

PK

educative(/learn)

Explore(/explore)

Tracks(/tracks)

My Courses(/mycourses)

Edpresso(/edpresso)

Refer a Friend(/refer-a-friend)

Create Courses(/courses/grokking-the-system-design-interview/mE2XkgGF)

Grokking the System Design Interview

(/collection/5668639101419520/56490502)

87% completed

Q

Search Course

●

Designing a URL Shortening service like TinyURL(/courses/grokking-the-system-design-interview/m2ygV4E81AR)

●

Designing Pastebin(/courses/grokking-the-system-design-interview/3jyvQ3pg6KO)

●

Designing Instagram(/courses/grokking-the-system-design-interview/m2yDVZnQ8IG)

●

Designing Dropbox(/courses/grokking-the-system-design-interview/m22Gymjp4mG)

●

Designing Facebook Messenger(/courses/grokking-the-system-design-interview/B8R22v0wqJo)

●

Designing Twitter(/courses/grokking-the-system-design-interview/m2G48X18NDO)

●

Designing Youtube or Netflix(/courses/grokking-the-system-design-interview/xV26VjZ7yMI)

●

Designing Typeahead Suggestion(/courses/grokking-the-system-design-interview/mE2XkgGRnmp)

●

Designing an API Rate Limiter(/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)

○

Designing Twitter Search(/courses/grokking-the-system-design-interview/xV9mMjj74gE)

○

Designing a Web Crawler(/courses/grokking-the-system-design-interview/NE5LpPrWrKv)

●

Designing Facebook's Newsfeed(/courses/grokking-the-system-design-interview/gxpWJ3ZKYwl)

●

Designing Yelp or Nearby Friends(/courses/grokking-the-system-design-interview/B8rpM8E16LQ)

●

Designing Uber backend(/courses/grokking-the-system-design-interview/YQVkj548NM)

●

Design Ticketmaster (\*New\*)(/courses/grokking-the-system-design-interview/YQyq6mBKq4n)

●

Additional Resources(/courses/grokking-the-system-design-interview/)

Designing Typeahead Suggestion

Let's design a real-time suggestion service, which will recommend terms to users as they enter text for searching.

Similar Services: Auto-suggestions, Typeahead search

Difficulty: Medium

We'll cover the following

1. What is Typeahead Suggestion?

2. Requirements and Goals of the System

3. Basic System Design and Algorithm

4. Permanent Storage of the Trie

5. Scale Estimation

6. Data Partition

7. Cache

8. Replication and Load Balancer

9. Fault Tolerance

10. Typeahead Client

11. Personalization

1. What is Typeahead Suggestion?

#

Typeahead suggestions enable users to search for known and frequently searched terms. As the user types into the search box, it tries to predict the query based on the characters the user has entered and gives a list of suggestions to complete the query. Typeahead suggestions help the user to articulate their search queries better. It's not about speeding up the search process but rather about guiding the users and lending them a helping hand in constructing their search query.

2. Requirements and Goals of the System

#

Functional Requirements:

As the user types in their query, our service should suggest top 10 terms starting with whatever the user has typed.

Non-function Requirements:

The suggestions should appear in real-time. The user should be able to see the suggestions within 200ms.



Explore  
(/explore)



Tracks  
(/tracks)



My Courses  
(/mycourses)



Edpresso  
(/edpresso)



Refer a  
Friend  
(/refer-a-  
friend)

## Grokking the System Design Interview

(/collection/5668639101419520/56490502)

