

Write a brief (<500 words) description of your approach, including answers to the following questions:

1) What are the most important goals of your autocomplete design and how are those goals

achieved?

- generate predictions that help people save time by allowing them to quickly complete their searches.

- Avoid people to make complex queries and typos.

- People require suggestions, trending topics that will further help them to use the product better.

To achieve the above goals

we would require a system given in the diagram.

To start we would require an API gateway to handle search queries. This will also take care of validation of queries and caching of repeated queries. Then,

It might have other responsibilities such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling.

Mainly our autocomplete consists of two parts:

* Giving Suggestions

get\_suggestions(prefix) --> We will use AJAX calls.

prefix: Whatever the user has typed until the moment the search query is sent.

* Adding new trending queries to the database

add\_to\_db(query):

query: a new unique trending query which has been searched above a certain threshold will be stored in our DB.

When a user starts writing in the search box, the get\_suggestions() query hits our application servers on every character the user enters. It obtains the top 10 suggestions.

Suggestions service is responsible to obtain the top 10 suggestions from the database.

If the query is popular, the query has been searched above a certain threshold number, then it must be added to the DB. The Trending Queries service is responsible to add the new trending queries to our database

DATA STRUCTURE TO STORE SEARCH QUERIES.

Trie is a data structure that fits the prefix search well. It gives search time complexity of O(l) where l is the length of the prefix which has been typed.

The frequency of each query will also be stored with each node.

we have entered Rod - all possible suggestions would be found by going through the entire subtree under Rod and all nodes which have frequency i.e which are complete words will be fetched.

From Rod we get -- Rod Derline (10), Rod Foster(12), Rod Higgins(15)

Time complexity to go to the prefix node- O(L)

L is the length of the prefix

We need to precompute all the possible top suggestions for each prefix node in the Trie data structure.

We need a hash table which will store prefixes as key and suggestions as value. We can store this hash table in a cache like Redis so that we can provide suggestions

Key-value store: A Trie can be represented in a hash table form by applying the following logic:

• Every prefix in the Trie is mapped to a key in a hash table.

• Data on each Trie node is mapped to a value in a hash table.

Storing the TRIE in database (using Document database, MongoDB, or Cassandra)

With each node, we can store what character it contains and how many children it has. Right after each node, we should put all its children.

One server gets filled by queries from a-abd, then the next server can store from abe-cdx. The next one will store from cdy-eaa and so on. This way we will need to store mapping with it so that it can route a prefix query to the correct shard.

Hash the words and the [(generated hash) % (number of shards)] will give the server number. This way queries will be stored randomly. We can introduce consistent hashing to avoid problems due to the failure of any Trie server.

Trending Queries Service collects popular or recent queries and if the frequency of these new queries is more than a certain threshold, it is stored in our Trie servers.

we have a log of queries, and their weight until a certain timestamp. Weight can be decided based on their frequency, its recency, no. of users making that query.

2) What technologies would your system leverage?

- For the microservices such as API gateway Trending queries & Suggestions we can Django (python)/ spring boot(java)

for

-TRIE data store can be implemented using Document database, MongoDB, or Cassandra.

- For cache - Top suggestions Redis, Memcached.

- NOSQL store itself can be used for storing the snapshots of the suggestions.