ACM SIGMOD Programming Contest 2017

Team: gStreamPKU
Shuo Han, Jiaze Chen, Lei Zou(advisor)
{hanshuo, chenjiaze, zoulei}@pku.edu.cn
Institute of Computer Science and Technology, Peking University

1. Task Description

The task is to solve the **N-gram matching** on documents where the dictionary of N-grams is dynamically updated. Firstly the test harness sends the initial set of N-grams. The time spent on loading, pre-processing or indexing the initial dictionary will not be counted into the total execution time.

Then the workload comes in batches. Each batch consists of three types of operations:

(1) *A gram* -- add a new N-gram to the dictionary.
(2) *D gram* -- delete a N-gram from the dictionary, if it exists.
(3) *Q doc* -- find all the N-grams that appear in the document.

Our goal is to answer these queries correctly, and as quickly as possible.

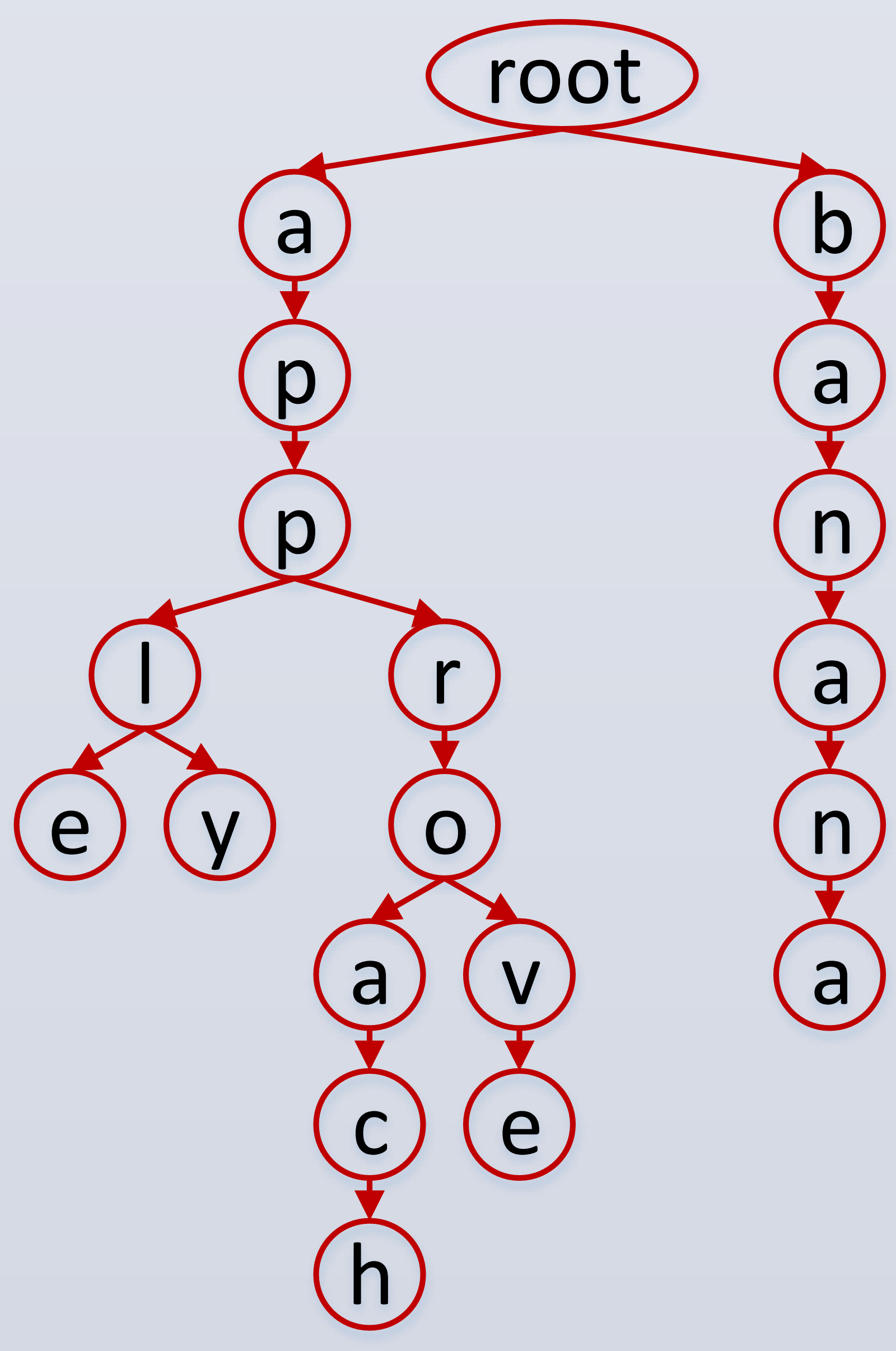
2. Solution Overview

We have tried to improve the performance from the following aspects:

