Pampoo: An Efficient Skip-Trie Based Query Processing Framework for P2P Systems*

Li Meifang[1], Zhu Hongkai[2], Shen Derong[1], Nie Tiezheng[1], Kou Yue[1], and Yu Ge[1]

[1] Department of Computer Science and Engineering,
Northeastern University, Shenyang, China, 110004
Li.Meifang@gmail.com, shenderong@ise.neu.edu.cn
[2] Baidu Inc., Beijing, China, 100080
zhuhongkai@baidu.com

Abstract. In this paper, we present Pampoo, a novel distributed framework for efficient query processing in P2P systems. We propose a new locality preserving data structure Skip-trie as its substrate. Skip-trie incorporates the advantages of skip graph with features of traditional trie. Thus, Pampoo can efficiently support various types of queries such as range queries and k nearest neighbor queries. We study the time cost of search and update operations on Skip-trie structure under our Pampoo framework. We further briefly present a repairing strategy to boost the robustness of Pampoo system. Extensive experiments are conducted to verify the effectiveness and efficiency of our approach.

1 Introduction

Distributed peer-to-peer (P2P) computing system has received growing attention for its wide area real-world applications in recent years, which brings forth the notion of sharing resources available at the edges of the Internet. The P2P paradigm specifies a fully distributed, self-organizing network design, where peers collectively form a system randomly. Therefore, it offers enormous potentials for extensive resource sharing, with remarkable features in terms of dynamics, scalability, resilience to failures, self-organizing and load balancing etc.. A large number of systems and architectures that utilize this technology have emerged since its initial success [1, 2, 3, 4].

Therefore, efficient query processing, as a key aspect in P2P systems, is increasingly important at present. Distributed Hash Table (DHT) has been a typical and the most widely applied strategy for peer routing, owing to its inherent characteristics such as scalability, load-balancing and fault-tolerance [1,2,5,6,7]. Nevertheless, DHT can only support exact queries since it adopts the cryptographic hash function such as SHA-1 to map application keys to their identifier space, which impairs the locality properties of the semantically close data items. Thus, DHT is marred by its deficiency in supporting range queries and other complex queries. Currently, several approaches have been proposed to remedy such shortcoming, such as Prefix Hash Tree[8,9], Skip Graph/Net[10,11,12], DP-tree[13] etc..

* Supported by the National Natural Science Foundation of China (60673139, 60473073, 60573090).

M. Xu et al. (Eds.): APPT 2007, LNCS 4847, pp. 190–198, 2007.

1.1 Motivations and Challenges

In our framework, Pampoo (means peer-bamboo in vigorous growth), we aim to design an efficient distributed data structure in support of a fairly rich set of possible data queries such as exact query for a key (e.g. a file name), partial query for a string (e.g. schema matching, prefix matching), k nearest-neighbor query for a numerical attribute, range query over various numerical attributes, multidimensional query, top-k query and point location query in Ad-hoc and sensor networks.

Applications of such queries include DNA databases, fuzzy systems, location-aware services, approximate searches for file names or data titles. Particularly, range query is significant in a large field of applications such as prefetching of web pages, enhanced browsing and efficient searching.

Therefore, in this paper, we mainly focus on a unified architecture in support of range query, i.e. locating resources whose keys lie within a certain specified range, which can also easily deal with the former three types of queries since they are the special cases for range query. For example, a prefix query for ISBN numbers in a book database acm.lib, we can resort it to range query constrained within the acm.lib range scope.
Our design of Pampoo intends to meet the following desired features:

1) Fault tolerance: the framework should adjust to the failure of some nodes, allowing simple repairing mechanism at small cost.
2) Efficient queries processing: the framework should support query processing in terms of the number of rounds of communication and number of messages that must be exchanged in order to complete requested query.
3) Small cost at network changes and data updates: the framework should flexibly tackle issues in node join/leave, data insertion/deletion as well as a necessary repairing.
4) Locality preserving: The structure should meet locality preserving in support of range queries that are based on an ordering of the data. This feature has certain practical advantages over DHT. For example, a search from c.neu.edu to k.neu.edu will not require contacting any node outside neu.edu, which not only reduce the searching scope, but also allow the message to be broadcast within neu.edu.

1.2 Contributions

The contributions of this paper are threefold:

- First, we propose a novel data structure Skip-trie which incorporates advantages of skip graph and the locality preserving feature of trie;
- Second, we present our Pampoo framework and study the time cost of search and update operations on Skip-trie structure;
- Third, we present a repairing strategy in support of the robustness and conduct extensive experiments to verify our approach.