87% completed



🔍 Search Course

- Designing a URL Shortening service like TinyURL  
(/courses/grokking-the-system-design-interview/m2ygV4E81AR)
- Designing Pastebin  
(/courses/grokking-the-system-design-interview/3jyvQ3pg6KO)
- Designing Instagram  
(/courses/grokking-the-system-design-interview/m2yDVZnQ8IG)
- Designing Dropbox  
(/courses/grokking-the-system-design-interview/m22Gymjp4mG)
- Designing Facebook Messenger  
(/courses/grokking-the-system-design-interview/B8R22v0wqJo)
- Designing Twitter  
(/courses/grokking-the-system-design-interview/m2G48X18NDO)
- Designing Youtube or Netflix  
(/courses/grokking-the-system-design-interview/xV26VjZ7yMI)
- Designing Typeahead Suggestion  
(/courses/grokking-the-system-design-interview/mE2XkgGRnmp)
- Designing an API Rate Limiter  
(/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)
- Designing Twitter Search  
(/courses/grokking-the-system-design-interview/xV9mMjj74gE)
- Designing a Web Crawler  
(/courses/grokking-the-system-design-interview/NE5LpPrWrKv)
- Designing Facebook's Newsfeed  
(/courses/grokking-the-system-design-interview/gxpWJ3ZKYwl)
- Designing Yelp or Nearby Friends  
(/courses/grokking-the-system-design-interview/B8rpM8E16LQ)
- Designing Uber backend  
(/courses/grokking-the-system-design-interview/YQVkj548NM)
- Design Ticketmaster (\*New\*)  
(/courses/grokking-the-system-design-interview/YQyq6mBKq4n)
- Additional Resources  
(/courses/grokking-the-system-design-interview/)

+

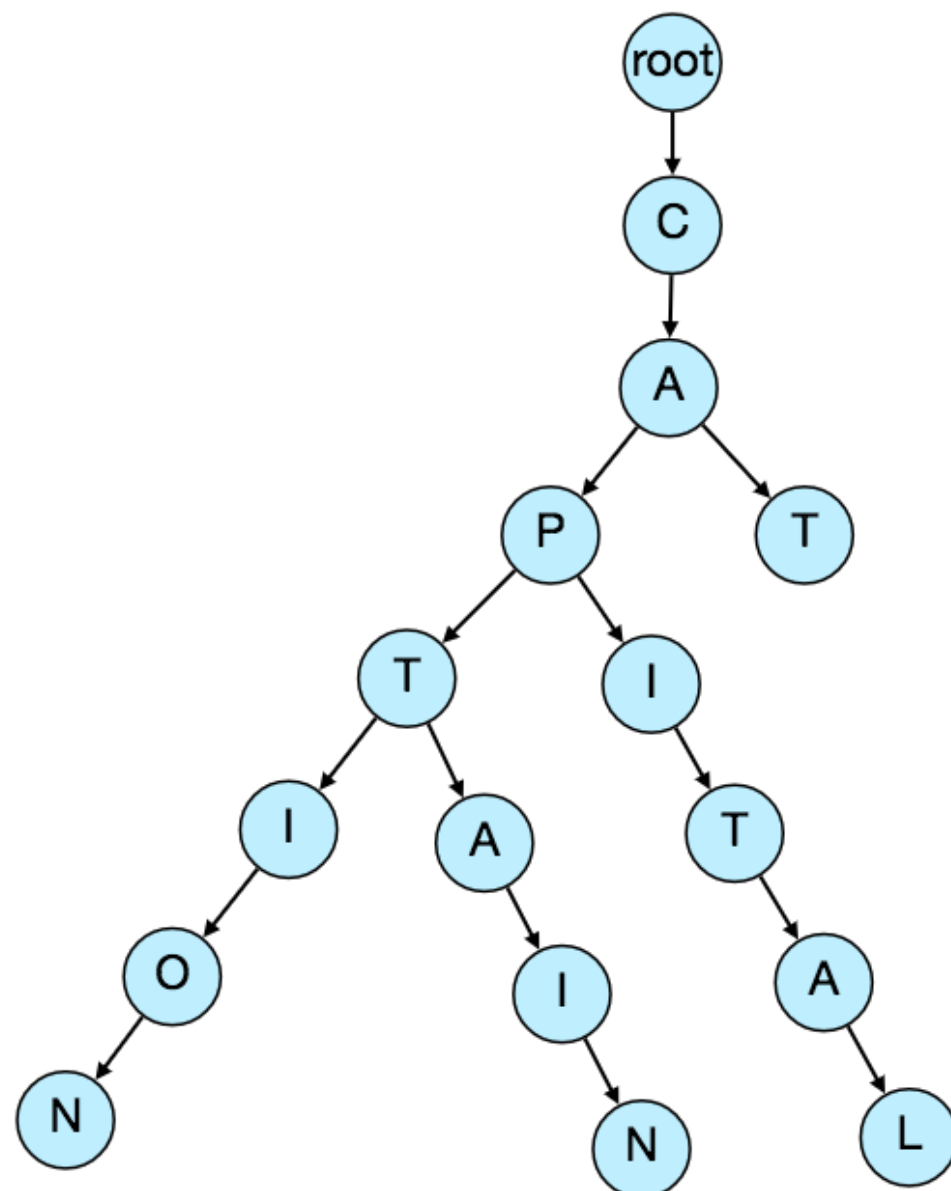
Create  
Courses/grokking-  
system-design-  
interview/mE2XkgGF

### #

The problem we are solving is that we have a lot of 'strings' that we need to store in such a way that users can search with any prefix. Our service will suggest next terms that will match the given prefix. For example, if our database contains the following terms: cap, cat, captain, or capital and the user has typed in 'cap', our system should suggest 'cap', 'captain' and 'capital'.