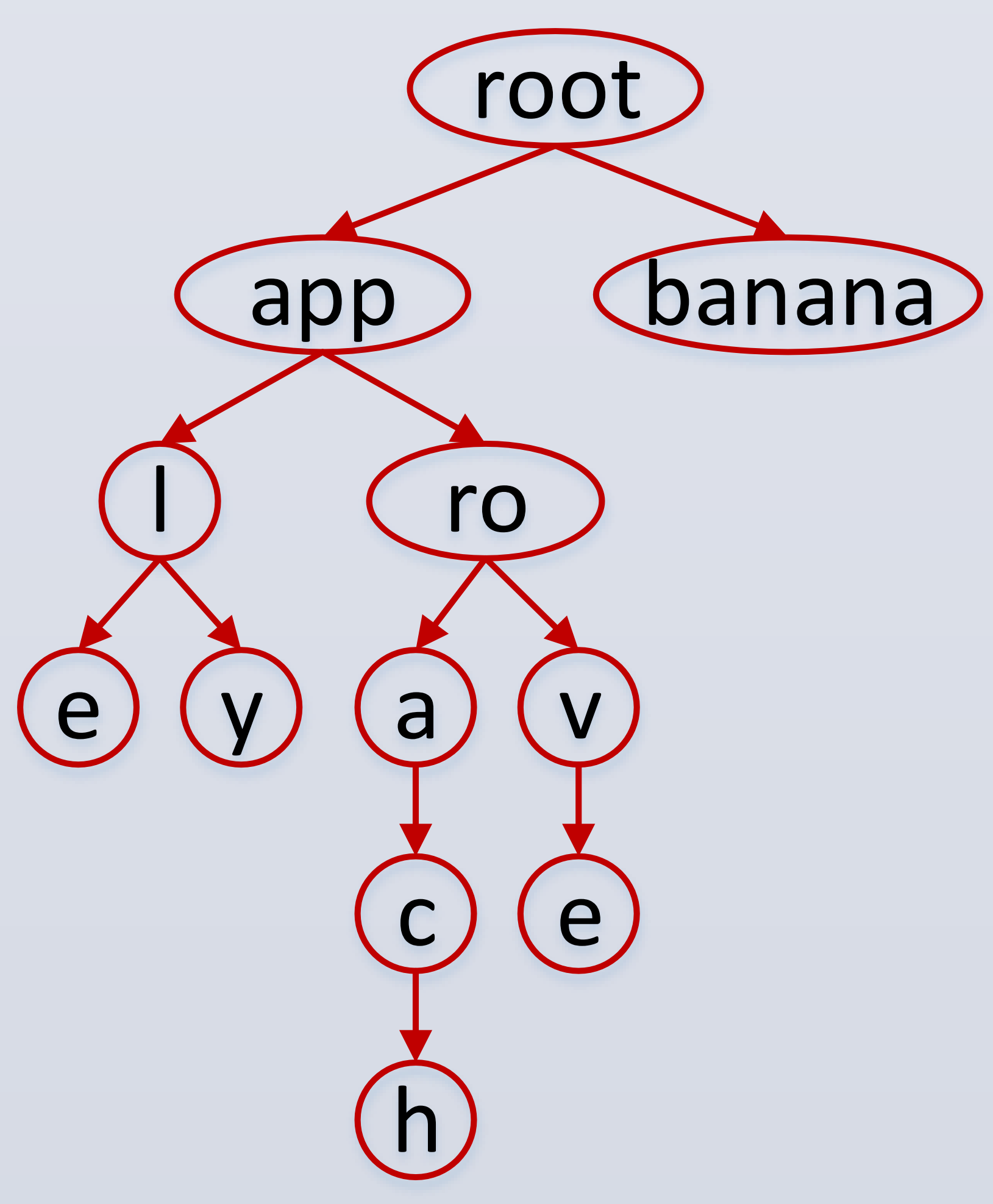
- Maintaining Compact Prefix Tree.
- Insert/Delete parallelism .
- Query parallelism.