The rest of this paper is organized as follows. We start by presenting our novel Skip-trie data structure in section 2; section 3 presents our Pampoo framework and study the operation cost under it. Extensive experiments are conducted in section 4. Section 5 describes a summary of related work, and finally section 6 draws the conclusion.

2 Skip-Trie Structure

2.1 Backgrounds

Trie, or prefix tree, is a common ordered tree data structure that is used to store an associative array of keys. The position in the tree shows what key a node is associated with. All the descendants of any one node have a common prefix of the string associated with that node, and the root is associated with the empty string. Though trie is commonly keyed by strings, it can also easily be adapted to serve similar functions of ordered lists of any construct, e.g., permutations on a list of digits, permutations on a list of shapes, etc.

Skip graph is a distributed data structure that extends the skip list into a distributed environment by adding redundant connectivity and multiple handles into the data structure [10,11]. On average, there are $O(log\ n)$ levels in skip graph. All keys appear in sorted order in the list at Level *0*. Each Level *i*, for $i > 0$, can now contain multiple linked-lists. Each key maintains a *membership vector*, which is a random string of bits. For each *i* greater than *0*, each node appears randomly in one of the many link lists in level *i* with two constraints. First, if node X is a singleton at level $i - 1$, it doesn't appear in any of the linked list at levels higher than $i - 1$. Second, for every linked list *L* at level *i*, there must be another linked list L' at level $i - 1$ where the elements in *L* are a subset of the elements in L'. Skip graph is highly concurrent and resistant to node failures. More importantly, skip graph does not employ a hashing function which allows it to support range queries, since logically similar keys will become neighbors in the skip graph. However, each key must store pointers to an average of two neighbors for each of the $O(log\ n)$ levels. The result is a cost of $O(log\ n)$ state *per key*. Besides, it is unclear how keys are assigned to machines in the system in skip graph, thus skip graph makes no guarantees about system wide load-balancing nor does it make any guarantees about the geographic locality of neighboring keys.

These two limitations of skip graph incur our interest in designing our own data structure skip-trie to address such problems in our designing of Pampoo framework.

2.2 Skip-Trie: Two-Layered Data Structure

For notational convenience, we assume the data items are (but not confined to) data base tuples of multi-attribute relations R, suppose the number of attributes in R is *n*, $R = \{A_1, A_2, ..., A_n\}$, with each attribute A_i ($1 \le i \le n$) being represented as a string (can be other constructs of ordered lists as well). Our Skip-trie is constructed under two steps. First, we dynamically build a reduced logical trie, aka., a longest prefix tree based on the strings(e.g. the attributes, the name ID of a peer in a physical network), which are mapped to trie. Evidently, there are no more than *n* leaf nodes in trie since some attributes may be the substring of others. We consider all the strings of a tuple *t* in R as a segment, which is uniquely identified by a primary key $key(t)$ that

should be an attribute A_i within R. Second, we hash the primary keys with an order preserving hash function, i.e., $h(key_i) = val_i$, and construct the skip graph based on those keys with values (non-redundant skip graph). A hash function is order preserving i.f.f. it satisfies the following property:

Given two input strings s_1 and s_2, $s_1 \prec s_2 \Rightarrow h(s_1) \prec h(s_2)$, where $\prec$ is the prefix operator.

Note here that originally, all the leaf nodes of the trie are assigned some values, but we only consider those keys with special interest (i.e. the primary keys). It is also worth noting that since no two primary keys can be identical, thus our skip graph layer is non-redundant, thus we coin it as NR-skip graph.

The fundamental idea of our approach is to make use of the skip graph for efficient routing, while trie for the locality preserving. The inherent features of both structures are capable of supporting range queries, which we have already addressed. The substantial number of pointers in merely skip graph approach makes it really hard to implement and maintain. Therefore, in our skip-trie structure, the NR-skip graph is constructed based only on the primary keys of the underlying trie structure. Given k primary keys in R, typically $k << n$, NR-skip graph will only maintain k nodes instead of n nodes, thus it is relatively non-densed and the complexity of our data structure is significantly reduced.

Inspired by the two-layer architecture in [14], we can also think of our skip-trie structure as being composed of two layers, with NR-skip graph as the upper layer and trie as the lower one, as is shown in Fig. 1.

