**Chapter 13**

Design a Search Autocomplete System

**Design Scope**

*Questions*

* Is the matching supported at the beginning of a search query only?
* How many autocomplete suggestions should the system return?
* How should the system prioritize the suggestions?
* Does the system support spell check?
* Are search queries in English?
* Do we allow capitalization and special characters?
* How many users use the product?

*Requirements*

* Fast response time
  + An article from Facebook’s autocomplete system reveals that the system needs to return results within 100 milliseconds, or it feel like stuttering
* Relevant
  + Suggestions should be relevant to the search term
* Sorted
  + Results returned by the system must be sorted by a priority system
* Scalable
* Highly available

*Back of the envelope estimations*

* Assume 10 million daily users (DAU) and a user performs 10 searches daily
* 20 bytes of data per query string:
  + Assuming ASCII characters – 1 character = 1 byte
  + Assume a query contains 4 words, each word contains 5 chars on average
  + 4 x 5 = 20 bytes per query
* On every character entered in the search box, the client sends a request to the backend for autocomplete suggestions
* Assume on average, 20 requests are sent for each search query
* Query per second = ~24,000 query per second (QPS)
  + 10 million users \* 10 queries / user \* 20 char / 24 hr / 3600 s
* Peak QPS ~ 48,000
* Assume 20% of the daily queries are new
  + 10 million \* 10 queries \* 20 byte per query \* 20% = 0.4 GB
  + 0.4 GB of new data is added to storage daily

**High-level Design**

Two systems will be explored:

* Data gathering service – gathers user queries and aggregates them in real-time
* Query service – given a search query/prefix, return 5 most frequently searched terms

*Data gathering service*

* Simplified concept
  + Assume we have a frequency table that stores the query string and its frequency
  + The table will off start empty but as the user enters queries, the table is updated by the amount of frequency which query is searched

A screenshot of a grid

Description automatically generated

*Query service*

* When a user types into the search box, the following top 5 searched queries with the prefix are displayed
* SQL query example:

A close-up of black text

Description automatically generated

* Acceptable solution when data set is small
* For large data sets, accessing the database is a bottleneck

**Design deep dive**

*Trie data structure*

* Tree-like data structure that can compactly store strings
* Overcomes the performance bottleneck with large datasets in relational databases
* Concept
  + The root represents an empty string
  + Each node stores a character and has 26 children, one for each possible character
  + To save space, we do not draw empty links
  + Each tree node represents a single word or prefix string
* Algorithm definitions
  + P – length of a prefix
  + N – total number of nodes in a trie
  + C – number of children of a given node
* Steps to get top k most searched queries:
  1. Find the prefix
     + Time complexity
  2. Transverse the subtree from the prefix node to get all valid children. A child is valid if it can form a valid query string
     + Time complexity
  3. Sort the children and get top k
     + Time complexity

A diagram of a company

Description automatically generated

* Total time complexity of this algorithm is:
* Drawback
  + Case scenario is slow – need to traverse the entire trie to get the results
* Optimizations
  + Limit the max length of a prefix
  + Cache top search queries at each node

*Limit the max length of a prefix*

* Users rarely type a long search query into the search box
* Safe to assume p is a small integer number – say 50
* Reduces to

*Cache top search queries at each node*

* To avoid traversing the whole trie, we store top k most frequently used queries at each node
* Significantly reduce time complexity
* Requires a lot of space to store top queries at every node
  + Trading space for time is well worth as fast response time is crucial

A diagram of a company

Description automatically generated

*With both optimizations*

* Find the prefix node – O(1)
* Return top k – O(1)
* Our algorithm becomes O(1) overall to fetch top k queries

*Data gathering service*

* Saving search queries in real-time is not practical
  + Users may enter billions of queries per day – updating per query will slow down server and cause server overloading
  + Top suggestions may not change much once the trie is built – unnecessary to update the trie frequently
* Data used to build the trie are usually from analytics and logging services

A diagram of a data processing process

Description automatically generated

*Analytics logs*

* Stores raw non-indexed data about search queries

*Aggregators*

* Aggregates data to be easily processed by the system
* data is aggregated differently depending on the use-case
* Aggregating data once a week is usually good enough
* If real-time results are important, aggregating data will need to be more frequent
  + E.g. Twitter aggregated data in shorter time intervals as real-time results are important

*Aggregated Data*

* Data outputted from aggregators
* Usually a table with the query search, time frame, and frequency

*Workers*

* Set of servers that helps build and update the trie data structure

*Trie Cache*

* A distributed cache system that keeps trie in memory for fast read
* Takes weekly snapshots of the database

*Trie Database*

* Persistent storage
* 2 options to store data
  + *Document store* – we periodically take a snapshot of the trie, serialize it and store the serialized data in the database like
    - MongoDB is a good fit for serialized data
  + *Key-value store* – can be represented in a hash table form
    - 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

A diagram of a data flow

Description automatically generated

*Query service Optimizations*

*AJAX request*

* Used to fetch autocomplete results
* Benefit of AJAX is that requests/responses do not refresh the page

*Browser caching*

* For many apps, autocomplete search suggestions will not change much within a short time
* Autocomplete suggestions can be saved in browser cache to allow subsequent requests to get results from the cache directly
  + E.g. Google search engine uses the same mechanism

*Data sampling*

* For a large-scale system, logging every search query requires a lot of processing power
* Only one out of every N number of requests is logged by the system

*Trie Operations*

*Create*

* Created by workers using aggregated data
* Source of data is from Analytics Log/Db

*Update*

* 2 methods
  1. Update the trie weekly
     + A new trie replaces the old one
  2. Update individual trie node directly
     + Usually slower
     + Acceptable solution if the trie is slow
     + When updating a trie node, its ancestors all the way up the root must also be updated

A diagram of a beer

Description automatically generated

*Delete*

* Remove hateful, violent, sexually explicit, or dangerous autocomplete suggestions
* Add a filter layer in front of the trie cache
  + Offers the flexibility of removing results based on different filter rules
  + Unwanted suggestions are physically removed from the database asynchronously to ensure accurate data in the next update cycle

*Scale the storage*

* Sharding database based on the first letter
  + E.g. 2 servers can divide by words starting from a-m and n-z
  + E.g. 26 servers can be divided into 26 alphabets
  + E.g. increasing from 26 can shard on the second or third letter
  + Uneven distributed – certain letters have a lot more word possibilities
    - E.g. comparing how many words starts with c than x
* Smarter sharding logic
  + Shard map manager – maintains a lookup database for identifying where rows should be stored

*Other considerations*

* Multi-language support
  + Store Unicode in trie nodes
* Localizing top search queries for different countries
  + Build different tries for different countries
* Support for real-time search query
  + Reduce the working data set by sharding
  + Change the ranking model and assign more weight to recent search queries
  + Data may come as streams, so we do not have access to all the data at once
    - Streaming data – data is generated continuously
    - Stream processing requires a different set of systems
      * Apache Hadoop MapReduce
      * Apache Spark Streaming
      * Apache Storm
      * Apache Kafka