3. Compact Prefix Tree

Example: 1. apple 2. apply 3. approach 4. approve 5. banana



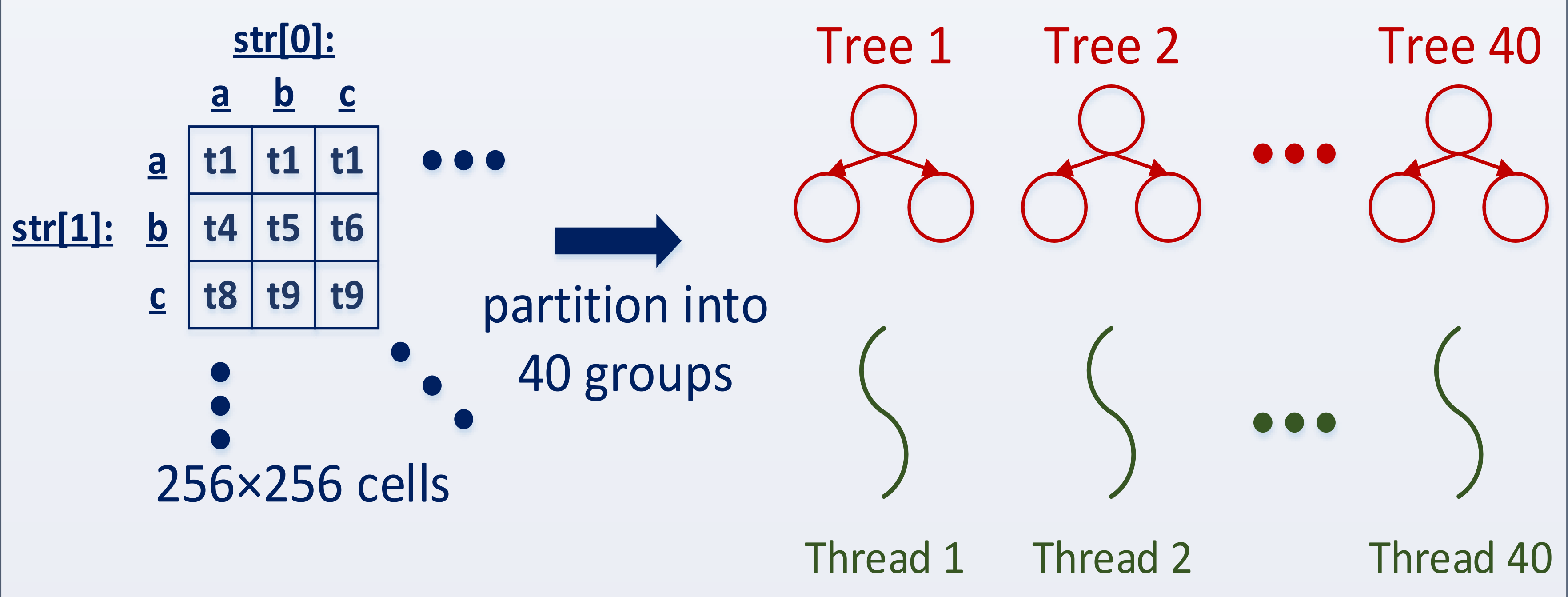
(a) Prefix Tree



(b) Compact Prefix Tree

Basically, we use prefix tree (trie) to store the set of N-grams. Unlike the regular prefix tree (Figure a) that stores each single character by one node, we compact each chain path on the tree into one condensed node (Figure b). The compact prefix tree reduces the tree's depth and the total number of tree nodes, so that improves both of insertion/deletion performance and query performance.

4. Insert/Delete Parallelism

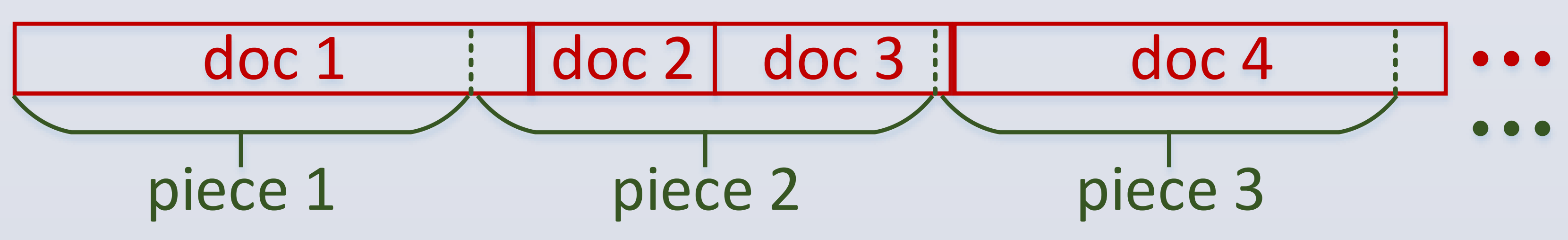


(c) N-grams partition & multithreading

In order to insert/delete these N-grams concurrently, we partition them into 40 non-overlapping groups, and each group corresponds to its own prefix tree. Thus each thread is responsible for one group's insertion or deletion. The N-grams in the same group are processed sequentially by one thread, while the 40 threads execute concurrently without locking.

The partition strategy is as follows: Firstly we divide the N-grams into 256*256 classes by the first two character (str[0] and str[1] in Figure c). Specifically, we consider the space character (' ') as the "second" character of unigrams. Then we count the size of each class over the initial N-grams set. Finally the 256*256 classes are combined into 40 groups according to their size. We devise a greedy algorithm to make the 40 groups' size as balanced as possible.

5. Query Parallelism



(d) splitting query documents into equal-sized pieces

We find that the length of query documents within a batch varies a lot. In order to achieve workload-balancing among different threads, we firstly concatenate all the queries into one long string, then split it into small and equal-sized pieces (in Figure d). These pieces are processed in a thread-pool manner. For each operation (insertion, deletion, and query) within a batch, we assign it a unique timestamp by the timing order. By comparing with the timestamps, we can check whether a match of N-gram is valid for its query document.

6. Other Optimization Issues

We also cache all the first two level tree node pointers, which improves the query performance slightly. Besides, we make some efforts on how to read data and write answers faster.

7. Third Party Libraries

- Tcmalloc in gperftools-2.5, under BSD License.
- Intel Thread Building Blocks 4.4, under Apache v2.0 License.