System Design (/courses/system-design)

☐ Hide Notes

/ Storage Scalability (/courses/system-design/topics/storage-scalability/) / Design Cache
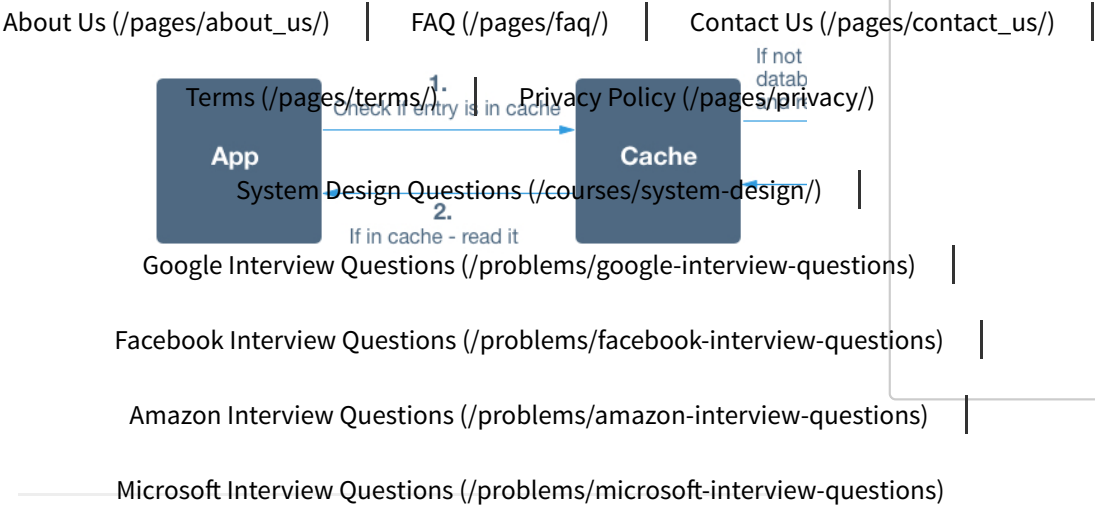
Design Cache

Bookmark

Notes

All Notes (/profile/rsubbu5

//notes/?ref=problem-page

```
cachesize - few TBs
queries per second
availability
consistency
LRU eviction strategy
Access pattern
write through
write around
write back

QPS
cache size = 30TB
key size = 100 bytes
num keys = 30 G keys
```

Design a distributed key value caching system, like Memcached or Redis.

About Us (/pages/about_us/)   FAQ (/pages/faq/)   Contact Us (/pages/contact_us/)

Terms (/pages/terms/)   Privacy Policy (/pages/privacy/)

App   **Cache**

System Design Questions (/courses/system-design/)

Google Interview Questions (/problems/google-interview-questions)

Facebook Interview Questions (/problems/facebook-interview-questions)

Amazon Interview Questions (/problems/amazon-interview-questions)

Microsoft Interview Questions (/problems/microsoft-interview-questions)

f Like Us (https://www.facebook.com/interviewbit)   🐦 Follow Us (https://twitter.com/interview_bit)

✉ Email (mailto:hello@interviewbit.com)

● **Features:**

We are in beta! We would love to hear your feedback.

Loved InterviewBit? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

> " *This is the first part of any system design interview, coming up with the features which the system should support. As an interviewee, you should try to list down all the features you can think of which our system should support. Try to spend around 2 minutes for this section in the interview. You can use the notes section alongside to remember what you wrote.* "

Q: What is the amount of data that we need to cache?
A: Let's assume we are looking to cache on the scale of Google or Twitter. The total size of the cache would be a few TBs.

Q: What should be the eviction strategy?
A: It is possible that we might get entries when we would not have space to accommodate new entries. In such cases, we would need to remove one or more entries to make space for the new entry.

Q: What should be the access pattern for the given cache?
A: There are majorly three kinds of caching systems :
Write through cache : This is a caching system where writes go through the cache and write is confirmed as success only if writes to DB and the cache BOTH succeed. This is really useful for applications which write and re-read the information quickly. However, write latency will be higher in this case as there are writes to 2 separate systems.
Write around cache : This is a caching system where write directly goes to the DB. The cache system reads the information from DB incase of a miss. While this ensures lower write load to the cache and faster writes, this can lead to higher read latency incase of applications which write and re-read the information quickly.
Write back cache : This is a caching system where the

write is directly done to the caching layer and the write is confirmed as soon as the write to the cache completes. The cache then asynchronously syncs this write to the DB. This would lead to a really quick write latency and high write throughput. But, as is the case with any non-persistent / in-memory write, we stand the risk of losing the data incase the caching layer dies. We can improve our odds by introducing having more than one replica acknowledging the write (so that we don't lose data if just one of the replica dies).

● Estimation:

> ❝ This is usually the second part of a design interview, coming up with the estimated numbers of how scalable our system should be. Important parameters to remember for this section is the number of queries per second and the data which the system will be required to handle.
> Try to spend around 5 minutes for this section in the interview. ❞

❓◀ Total cache size : Let's say 30TB as discussed earlier.