Since we've got to serve a lot of queries with minimum latency, we need to come up with a scheme that can efficiently store our data such that it can be queried quickly. We can't depend upon some database for this; we need to store our index in memory in a highly efficient data structure.

One of the most appropriate data structures that can serve our purpose is the Trie (pronounced "try"). A trie is a tree-like data structure used to store phrases where each node stores a character of the phrase in a sequential manner. For example, if we need to store 'cap, cat, caption, captain, capital' in the trie, it would look like:




Now if the user has typed 'cap', our service can traverse the trie to go to the node 'P' to find all the terms that start with this prefix (e.g., cap-tion, cap-ital etc).


We can merge nodes that have only one branch to save storage space. The above trie can be stored like this:


 Explore  
(/explore)

 Tracks  
(/tracks)

 My Courses  
(/mycourses)

 Edpresso  
(/edpresso)

 Refer a Friend  
(/refer-a-friend)

 Create  
Courses/grokking-  
system-design-  
interview/mE2XkgGF

















## Grokking the System Design Interview

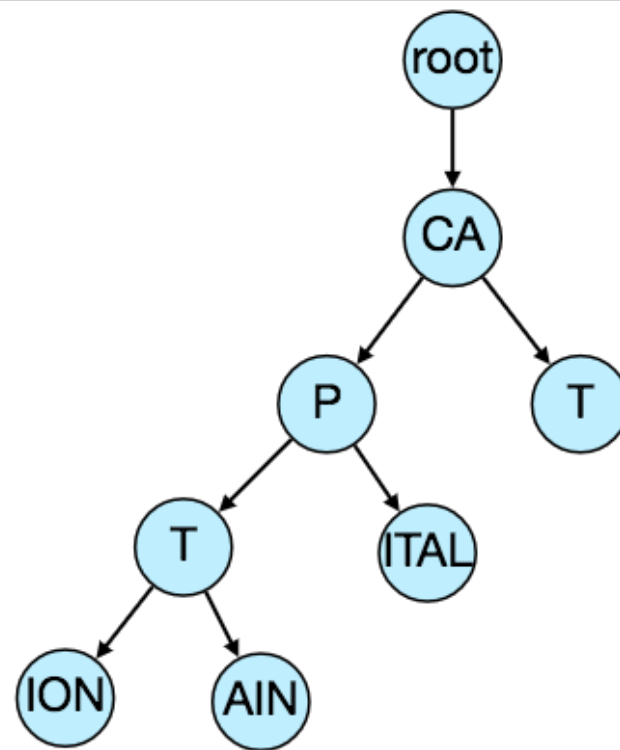
(/collection/5668639101419520/56490502)

87% completed



 Search Course

-  Designing a URL Shortening service like TinyURL  
(/courses/grokking-the-system-design-interview/m2ygV4E81AR)
-  Designing Pastebin  
(/courses/grokking-the-system-design-interview/3jyvQ3pg6KO)
-  Designing Instagram  
(/courses/grokking-the-system-design-interview/m2yDVZnQ8IG)
-  Designing Dropbox  
(/courses/grokking-the-system-design-interview/m22Gymjp4mG)
-  Designing Facebook Messenger  
(/courses/grokking-the-system-design-interview/B8R22v0wqJo)
-  Designing Twitter  
(/courses/grokking-the-system-design-interview/m2G48X18NDO)
-  Designing Youtube or Netflix  
(/courses/grokking-the-system-design-interview/xV26VjZ7yMI)
-  Designing Typeahead Suggestion  
(/courses/grokking-the-system-design-interview/mE2XkgGRnmp)
-  Designing an API Rate Limiter  
(/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)
-  Designing Twitter Search  
(/courses/grokking-the-system-design-interview/xV9mMjj74gE)
-  Designing a Web Crawler  
(/courses/grokking-the-system-design-interview/NE5LpPrWrKv)
-  Designing Facebook's Newsfeed  
(/courses/grokking-the-system-design-interview/gxpWJ3ZKYwl)
-  Designing Yelp or Nearby Friends  
(/courses/grokking-the-system-design-interview/B8rpM8E16LQ)
-  Designing Uber backend  
(/courses/grokking-the-system-design-interview/YQVkj548NM)
-  Design Ticketmaster (\*New\*)  
(/courses/grokking-the-system-design-interview/YQyq6mBKq4n)
-  Additional Resources  
(/courses/grokking-the-system-design-interview/...)



