Brief Announcement: ART: Sub-Logarithmic Decentralized Range Query Processing with Probabilistic Guarantees

Spyros Sioutas
Department of Informatics
Ionian University
sioutas@ionio.gr

George Papaloukopoulos
Department of Computer Engineering and Informatics
University of Patras
papalukg@ceid.upatras.gr

Evangelos Sakkopoulos
Department of Computer Engineering and Informatics
University of Patras
sakkopul@ceid.upatras.gr

Kostas Tsichlas
Department of Informatics
Aristotle University of Thessaloniki
tsichlas@csd.auth.gr

Yannis Manolopoulos
Department of Informatics
Aristotle University of Thessaloniki
manolopo@csd.auth.gr

Peter Triantafillou
Department of Computer Engineering and Informatics
University of Patras
peter@ceid.upatras.gr

ABSTRACT

We focus on range query processing on large-scale, typically distributed infrastructures. In this work we present the ART (**A**utonomous **R**ange **T**ree) structure, which outperforms the most popular decentralized structures, including Chord (and some of its successors), BATON (and its successor) and Skip-Graphs. ART supports the join/leave and range query operations in $O(\log \log N)$ and $O(\log_b^2 \log N + |A|)$ expected w.h.p number of hops respectively, where the base b is a double-exponentially power of two, N is the total number of peers and $|A|$ the answer size.

Categories and Subject Descriptors

H.2 [**Database Management**]: Decentralized Query Processing

General Terms

Algorithms, Design, Measurement, Performance, Verification

Keywords

Distributed Infrastructures, P2P Overlays

1. INTRODUCTION AND MOTIVATION

Existing structured P2P systems can be classified into three categories: distributed hash table (DHT) based systems, skip list based systems, and tree based systems (for details see the survey book [1]). The available solutions for architecting such large-scale systems are inadequate for our purposes, since at the envisaged scales (trilions of data items at millions of nodes) the classic logarithmic complexity (for point queries) offered by these solutions is still too expensive. And for range queries, it is even more disappointing. Our aim with this work is to provide a solution that is comprehensive and outperforms related work *with respect to all major operations, such as lookup (insert/delete), join/leave and to the required routing state that must be maintained in order to support these operations.* Specifically, we aim at achieving a sub-logarithmic complexity for all the above! Thus, we present ART, an exponential-tree structure, which remains unchanged w.h.p., and organizes a number of fully-dynamic cluster_peers in efficient way.

PODC'10, July 25–28, 2010, Zurich, Switzerland.
ACM 978-1-60558-888-9/10/07.

2. OUR SOLUTION

First, we build the LRT (**L**evel **R**ange **T**ree) structure, one of the basic components of the final ART structure. LRT will be called upon to organize collections of peers at each level of ART.

Building LRT structure: LRT is built by grouping nodes having the same ancestor and organizing them in a tree structure recursively. The innermost level of nesting (recursion) will be characterized by having a tree in which no more than b nodes share the same direct ancestor, where b is a double-exponentially power of two (e.g. 2,4,16,...). Thus, multiple independent trees are imposed on the collection of nodes. Figure 1 illustrates a simple example, where $b = 2$.

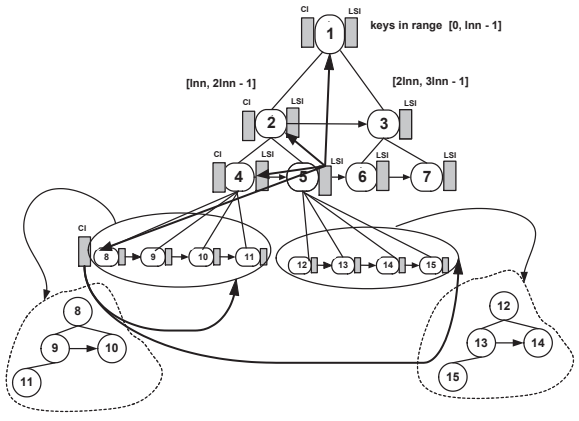

Figure 1: The LRT structure

The degree of the overlay peers at level $i > 0$ is $d(i) = t(i)$, where $t(i)$ indicates the number of peers at level i. It holds that $d(0)$=2 and $t(0)$=1. Let n be w-bit keys. Each peer

with label i (where $1 \leq i \leq N$) stores ordered keys that belong in the range $[(i-1)\ln n, i\ln n–1]$, where $N = n/lnn$ is the number of peers. We also equip each peer with a table named *Left Spine Index* (LSI), which stores pointers to the peers of the left-most spine (see pointers starting from peer 5). Furthermore, each peer of the left-most spine is equipped with a table named *Collection Index* (CI), which stores pointers to the collections of peers presented at the same level (see pointers directed to collections of last level). Peers having the same father belong to the same collection.

Lookup Algorithm: Assume we are located at peer s and seek a key k. First, we find the range where k belongs. If $k \in [(j-1)\ln n, j\ln n-1]$, we have to search for peer j. The first step of our algorithm is to find the LRT level where the desired peer j is located. For this purpose, we exploit a nice arithmetic property of LRT. This property says that for each peer x located at the left-most spine of level i, the following formula holds:

$$label(x) = label(father(x)) + 2^{2^{i-2}} \quad (1)$$

For each level i (where $0 \leq i \leq \log\log N$), we compute the value x of its left most peer by applying Equation (1). Then, we compare the value j with the computed value x. If $j \geq x$, we continue by applying Equation (1), otherwise we stop the loop process with current value i. The latter means that node j is located at the i-th level. Then, we follow the i-th pointer of the LSI table located at peer s. Let x the destination peer, that is the leftmost peer of level i. Now, we must compute the collection in which the peer j belongs to. Since the number of collections at level i equals the number of nodes located at level $(i-1)$, we divide the distance between j and x by the factor $t(i-1)$ and let m the result of this division. Then, we follow the $(m+1)$-th pointer of the CI table. Since the collection indicated by the CI[m+1] pointer is organized in the same way at the next nesting level, we continue this process recursively.

