Fifth Annual SIGMOD Programming Contest
Streaming Document Filtering System

Pavel Fedotovsky pavel.v.fedotovsky@gmail.com
Kirill Cherednik kirill.cherednik@math.spbu.ru
George Erokhin george.erokhin@gmail.com

Supervised by: George Chernishev, Kirill Smirnov

Saint-Petersburg University, Russia

Brief Description

General idea

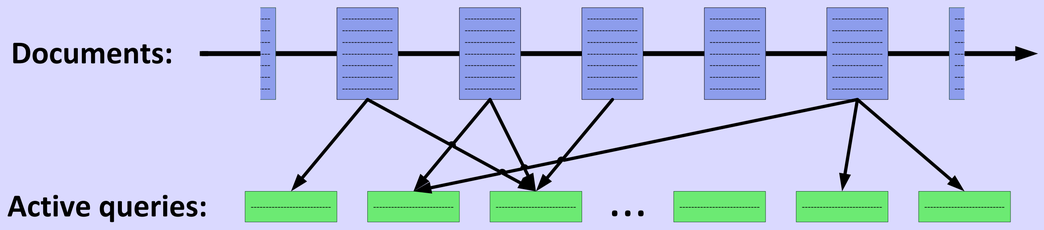
Filter a stream of documents using a dynamic set of exact and approximate continuous keyword match.

Input: Stream of documents and queries
Output: For each document a set of active queries satisfied by this document (for a document to satisfy a query it should contain all the words in the query)

Metrics:
- Discrete
- Hamming distance
- Edit distance

Interface

- **StartQuery()** - Add a query (associated with match type and match distance, where match distance ≤ 3) to the active query set.
- **EndQuery()** - Remove a query from the active query set.
- **MatchDocument() -** Push a document to the server.
- **GetNextAvailRes()** - Return the next available active queries subset that matches any previously submitted document, sorted by query IDs. The returned result must depend on the state of the active queries at the time of calling MatchDocument().

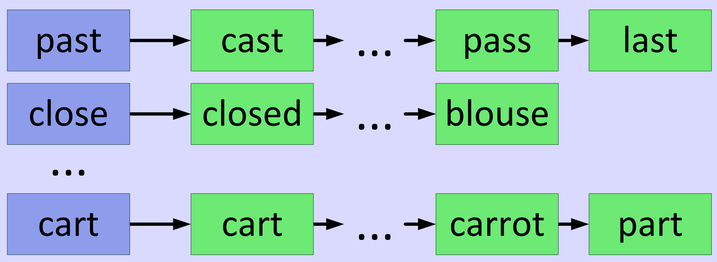


Examples

- If one thinks of tweets as documents, and queries as some hash-tags a user wishes to follow, then it is desirable to know which users should be notified when a new tweet arrives.
- Subscriptions to news services, where a user wishes to be notified each time an article of interest is published.

Main Idea

Keep global cache of all document words and query words that are close to it as a map: ***UDW*** *(Unique document word)* **→** ***List of UQW*** *(Unique query word)*



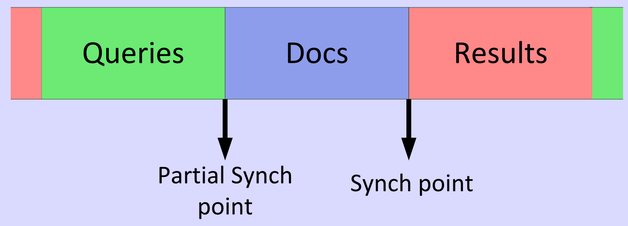
Pros:
- Caching of results in order to minimize number of word matchings
- Efficient document processing
- Efficient usage of multithreaded environment

Cons:
- High memory usage
- Additional expenses to ensure consistency of data structures

All requests to the server are divided into three stages:
- **QUERIES**
- **DOCUMENTS**
- **RESULTS**