**Should we have case insensitive trie?** For simplicity and search use-case, let's assume our data is case insensitive.

**How to find top suggestion?** Now that we can find all the terms for a given prefix, how can we find the top 10 terms for the given prefix? One simple solution could be to store the count of searches that terminated at each node, e.g., if users have searched about 'CAPTAIN' 100 times and 'CAPTION' 500 times, we can store this number with the last character of the phrase. Now if the user types 'CAP' we know the top most searched word under the prefix 'CAP' is 'CAPTION'. So, to find the top suggestions for a given prefix, we can traverse the sub-tree under it.

**Given a prefix, how much time will it take to traverse its sub-tree?** Given the amount of data we need to index, we should expect a huge tree. Even traversing a sub-tree would take really long, e.g., the phrase 'system design interview questions' is 30 levels deep. Since we have very strict latency requirements we do need to improve the efficiency of our solution.

**Can we store top suggestions with each node?** This can surely speed up our searches but will require a lot of extra storage. We can store top 10 suggestions at each node that we can return to the user. We have to bear the big increase in our storage capacity to achieve the required efficiency.

We can optimize our storage by storing only references of the terminal nodes rather than storing the entire phrase. To find the suggested terms we need to traverse back using the parent reference from the terminal node. We will also need to store the frequency with each reference to keep track of top suggestions.

**How would we build this trie?** We can efficiently build our trie bottom up. Each parent node will recursively call all the child nodes to calculate their top suggestions and their counts. Parent nodes will combine top suggestions from all of their children to determine their top suggestions.







Explore  
(/explore)




Tracks  
(/tracks)



My Courses  
(/mycourses)



Edpresso  
(/edpresso)



Refer a  
Friend  
(/refer-a-  
friend)



Create  
Courses/grokking-  
system-design-  
interview/mE2XkgGF

## Grokking the System Design Interview

(/collection/5668639101419520/56490502)

87% completed



 Search Course

- Designing a URL Shortening service like TinyURL (/courses/grokking-the-system-design-interview/m2ygV4E81AR)
- Designing Pastebin (/courses/grokking-the-system-design-interview/3jyvQ3pg6KO)
- Designing Instagram (/courses/grokking-the-system-design-interview/m2yDVZnQ8IG)
- Designing Dropbox (/courses/grokking-the-system-design-interview/m22Gymjp4mG)
- Designing Facebook Messenger (/courses/grokking-the-system-design-interview/B8R22v0wqJo)
- Designing Twitter (/courses/grokking-the-system-design-interview/m2G48X18NDO)
- Designing Youtube or Netflix (/courses/grokking-the-system-design-interview/xV26VjZ7yMI)
- Designing Typeahead Suggestion (/courses/grokking-the-system-design-interview/mE2XkgGRnmp)
- Designing an API Rate Limiter (/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)
- Designing Twitter Search (/courses/grokking-the-system-design-interview/xV9mMjj74gE)
- Designing a Web Crawler (/courses/grokking-the-system-design-interview/NE5LpPrWrKv)
- Designing Facebook's Newsfeed (/courses/grokking-the-system-design-interview/gxpWJ3ZKYwl)
- Designing Yelp or Nearby Friends (/courses/grokking-the-system-design-interview/B8rpM8E16LQ)
- Designing Uber backend (/courses/grokking-the-system-design-interview/YQVkj548NM)
- Design Ticketmaster (\*New\*) (/courses/grokking-the-system-design-interview/YQyq6mBKq4n)
- Additional Resources (/courses/grokking-the-system-design-interview/...

