here, we analytically compare SSA to the existing methods described in Section II-A. In this section, we use the following notations.

- N: number of physical nodes
- M: average number of labels for each physical node
- L_{avg}: average length of labels
- L_s: length of search word s

Table II shows a summary of the comparison.

Relative to search time, PIER and DST require $O(L_s \log N)$, because they perform lookups in a DHT multiple times according to the length of a search word. For SSA, the height of the Skip Graph is $O(\log N)$, because membership vectors are assigned for each physical node. In addition, the number of physical nodes at level i between two virtual nodes connected at level $i+1$ is expected to be $O(1)$[3]; therefore, SSA incurs $O(\log N)$ search time. Note that each method can be improved in some manner, e.g., parallel processing and caching; however, such improvements are beyond the scope of this paper to clarify the essential differences.

TABLE II
Comparison of partial match retrieval methods

	Search time	Storage cost
PIER	$O(L_s \log N)$	$O(M L_{avg} + \log N)$
DST	$O(L_s \log N)$	$O(M L_{avg} + \log N)$
SSA	$O(\log N)$	$O(M L_{avg} \log N)$

From a storage cost perspective, PIER requires that each node has n-gram data and the routing table of a DHT. Thus, the storage cost is $O(ML_{avg} + \log N)$. For DST, each node has the routing table of a DHT and the edges of a suffix tree it is responsible for. Here, the number of edges is $O(ML_{avg}N)$; therefore, the storage cost on each DST node is $O(ML_{avg} + \log N)$. In SSA, each physical node has ML_{avg} virtual nodes on average; thus, the storage cost is expected to be $O(ML_{avg} \log N)$. Although this indicates SSA is disadvantageous, existing methods suffer from load imbalance (Section II-B). Since the load on each node is unpredictable, ensuring sufficient storage capacity for the possible worst case may be required. Note that we evaluate load imbalance in Section V-C.

V. Evaluation

We evaluated SSA in simulation experiments. The simulation program was implemented in Java (Java SE Development Kit 8) and run on a machine with a quad-core CPU (Intel Core i7-8650U 1.90 GHz), 16 GB memory, and the Windows 10 operating system.

Here, the following five methods were evaluated.

SSA	: proposed method
SSA (old)	: previous version of SSA [10]
PIER (3-gram)	: PIER using 3-grams
PIER (2-gram)	: PIER using 2-grams
DST	: DST

In this evaluation, the simulator generated $10,000$ nodes for each method. Each node has a single label from the set of hashtags described in Section II-B (without duplication). We considered six search word length patterns (search words length: $3, 4, 5, 6, 7, 8$) and prepared $1,000$ random partial match queries for each pattern. Here, each query had two kind of information, i.e., a search word and a source physical node of partial match retrieval. The former was selected randomly from existing substrings of corresponding length in the hashtags. The latter was also selected randomly from the $10,000$ nodes such that the label of the selected node did not contain the given search word.

[3]Strictly speaking, $O(|\ |)$ where $|\ |$ is the size of the alphabet for membership vectors.

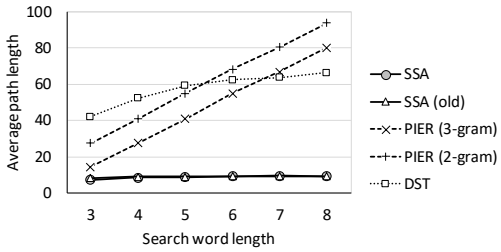


Fig. 7. Average path length in 1; 000 queries

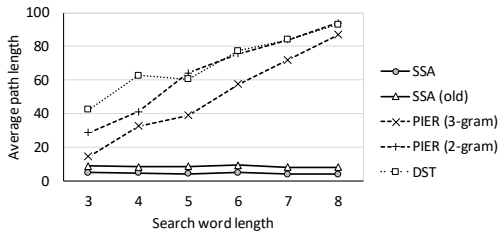


Fig. 8. Average path length in queries with greater than 100 results

For PIER and DST, we used a DHT built on top of Skip Graph. This was realized using "Routing by Numeric ID" in SkipNet [19]. Using a DHT on top of Skip Graph is convenient relative to fairness of the experimental conditions, such as routing table size.

For all experiments, we set the size of the alphabet for membership vectors to 2.

A. Search time

As mentioned in Section II-A, existing methods require multiple exact match queries to process a single partial match query. In contrast, SSA can perform partial match retrieval using a single range query.

To confirm the influence on search time, we performed an experiment to measure the average path length. For each search word length, the simulator performed each of $1,000$ queries and calculated the average path length from the physical source node to the corresponding physical nodes. Then, the average of the $1,000$ queries was calculated.

Fig. 7 shows the results. In PIER, the average path length increased linearly with search word length, and that of DST increased relatively gently. In contrast, SSA and SSA (old) were essentially unaffected by search word length. With a search word length of 8, the proposed SSA reduced the average path length by more than 85% compared to the existing methods.

Due to the improvement of the routing algorithm (Section III-D), the average path length of SSA is shorter than that of SSA (old), especially when using short search word lengths. In principle, since the improvement is for routing within a target range, the difference should be more prominent when the number of results is large (i.e., the target range is wide). In fact, we confirmed this in Fig. 8 which shows the average path length of queries with greater than 100 search results. In these queries, SSA reduced the average path length by more than 44% compared to SSA (old) with every search word length.

B. Communication cost

We also confirmed the tendencies of communication cost. Similar to the search time experiment, the simulator performed each of $1,000$ queries and counted the number of messages exchanged among physical nodes for each search word length. Then, the average of the $1,000$ queries was calculated.

Fig. 9 shows the results. Here, for all methods, search word length of 3 caused a relatively larger number of messages. This is considered to depend on the number of search results. Fig. 10 shows the average number of results for each search word length. As can be seen, shorter search word lengths resulted in a larger average number of results. Especially in PIER (2-gram), a short search word also caused false positive results. Note that SSA, SSA (old), and DST yielded the same number of results because they do not involve false positive results. Fig. 11 shows the difference in the false positive rate between PIER (2-gram) and PIER (3-gram). Using 3-gram rather than 2-gram appears to be effective relative to avoiding the influence of false positive; however, it has a disadvantage of inability to search by words of two or less length.

In Fig. 9, PIER seems to have the tendency of proportional increase to the search word length. DST appears to be insensitive to search word length, despite the increasing tendency of its average path length. This also appears to be caused by the number of search results, i.e., longer search word length simultaneously brings about longer average path length and a smaller average number of search results. Nevertheless, the average number of messages of PIER and DST is quite larger than SSA and SSA (old). With a search word length of 8, the proposed SSA reduced the number of messages by more than 85% compared to the existing methods.

C. Load balance

As described in Section II-B, PIER and DST involve load imbalance. Fig. 12 and Fig. 13 plot the number of messages and storage cost on each node in descending order, respectively. The number of messages was measured using search word length of 8. The storage cost is the sum of the number of routing table's entries and the number of node IDs stored with n-grams (PIER) or edges (DST).

As shown in Fig. 12, SSA and SSA (old) have relatively small deviation compared to PIER and DST.

As shown in Fig. 13, SSA and SSA (old) tend to have greater storage cost than PIER and DST; however, the deviations are small. Note that a good feature of the proposed SSA is storage cost predictability. The storage cost of each node in SSA is dependent on the number of labels (one in this evaluation) which the node has and their length. In contrast, in the existing methods, each node cannot know its storage cost in advance because the data stored on each node are determined by cryptographic hash functions. Fig. 14 shows the label length for each storage cost rank. Here, the results are smoothed with the moving average according to a subset size of 100.