SIGMOD Programming Contest

Yu Jiang, Jian He, Dong Deng, Jiannan Wang

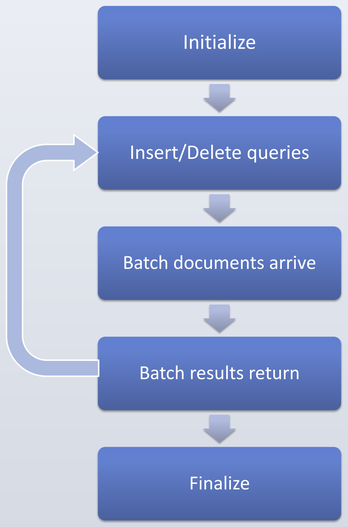
Advisor: Guoliang Li, Jianhua Feng

Department of Computer Science and Technology, Tsinghua University, China

Task Overview

In brief, the task is to filter streaming documents by a dynamic set of exact and approximate queries.

Below is the typical workflow of the system:



Task Detail

3 kinds of queries:

1. Exact matching
2. Approximate matching with an edit distance threshold
3. Approximate matching with a hamming distance threshold

A query should appear in the result of a document if and only if we can find a same(for kind 1) or similar(for kind 2 and 3) word in the document for every word in the query.

Some details:

1. Edit(hamming) distance threshold is no larger than 3.
2. The number of the words in a query is no larger than 5.
3. Only a-z will appear in a word.

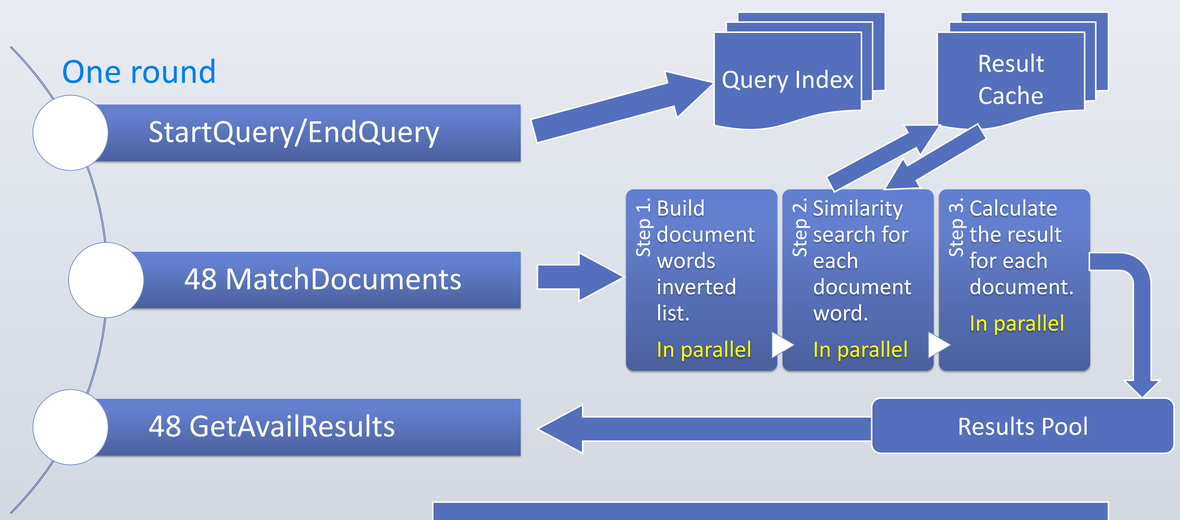
Some statistics:

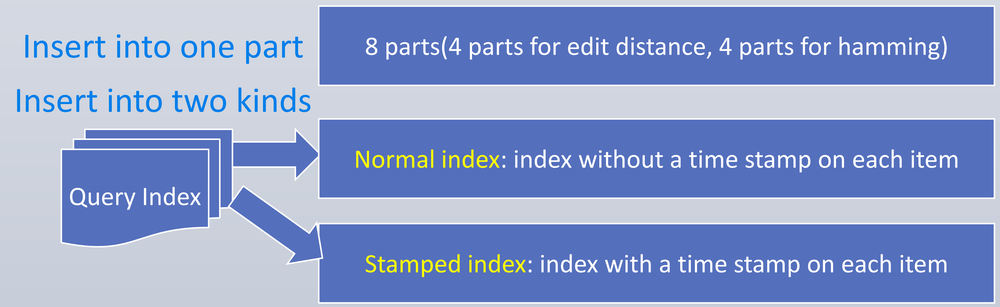
1. The average length of the words is 6-8.
2. Each type of query accounts for one third.
3. The max number of active queries is ~500,000.
4. One document may contain about 300-3000 words.
5. Each round, the program will deal with 48 documents.
6. Queries may share words with each other. One query can provide one unique word on average.
7. Documents in same round may share words with each other. A word will occur in 2-3 documents on average.
8. Documents in different rounds may also share words with each other. The words in a round will probably occur in the next round.