**How to update the trie?** Assuming five billion searches every day, which would give us approximately 60K queries per second. If we try to update our trie for every query it'll be extremely resource intensive and this can hamper our read requests, too. One solution to handle this could be to update our trie offline after a certain interval.

As the new queries come in we can log them and also track their frequencies. Either we can log every query or do sampling and log every 1000th query. For example, if we don't want to show a term which is searched for less than 1000 times, it's safe to log every 1000th searched term.

We can have a Map-Reduce (MR) (<https://en.wikipedia.org/wiki/MapReduce>) set-up to process all the logging data periodically say every hour. These MR jobs will calculate frequencies of all searched terms in the past hour. We can then update our trie with this new data. We can take the current snapshot of the trie and update it with all the new terms and their frequencies. We should do this offline as we don't want our read queries to be blocked by update trie requests. We can have two options:

1. We can make a copy of the trie on each server to update it offline. Once done we can switch to start using it and discard the old one.
2. Another option is we can have a master-slave configuration for each trie server. We can update slave while the master is serving traffic. Once the update is complete, we can make the slave our new master. We can later update our old master, which can then start serving traffic, too.

**How can we update the frequencies of typeahead suggestions?** Since we are storing frequencies of our typeahead suggestions with each node, we need to update them too! We can update only differences in frequencies rather than recounting all search terms from scratch. If we're keeping count of all the terms searched in last 10 days, we'll need to subtract the counts from the time period no longer included and add the counts for the new time period being included. We can add and subtract frequencies based on Exponential Moving Average (EMA) ([https://en.wikipedia.org/wiki/Moving\\_average#Exponential\\_moving\\_average](https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average)) of each term. In EMA, we give more weight to the latest data. It's also known as the exponentially weighted moving average.

After inserting a new term in the trie, we'll go to the terminal node of the phrase and increase its frequency. Since we're storing the top 10 queries in each node, it is possible that this particular search term jumped into the top 10 queries of a few other nodes. So, we

# Grokking the System Design Interview

(/collection/5668639101419520/56490502)

87% completed

Search Course

- Designing a URL Shortening service like TinyURL (/courses/grokking-the-system-design-interview/m2ygV4E81AR)
- Designing Pastebin (/courses/grokking-the-system-design-interview/3jyvQ3pg6KO)
- Designing Instagram (/courses/grokking-the-system-design-interview/m2yDVZnQ8IG)
- Designing Dropbox (/courses/grokking-the-system-design-interview/m22Gymjp4mG)
- Designing Facebook Messenger (/courses/grokking-the-system-design-interview/B8R22v0wqJo)
- Designing Twitter (/courses/grokking-the-system-design-interview/m2G48X18NDO)
- Designing Youtube or Netflix (/courses/grokking-the-system-design-interview/xV26VjZ7yMI)
- Designing Typeahead Suggestion (/courses/grokking-the-system-design-interview/mE2XkgGRnmp)
- Designing an API Rate Limiter (/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)
- Designing Twitter Search (/courses/grokking-the-system-design-interview/xV9mMjj74gE)
- Designing a Web Crawler (/courses/grokking-the-system-design-interview/NE5LpPrWrKv)
- Designing Facebook's Newsfeed (/courses/grokking-the-system-design-interview/gxpWJ3ZKYwl)
- Designing Yelp or Nearby Friends (/courses/grokking-the-system-design-interview/B8rpM8E16LQ)
- Designing Uber backend (/courses/grokking-the-system-design-interview/YQVkj548NM)
- Design Ticketmaster (\*New\*) (/courses/grokking-the-system-design-interview/YQyq6mBKq4n)
- Additional Resources (/courses/grokking-the-system-design-interview/NE5LpPrWrKv)

need to update the top 10 queries of those nodes then. We have to traverse back from the node to all the way up to the root. For every