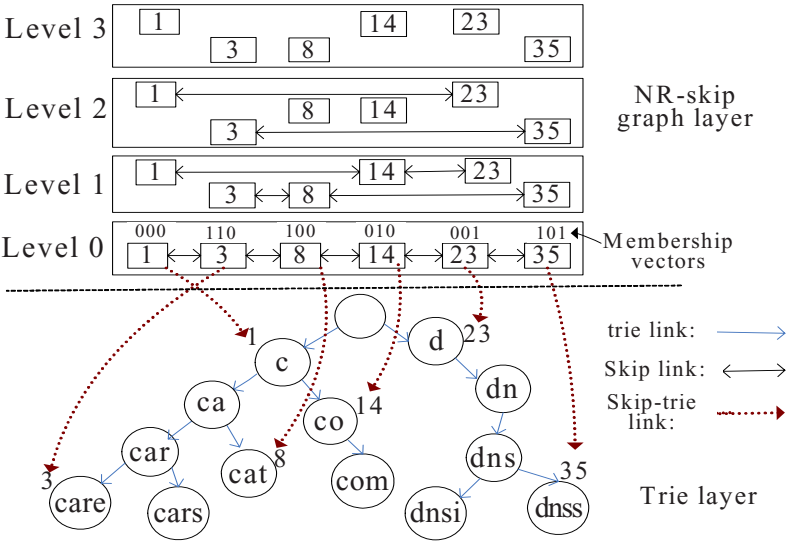

Fig. 1. Two-layered Skip-trie data structure

Observing the properties of skip graphs, we also have following theorem.

Theorem 1. In an NR-skip graph on k nodes, the height of every node is $O(\log k)$ with high probability.

Proof. It is identical to the theorem that with n nodes, the height of every node in skip graph is $O(\log k)$ with high probability.

3 Pampoo: A Skip-Trie Based Framework

In Pampoo, we denote the set of all the peers within the framework as PP , thus each peer $p_i \in PP$ is associated with a path $\vec{p}_i$ in the trie layer of Skip-trie, which corresponds to a binary string. The path may only involve the inner nodes of the trie, which is different from the trie-layered P-Grid architecture that each node only associates with the leaf node. Each peer stores $\vec{p}_i$ the prefixes of its path, thus allowing for efficient search routing.

Now we will discuss how the different operations are addressed in our Skip-trie based framework Pampoo.

3.1 Skip-Trie Search Algorithm

In skip graph, the search operation is achieved in a top-down manner. It is initiated by a top level node -skip layer seeking a key and proceeds down the lower level until it reaches level 0. However, in our skip-trie structure, we approach this operation quite differently. To search for a node with key from node X, we start from the trie layer first and proceed up to the NR-skip graph layer and then down to the trie layer again in a bottom-up-down manner, quite similar in family tree[15]. We incorporate the idea of shower algorithm in [14] that aims to process range queries concurrently, and propose our Skip-trie Search algorithm, which is illustrated as follows.

Algorithm 1. **Skip-trie Search Algorithm**: SSA(p_i, X)

1. Lookup $prefix\_path(p_i)$ // p_i caches the prefixes of its path
2. If $X \subseteq prefix\_path(p_i)$ Then
3. Return X
4. End if
5. $L\_p(X, p_i) \leftarrow$ Longest-prefix-search (X, p_i),
   $X' \leftarrow$ Trie-lookup $(X, prefix\_path(X))$ // find the closest nodes X' // to X with hashed value
6. $L\_p(X, p_i).key \leftarrow$ Map-trie-skip_Level_0 $(L\_p(X, p_i))$,
   $X'.key \leftarrow$ Map-trie-skip_Level_0 (X')
7. $X' \leftarrow$ Skip-level-search $(L\_p(X, p_i).key, X'.key)$
8. Return Trie-lookup (X, X')

In this algorithm, we start from the trie layer with the purpose of making use of the locality property and start the node near the destined node, thus perform the algorithm in an aggressively greedy way.

Lemma 2. The search operation in Skip-trie with n nodes in trie layer and k nodes in NR-skip graph layer takes $O(\log k + r)$ with high probability, where $r =$ maximal_cost{ Trie-lookup$(X, prefix_path(X))$, Trie-lookup(X, X') } .

Proof. In trie with n nodes, the lookup operation takes $O(n)$ amortized time cost, while in NP-skip layer, it takes $O(\log k)$ with high probability, thus verifies lemma 1. Moreover, our Skip-trie Search algorithm is processed in an aggressively greedy way, the cost of lookup operation in trie of n nodes is $O(\log n)$ with high probability (see [16] for details), thus practically the cost is much smaller than in Lemma with high probability.

3.2 Skip-Trie Update

We Address approaches of node join and node leave and their respective time cost in this section.