Q: What is the kind of QPS we expect for the system?

A: This estimation is important to understand the number of machines we will need to answer the queries. For example, if our estimations state that a single machine is going to handle 1M QPS, we run into a high risk of high latency / the machine dying because of queries not being answered fast enough and hence ending up in the backlog queue.

Again, let's assume the scale of Twitter / Google. We can expect around 10M QPS if not more.

6 ()

Q: What is the number of machines required to cache?

A: A cache has to be inherently of low latency. Which means all cache data has to reside in main memory. A production level caching machine would be 72G or 144G of RAM. Assuming beefier cache machines, we have 72G of main memory for 1 machine. Min. number of machine required = 30 TB / 72G which is close to 420 machines. Do know that this is the absolute minimum. Its possible we might need more machines because the QPS per machine is higher than we want it to be.

6 ()

● **Design Goals:**

> **Latency** - Is this problem very latency sensitive (Or in other words, Are requests with high latency and a failing request, equally bad?). For example, search typeahead suggestions are useless if they take more than a second.
> **Consistency** - Does this problem require tight consistency? Or is it okay if things are eventually consistent?
> **Availability** - Does this problem require 100% availability?
>
> *There could be more goals depending on the problem. It's possible that all parameters might be important, and some of them might conflict. In that case, you'd need to prioritize one over the other.* "

Q: Is Latency a very important metric for us?

A: Yes. The whole point of caching is low latency.

6 ()

Q: Consistency vs Availability?

A: Unavailability in a caching system means that the caching machine goes down, which in turn means that we have a cache miss which leads to a high latency.
As said before, we are caching for a Twitter / Google like system. When fetching a timeline for a user, I would be okay if I miss on a few tweets which were very recently posted as long as I eventually see them in reasonable time.
Unavailability could lead to latency spikes and increased load on DB. Choosing from consistency and availability, we should prioritize for availability.

5 ()

● **Deep Dive:**

> ❝ Lets dig deeper into every component one by one. Discussion for this section will take majority of the interview time(20-30 minutes). ❞

Q: How would a LRU cache work on a single machine which is single threaded?

Q: What if we never had to remove entries from the LRU cache because we had enough space, what

would you use to support and get and set?

A: A simple map / hashmap would suffice.

Q: How should we modify our approach if we also have to evict keys at some stage?

A: We need a data structure which at any given instance can give me the least recently used objects in order. Let's see if we can maintain a linked list to do it. We try to keep the list ordered by the order in which they are used.

So whenever, a get operation happens, we would need to move that object from a certain position in the list to the front of the list. Which means a delete followed by insert at the beginning. Insert at the beginning of the list is trivial. How do we achieve erase of the object from a random position in least time possible? How about we maintain another map which stores the value to the corresponding linked list node.

Ok, now when we know the node, we would need to know its previous and next node in the list to enable the deletion of the node from the list. We can get the next in the list from next pointer ? What about the previous node ? To encounter that, we make the list doubly linked list.

Head over to https://www.interviewbit.com/problems
/least-recently-used-cache/
(https://www.interviewbit.com/problems/least-recently-used-cache/) to write code and see if you completely got it.

and implement a LRU cache using a linked list and a map. The Map stores the value to the corresponding linked list node and is useful to move the recently accessed node to the front of the list.
Head over to https://www.interviewbit.com/problems /least-recently-used-cache/ (https://www.interviewbit.com/problems/least-recently-used-cache/) to write code and see if you completely got it.

4 ()

Q: How would a LRU cache work on a single machine which is multi threaded ?

Q: How would you break down cache write and read into multiple instructions?

4 ()

A:
Read path : Read a value corresponding to a key. This requires :
Operation 1 : A read from the HashMap and then,
Operation 2 : An update in the doubly LinkedList

Write path : Insert a new key-value entry to the LRU cache. This requires :
If the cache is full, then
Operation 3: Figure out the least recently used item from the linkedList
Operation 4: Remove it from the hashMap
Operation 5: Remove the entry from the linkedList.
Operation 6: Insert the new item in the hashMap
Operation 7: Insert the new item in the linkedList.

We are in beta! We would love to hear your   **Loved InterviewBit? Write us a testimonial.**
Q: How would you prioritize above operations to keep **(http://www.quora.com/What-is-your-review-**
feedback.   **of-InterviewBit)**

latency to a minimum for our system?

② ()

A: As is the case with most concurrent systems, writes compete with reads and other writes. That requires some form of locking when a write is in progress. We can choose to have writes as granular as possible to help with performance.

Read path is going to be highly frequent. As latency is our design goal, Operation 1 needs to be really fast and should require minimum locks. Operation 2 can happen asynchronously. Similarly, all of the write path can happen asynchronously and the client's latency need not be affected by anything other than Operation 1. Let's dig deeper into Operation 1. What are the things that Hashmap is dealing with?