Analysis: Since $t(i) = t(i-1)d(i-1)$, we get $d(i) = t(i) = 2^{2^{i-1}}$ for $i \geq 1$. Thus, the height and the maximum number of possible nestings is $O(\log\log N)$ and $O(\log_b \log N)$ respectively. Thus, each key is stored in $O(\log_b \log N)$ levels at most and the whole searching process requires $O(\log_b \log N)$ hops. Moreover, the maximum size of the CI and RSI tables is $O(\sqrt{N})$ and $O(\log\log N)$ in worst-case respectively.

Building ART structure: The backbone of ART is exactly the same with LRT. During the initialization step we choose as cluster_peer representatives the 1st peer, the $(\ln n)$-th peer, the $(2\ln n)$-th peer and so on. This means that each cluster_peer with label i' (where $1 \leq i' \leq N'$) stores ordered peers with keys belonging in the range $[(i'-1)\ln^2 n, \ldots, i'\ln^2 n - 1]$, where $N' = n/\ln^2 n$ is the number of cluster_peers. ART stores cluster_peers only, each of which is structured as an independent decentralized architecture. Moreover, instead of the **L**eft-most **S**pine **I**ndex (LSI), which reduces the robustness of the whole system, we introduce the **R**andom **S**pine **I**ndex (RSI) routing table, which stores pointers to randomly chosen (and not specific) cluster_peers (see pointers starting from peer 3). In addition, instead of using fat CI tables, we access the appropriate collection of cluster_peers by using a 2-level LRT structure.

Load Balancing: We model the join/leave of peers inside a cluster_peer as the combinatorial game of bins and balls presented in [2]. In this way, for a $\mu(\cdot)$ random sequence of join/leave peer operations, the load of each cluster_peer never exceeds $\Theta(\text{polylog } N')$ size and never becomes zero in expected w.h.p. case.

Routing Overhead: The 2-level LRT is an LRT structure over $\log^{2c} Z$ buckets each of which organizes $\frac{Z}{\log^{2c} Z}$ collections in a LRT manner, where Z is the number of collections at current level and c is a big positive constant. As a consequence, the routing information overhead becomes $O(N^{1/4}/\log^c N)$ in the worst case (even for an extremely large number of peers, let say N=1.000.000.000, the routing data overhead becomes 6 for $c = 1$).

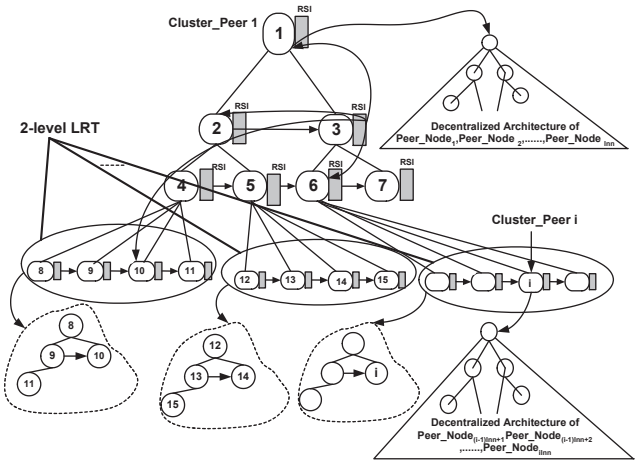

Figure 2: The ART structure

Lookup Algorithms: Since the maximum number of nesting levels is $O(\log_b \log N)$ and at each nesting level i we have to apply the standard LRT structure in $N^{1/2^i}$ collections, the whole searching process requires $O(\log_b^2 \log N)$ hops. Then, we have to locate the target peer by searching the respective decentralized structure. Through the polylogarithmic load of each cluster_peer, the total query complexity $O(\log_b^2 \log N)$ follows. Exploiting now the order of keys on each peer, range queries require $O(\log_b^2 \log N + |A|)$ hops, where $|A|$ the answer size.

Join/Leave Operations: A peer u can make a join/leave request at a particular peer v, which is located at cluster_peer W. Since the size of W is bounded by a $polylogN$ size in expected w.h.p. case, the peer join/leave can be carried out in $O(loglogN)$ hops.

Node Failures and Network Restructuring: Obviously, node failure and network restructuring operations are according to the decentralized architecture we use in each cluster_peer.

Performance Evaluation: Our experimental performance studies include our development of BATON*. The source code of the whole evaluation process, which showcases the improved performance, scalability, and robustness of ART is publicly available at http://code.google.com/p/d-p2p-sim/.

3. REFERENCES

[1] J. F. Buford, H. Yu, and E. K. Lua. *P2P Networking and Applications*. Morgan Kaufman Publications, California, 2008.

[2] A. Kaporis, C. Makris, S. Sioutas, A. Tsakalidis, K. Tsichlas, and C. Zaroliagis. Improved bounds for finger search on a ram. In *Proceedings 11th Annual European Symposium on Algorithms (ESA)*, pages 84–89. 325-336, September 2003.