3.2.1 Node Join. Most of the work required to join (i.e., insert) a node is accomplished by calls to the search operation described in Section 3.1. When a new node X joins, we first find the node sharing the longest prefix with X, and then insert the suffix of X into the trie layer. If the string is a not a primary key, then the operation is over; however, if it is a primary key, we have further to do the insert operations in the NR-skip graph layer identical as described in skip graph.

Lemma 3. The insert operation in Skip-trie with n nodes in trie layer and k nodes in NR-skip graph layer takes $O(\log k + r)$ in expectation and $O(\log^2 k + r)$ with high probability, where r is the same as specified in section 3.1.

Proof. Similar to Lemma 2.

3.2.2 Node Leave. The algorithm for deleting a node X is straightforward. If X only belongs to the trie layer, we simply delete the path. If X also belongs to the NR-skip layer, then it is non-trivial. First, we have to enumerate the nodes with pointers to X and update them to the appropriate predecessor and successor. Then we delete the path in the trie layer.

Lemma 4. The delete operation in Skip-trie with n nodes in trie layer and k nodes in NR-skip graph layer takes $O(\log k + r)$ in expectation and $O(\log^2 k + r)$ with high probability, where r is the same as specified in section 3.1.

Proof. Identical to Lemma 3.

3.3 Repair Strategy

In this section, we describe a self-stabilization strategy in Pampoo that repairs our Skip-tree in case of node failures. If the node only lies in the trie layer, then we do not

have to take any measures since the system is not affected. However, if the node belongs to the NR-skip layer, we have to repair the system for robustness. Thus, the repair strategy mainly focuses on the NR-Skip layer: each node in NR-skip graph layer sends message to its neighbors periodically to see if they are alive. If one of the neighbors fails, then we try to fix the link to the next live neighbor. Our repair strategy works quite similar to that in Skip B-tree[17].

Since load is generally uniform in trie structure, our Skip-trie does not have to handle load balancing problem.

4 Experimental Evaluation

To evaluate the performance of our Skip-trie structure, we implemented Pampoo framework in Java and ran it over Planetlab [7], a testbed for large-scale distributed systems. In our implementation, each peer node is identified both physically by a pair of IP address and port number and logically by its position in the Skip-trie structure.

We compare Skip-trie with PHT with different distribution of data and range queries, since PHT also supports range queries and is easy to implement.

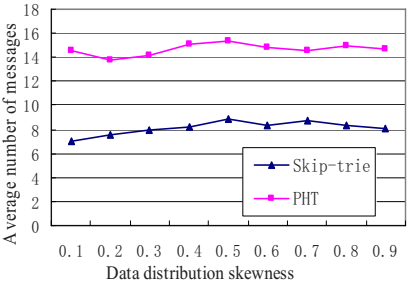

Fig. 2. Comparison of Skip-trie and PHT in number of message with different data distribution

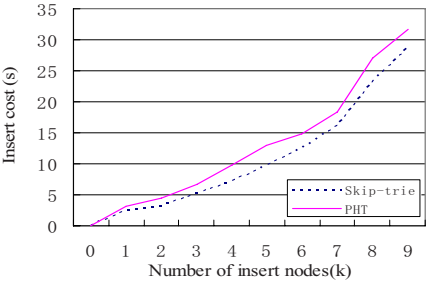

Fig. 3. Comparison of insert cost between Skip-trie and PHT

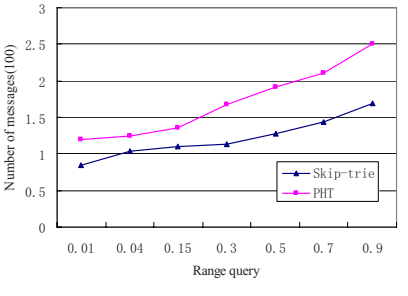

Fig. 4. Comparison of Skip-trie and PHT in number of message with different range query on uniform data distribution

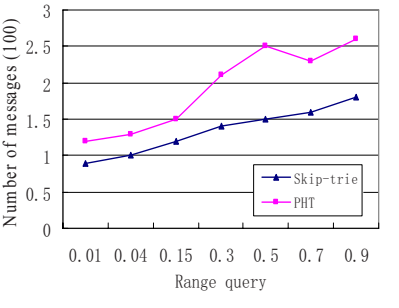

Fig. 5. Comparison of Skip-trie and PHT in number of message with different range query on skewed data distribution

From Fig. 2, we see that the number of messages in Skip-trie is much smaller that in PHT. Fig. 3 indicates that when the number of nodes to be inserted is small, we have fairly small insert cost; however, as the number increase, the time cost grows quickly.

Fig. 4 and Fig. 5 study the number of messages between Skip-trie and PHT under different data distribution. Skip-trie still performs much better than PHT and is not much affected by the skewness distribution.