Interfaces

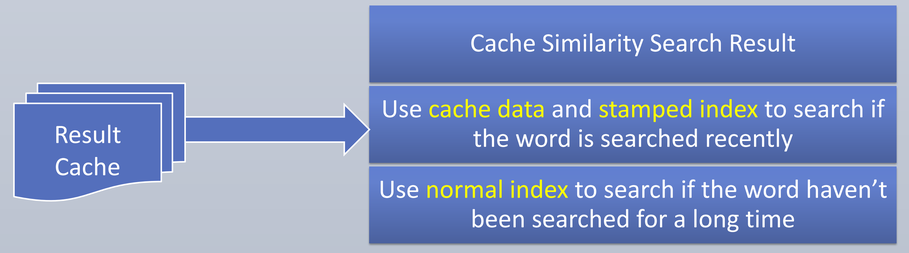
```
void InitializeIndex(); // do the initialization at first
void StartQuery(int query_id, const char *query_str, MatchType match_type, unsigned match_distance); // insert a new query
void EndQuery(int query_id); // remove a query from the system
void MatchDocument(int doc_id, const char *doc_str); // get the filtering result which will be returned later
void GetNextAvailRes(int *p_doc_id, unsigned *p_num_res, int **p_query_ids); // return an available result
void DestroyIndex(); // do the finalization at last
```

Our Approach





similarity search index
we use "PassJoin"[1] algorithm to do similarity search



Why 2 kinds of index?

When a query is deleted, we just mark it as removed rather than do real deletion in the index.
So, the index and the cache will have a lot of redundant information which will slow down our computation.
Stamped index is used to speed up since we only need to search a small part of the index, but it cannot clean redundant data. Normal index will rebuild periodically and when we use it, most redundant information is cleared.

About Parrallel

MatchDocuments step1. We build 26 inverted lists and each list contains only the words start with a specific letter. So we can build these lists in parallel.

MatchDocuments step2. We have many words to search so we can search them in parallel.

MatchDocuments step3. We have 48 documents to deal with so we can do it in parallel.

Evaluation environment: 12 cores, 24 threads(hyper-threading).

Our setting: 12threads, using threading pool.

About PassJoin[1]

An algorithm to do similarity join with edit distance constraints.

Basic idea: If the edit distance between A and B is no larger than T and A is split into T + 1 segments in an arbitrary way, B should contain a substring which is same with one of the segments.

Example:

(with T = 2)

String A: abc | def | ghi

String B: abd dde gh (filtered!)

Basic implementation:

Select every substring of B(huge number!)

Improve implementation(multi-aware selection):

Choose "useful" strings: if some substring is same with a specific segment,

a. they must have same length

b. their position should be close to each other

(Details can be found in the paper)

★ In the EDBT Similarity Search/Join Competition, we have achieved 3 champions of the 4 tasks. Our algorithm is based on PassJoin[1]. It's the state of the art algorithm in such area.

Other Techniques

1. We choose google sparsehash[2] as the hashmap. It achieves much better performance in multithread environment than gcc 4.7 std::unordered_map and std::tr1::unordered_map.
2. We use SIMD[3] technology to speed up edit distance computation, which is used in the verification stage of the PassJoin[1] algorithm. By using SSE instruction set, which supports 128-bit integer and floating point numbers, we can verify 8 strings each time. By using AVX2 instruction set, which supports 256-bit integer operations, we can verify 16 strings each time.

Reference

[1] Guoliang Li, Dong Deng, Jiannan Wang, Jianhua Feng: PASS-JOIN: A Partition-based Method for Similarity Joins. PVLDB 5(3): 253-264(2011)

[2] google sparsehash: https://code.google.com/p/sparsehash/

[3] SIMD introduction: http://en.wikipedia.org/wiki/SIMD