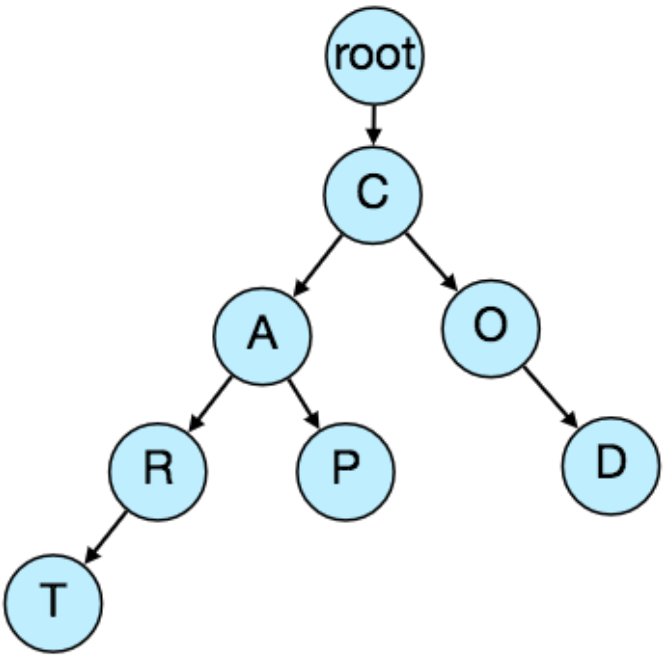
parent, we check if the current query is part of the top 10. If so, we update the corresponding frequency. If not, we check if the current query's frequency is high enough to be a part of the top 10. If so, we insert this new term and remove the term with the lowest frequency.

**How can we remove a term from the trie?** Let's say we have to remove a term from the trie because of some legal issue or hate or piracy etc. We can completely remove such terms from the trie when the regular update happens, meanwhile, we can add a filtering layer on each server which will remove any such term before sending them to users.

**What could be different ranking criteria for suggestions?** In addition to a simple count, for terms ranking, we have to consider other factors too, e.g., freshness, user location, language, demographics, personal history etc.

## 4. Permanent Storage of the Trie

**How to store trie in a file so that we can rebuild our trie easily - this will be needed when a machine restarts?** We can take a snapshot of our trie periodically and store it in a file. This will enable us to rebuild a trie if the server goes down. To store, we can start with the root node and save the trie level-by-level. With each node, we can store what character it contains and how many children it has. Right after each node, we should put all of its children. Let's assume we have the following trie:



If we store this trie in a file with the above-mentioned scheme, we will have: "C2,A2,R1,T,P,O1,D". From this, we can easily rebuild our trie.

If you've noticed, we are not storing top suggestions and their counts with each node. It is hard to store this information; as our trie is being stored top down, we don't have child nodes created before the parent, so there is no easy way to store their references. For this, we have to recalculate all the top terms with counts. This







PK

educative(/learn)

Explore(/explore)

Tracks(/tracks)

My Courses(/mycourses)

Edpresso(/edpresso)

Refer a Friend(/refer-a-friend)

Create Courses/grokking-the-system-design-interview/mE2XkgGF

Grokking the System Design Interview

(/collection/5668639101419520/56490502)

87% completed

Q

Search Course

●

Designing a URL Shortening service like TinyURL  
(/courses/grokking-the-system-design-interview/m2ygV4E81AR)

●

Designing Pastebin  
(/courses/grokking-the-system-design-interview/3jyvQ3pg6KO)

●

Designing Instagram  
(/courses/grokking-the-system-design-interview/m2yDVZnQ8IG)

●

Designing Dropbox  
(/courses/grokking-the-system-design-interview/m22Gymjp4mG)

●

Designing Facebook Messenger  
(/courses/grokking-the-system-design-interview/B8R22v0wqJo)

●

Designing Twitter  
(/courses/grokking-the-system-design-interview/m2G48X18NDO)

●

Designing Youtube or Netflix  
(/courses/grokking-the-system-design-interview/xV26VjZ7yMI)

●

Designing Typeahead Suggestion  
(/courses/grokking-the-system-design-interview/mE2XkgGRnmp)

●

Designing an API Rate Limiter  
(/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)

○

Designing Twitter Search  
(/courses/grokking-the-system-design-interview/xV9mMjj74gE)

○

Designing a Web Crawler  
(/courses/grokking-the-system-design-interview/NE5LpPrWrKv)

●

Designing Facebook's Newsfeed  
(/courses/grokking-the-system-design-interview/gxpWJ3ZKYwl)

●

Designing Yelp or Nearby Friends  
(/courses/grokking-the-system-design-interview/B8rpM8E16LQ)

●

Designing Uber backend  
(/courses/grokking-the-system-design-interview/YQVkj548NM)

●

Design Ticketmaster (\*New\*)  
(/courses/grokking-the-system-design-interview/YQyq6mBKq4n)

●

Additional Resources  
(/courses/grokking-the-system-design-interview/)

#

We should realize that caching the top searched terms will be extremely helpful in our service. There will be a small percentage of queries that will be responsible for most of the traffic. We can have separate cache servers in front of the trie servers holding most frequently searched terms and their typeahead suggestions. Application servers should check these cache servers before hitting the trie servers to see if they have the desired searched terms. This will save us time to traverse the tri.