5 Related Work

There are a wealth of work addressing issues in support of range query in P2P systems. To support approximate range queries, locality preserving hashing to hash ranges instead of keywords is used in [18]. An improvement of this approach to support exact range queries is proposed in [19]. The fundamental problem of these approaches is that the ranges themselves are hashed, and hence, simple key search operations are not supported or are highly inefficient.

Ganesan et. al. propose storage load balance algorithms combined with distributed routing structures which can support range queries [20]. Their solution may sup-port load balance in skewed data distributions, but it does not ensure balance in skewed query distributions. BATON is a balanced binary tree overlay network which can support range queries, and query load balancing by data migration between two, not necessarily adjacent, nodes[11]. In Mercury system, Bharambe et al support multi-attribute range queries and explicit load balancing, using random sampling[5]; nodes are grouped into routing hubs, each of which is responsible for various attributes.In terms of key search efficiency, support for range queries and storage load-balancing, there are some interesting novel structured overlay network abstractions which exhibit performance comparable to our trie-structured proposal: Skip Graphs [10, 11] which are based on skip lists [21].A detailed survey of search mechanisms in P2P systems, including range queries can be found in [22].

6 Conclusion

In this paper we propose a new two-layered data structure called Skip-trie which has several desirable properties. Skip-trie supports range queries in that it exploits the locality preserving feature in location of resources. Based on Skip-trie, we build a distributed P2P framework Pampoo, which aim to support efficient query processing and complex queries. We have studied the time cost of the basic operations in Skip-trie under our Pampoo framework and conducted extensive experiments to verify our approach.

Next, we will study the strategy to support top-k queries and multidimensional queries in our Pampoo framework.

References

1. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM 2001 (2001)
2. Druschel, P., Rowstron, A.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Middleware (2001)

3. Aberer, K., Punceva, M., Hauswirth, M., Schmidt, R.: Improving data access in P2P systems. IEEE Internet Computing 6(1), 58–67 (2002)
4. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-Grid: A Self organizing Structured P2P System. In: ACM SIGMOD Record (2003)
5. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware (2001)
6. Cuenca-Acuna, F.M., et al.: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. Technical Report DCS-TR-487, Rutgers University (September 2002)
7. Ratnasamy, S., et al.: A scalable content-addressable network. In: SIGCOMM 2001 (2001)
8. Ramabhadran, S., Ratnasamy, S., Hellerstein, J., Shenker, S.: Brief Announcement: Prefix Hash Tree. In: Proc. of PODC 2004 (2004)
9. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. ACM SIGCOMM 2001, ACM Press, New York (2001)
10. Aspnes, J., Kirsch, J., Krishnamurthy, A.: Load balancing and locality in range-queriable data structures. In: ACM PODC 2004, ACM Press, New York (2004)
11. Aspnes, J., Shah, G.: Skip graphs. In: ACM-SIAM Symposium on Discrete Algorithms(January 2003)
12. Harvey, N., et al.: SkipNet: A scalable overlay network with practical locality preserving properties. In: Proc.of 4th USENIX Symp. on Internet Technologies and Systems (2003)
13. Mei Li.DP-tree: A Balanced Tree-based Indexing Framework for Peer-to-Peer Systems. In Proc. Of icnp 2006 (2006)
14. Datta, A., et,: al. Range queries in trie-structured overlays. In: Proc. of P2P 2005 (2005)
15. Zatloukal, K.C., Harvey, N.J.A.: Family Trees:An ordered dictionary with optimal congestion, locality, degree, and search time. In: 15th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 301–310. ACM Press, New York (2004)
16. Naor, M., Wieder, U.: Know thy neighbor's neighbor: Better routing in skip-graphs and small worlds. In: 3rd Int. Workshop on Peer-to-Peer Systems (2004)
17. Abraham, I., Aspnes, J., Yuan, J.: Skip B-Trees. In: Proc. of Opodis 2005 (2005)
18. Gupta, A., Agrawal, D., Abbadi, A.E.: Approximate Range Selection Queries in Peer-to-Peer Systems. In: CIDR 2003. 1st Biennial Conference on Innovative Data Systems Research (2003)
19. Sahin, O.D., Gupta, A., Agrawal, D., Abbadi., A.E., Peer-to-peer, A.: Framework for Caching Range Queries. In: 20th ICDE 2004 (2004)
20. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: Proc. of VLDB 2004 (2004)
21. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. Communications of the ACM 33(6) (1990)
22. Risson, J., Moors, T.: Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia (September 2004)