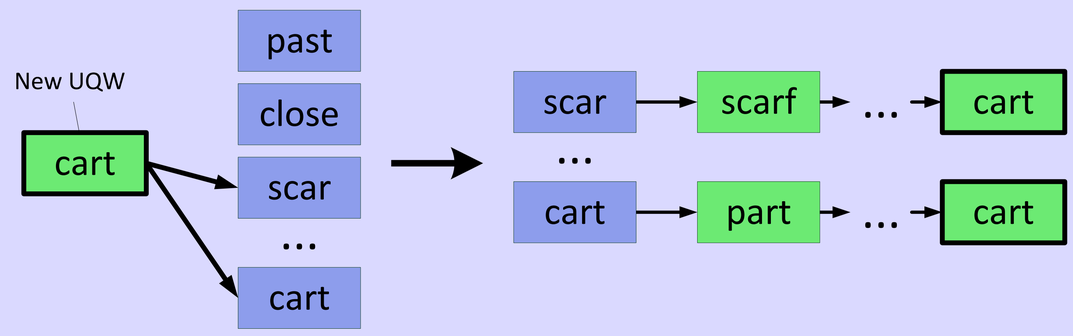
All requests that belong to a single stage are divided into smaller tasks and all these subtasks can be performed in parallel.

This approach allows us to use multithreaded environment more efficiently because synchronization might be needed only when the current stage changes.



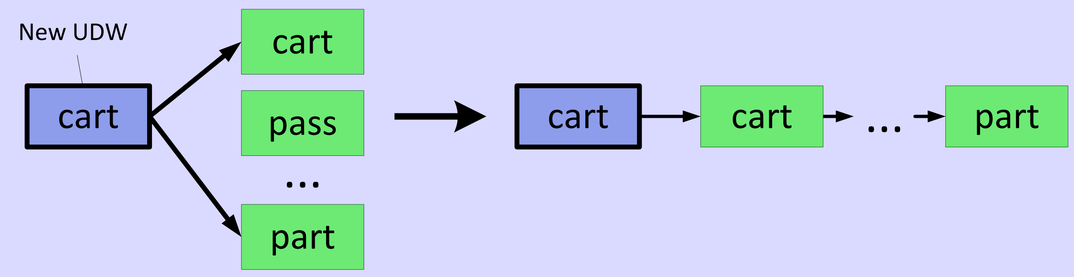
Query Processing

- **Parsing:** Query text → *Set of UQW.*
- **Structures update:** Each new UQW is processed by a single thread. For all UDWs that are close to this UQW, UQW's id is added to their lists.



Document Proccessing

- **Parsing:** Document text → *Set of UDW.*
- **Structures update:** Each new UDW is processed by a single thread.



- **Result construction:** Union of corresponding lists for each UDW is computed and using this union, the result is constructed.

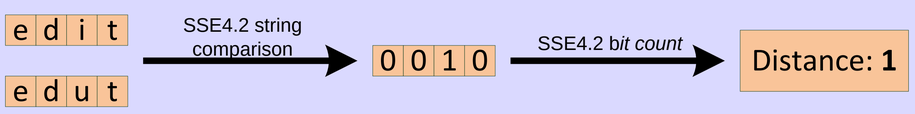
Word Matching

Exact match

Exact match is processed via hashtable check. So every new word is processed in O (1) time.

Hamming distance

In this case we have to check whether two words have hamming distance less than a threshold. The checking is performed using special processor instruction available at **SSE4.2.**



Edit distance

Standard Dynamic Programming approach takes O (N × M) time and computes distance between two given strings.

		e	l	e	p	h	a	n	t
	0	1	2	3	4	5	6	7	8
r	1	1	2	3	4	5	6	7	8
e	2	1	2	2	3	4	5	6	7
l	3	2	1	2	3	4	5	6	7
e	4	3	2	1	2	3	4	5	6
v	5	4	3	2	2	3	4	5	6
a	6	5	4	3	3	3	3	4	5
n	7	6	5	4	4	4	4	3	4
t	8	7	6	5	5	5	5	4	3

But the given task is not to compute edit distance but to check if it is less than a given threshold.
- Not all of the matrix elements are needed
- Computation may be aborted
- Some transitions may be precomputed (automata)