We can also build a simple Machine Learning (ML) model that can try to predict the engagement on each suggestion based on simple counting, personalization, or trending data, and cache these terms beforehand.

8. Replication and Load Balancer

#

We should have replicas for our trie servers both for load balancing and also for fault tolerance. We also need a load balancer that keeps track of our data partitioning scheme and redirects traffic based on the prefixes.

9. Fault Tolerance

#

**What will happen when a trie server goes down?** As discussed above we can have a master-slave configuration; if the master dies, the slave can take over after failover. Any server that comes back up, can rebuild the trie based on the last snapshot.

10. Typeahead Client

#

We can perform the following optimizations on the client side to improve user's experience:

1. The client should only try hitting the server if the user has not pressed any key for 50ms.

2. If the user is constantly typing, the client can cancel the in-progress requests.

3. Initially, the client can wait until the user enters a couple of characters.

4. Clients can pre-fetch some data from the server to save future requests.

5. Clients can store the recent history of suggestions locally. Recent history has a very high rate of being reused.

6. Establishing an early connection with the server turns out to be one of the most important factors. As soon as the user opens the search engine website, the client can open a connection with the server. So when a user types in the first



PK

educative

(/learn)

Explore

(/explore)

Tracks

(/tracks)

My Courses

(/mycourses)

Edpresso

(/edpresso)

Refer a Friend

(/refer-a-friend)

Create Courses

(/courses/grokking-the-system-design-interview/mE2XkgGF)

Grokking the System Design Interview

(/collection/5668639101419520/56490502)

87% completed

Search Course

Designing a URL Shortening service like TinyURL

(/courses/grokking-the-system-design-interview/m2ygV4E81AR)

Designing Pastebin

(/courses/grokking-the-system-design-interview/3jyvQ3pg6KO)

Designing Instagram

(/courses/grokking-the-system-design-interview/m2yDVZnQ8IG)

Designing Dropbox

(/courses/grokking-the-system-design-interview/m22Gymjp4mG)

Designing Facebook Messenger

(/courses/grokking-the-system-design-interview/B8R22v0wqJo)

Designing Twitter

(/courses/grokking-the-system-design-interview/m2G48X18NDO)

Designing Youtube or Netflix

(/courses/grokking-the-system-design-interview/xV26VjZ7yMI)

Designing Typeahead Suggestion

(/courses/grokking-the-system-design-interview/mE2XkgGRnmp)

Designing an API Rate Limiter

(/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)

Designing Twitter Search

(/courses/grokking-the-system-design-interview/xV9mMjj74gE)

Designing a Web Crawler

(/courses/grokking-the-system-design-interview/NE5LpPrWrKv)

Designing Facebook's Newsfeed

(/courses/grokking-the-system-design-interview/gxpWJ3ZKYwl)

Designing Yelp or Nearby Friends

(/courses/grokking-the-system-design-interview/B8rpM8E16LQ)

Designing Uber backend

(/courses/grokking-the-system-design-interview/YQVkjP548NM)

Design Ticketmaster (\*New\*)

(/courses/grokking-the-system-design-interview/YQyq6mBKq4n)

Additional Resources

(/courses/grokking-the-system-design-interview/mE2XkgGF)

character, the client doesn't waste time in establishing the connection.

7. The server can push some part of their cache to CDNs and Internet Service Providers (ISPs) for efficiency.

11. Personalization

#

Users will receive some typeahead suggestions based on their historical searches, location, language, etc. We can store the personal history of each user separately on the server and also cache them on the client. The server can add these personalized terms in the final set before sending it to the user. Personalized searches should always come before others.

← Back

(/courses/grokking-the-system-design-interview/xV26VjZ7yMI)

Next →

COMPLETED

(/courses/grokking-the-system-design-interview/3jYKmrVAPGQ)

Report an Issue

Ask a Question

(https://discuss.educative.io/c/grokking-the-system-design-gurus/system-design-problems-designing-typeahead-suggestion)