Hashmap deals with Operation 1, 4 and 6 with Operation 4 and 6 being write operations. One simple, but not so efficient way of handling read/write would be to acquire a higher level Read lock for Operation 1 and Write lock for Operation 4 and 6.

However, Operation 1 as stressed earlier is the most frequent (by a huge margin) operation and its performance is critical to how our caching system works.

Q: How would you implement HashMap?

⑥ ()

A: The HashMap itself could be implemented in multiple ways. One common way could be hashing with linked list (colliding values linked together in a linkedList) :

Let's say our hashmap size is N and we wish to add (k,v) to it

Let H = size N array of pointers with every element initialized to NULL

H[g] = newEntry

More details at https://en.wikipedia.org
/wiki/Hash_table (https://en.wikipedia.org
/wiki/Hash_table)
Given this implementation, we can see that instead of
having a lock on a hashmap level, we can have it for
every single row. This way, a read for row i and a write
for row j would not affect each other if i != j. Note that
we would try to keep N as high as possible here to
increase granularity.

A: The key to understanding and optimizing concurrency
problems lies in breaking the problem down into as
granular parts as possible.
As is the case with most concurrent systems, writes
compete with reads and other writes, which requires some
form of locking when a write is in progress. We can choose
to have writes as granular as possible to help with
performance. Instead of having a lock on a hashmap level
if we can have it for every single row, a read for row i and a
write for row j would not affect each other if i != j. Note
that we would try to keep N as high as possible here to
increase granularity.

(5) ()

Q: Now that we have sorted how things look on a single
server, how do we shard?

Q: What QPS would a machine have to handle if we
shard in blocks of 72GB?

(4) ()

A: In estimation section, we saw total data we would
have to store is 30TB. For every chunk of data, we

store a copy in the hashmap and we store an entry (
without the value) in a linkedList. Let's assume that

the size of value is big enough to ignore overheads like an entry in linkedList.
We can accommodate 72G of data on every single machine (We have neglected process memory overheads for the time being). With that, we would need 420 machines.
With that config, every machine would handle around 23000 QPS.

Q: Will our machines be able to handle qps of 23000?

⟨**7**⟩ ()

A: CPU time available for 23k queries : 1 second * 4 = 4 seconds
CPU time available per query = 4 * 1000 * 1000 / 23000 microseconds = 174us. Can we handle entries into a hashmap of size 72G with a CPU time of 174us (Do note that context switches has its own overhead. So, even with a perfectly written asynchronous server, we would have much less than 174us on our hand). Make sure you know about the latency numbers from here : https://gist.github.com/jboner/2841832 (https://gist.github.com/jboner/2841832). The actual answer depends on the distribution of read vs write traffic, the size of the value being read, the throughput capacity of our server.

Q: What if we shard among machines with 16GB of RAM?

⟨**3**⟩ ()

A: Number of shards = 30 * 1000 / 16 = 1875
This leads to a QPS of approximately 5500 per shard which should be reasonable (Note that with lower main memory size, CPU cycles required for access

lower). We would now need to decide the shard number for every key. A simple way to do it

would be to shard based on hash(key) %
TOTAL_SHARDS The hash function might differ based
on expected properties of the key. If the key is an
auto-incremental user_id, then hash(key) = key
hashing might work well.

One downside to this is that if the total number of
shard changes, all the currently cached data becomes
invalid and all requests would have to hit the DB to
warm up the cache. The other way to do it would be
to use consistent hashing with multiple copies of
every shard on the ring (Read more about consistent
hashing at https://en.wikipedia.org
/wiki/Consistent_hashing (https://en.wikipedia.org
/wiki/Consistent_hashing)). This would perform well
as new shards are added.

A: Recall that the total data we have is 30TB and for every
chunk of data, we store a copy in the hashmap and we
store an entry (without the value) in a linkedList. Lets
assume that the size of value is big enough to ignore
overheads like an entry in linkedList. We can
accommodate 72G of data on every single machine (We
have neglected process memory overheads for the time
being). With that, we would need 420 machines which
would lead to a QPS of 23000 which is not easily feasible.
So we rather create shards of 16GB, 1875 shards each
supporting qps of 5500.

💬 **0** ()

Q: What happens when a machine handling a shard goes
down?

A: If we only have one machine per shard, then if the
machine goes down, all requests to that shard will start

hitting the DB and hence there will be elevated latency.
As mentioned in the design goals, we would like to reduce
high latency cases as much as possible.

If we have a lot of machines, one way to avoid these cases would be to have multiple machines per shard where they maintain exactly the same amount of data.

A read query for the shard could go to all the servers in the shard and we can use the data from the one that responds first. This takes care of one machine going down, but introduces a bunch of other complications. If occasional high latency is not a big issue wrt product, its better to stick to one server per shard (Less maintenance overhead and a much simpler system).

Complications of multiple servers : Since we have multiple servers maintaining the same data, it is possible that the data is not in sync between the servers. This means that a few keys might be missing on some of the servers, and a few servers might have older values for the same keys (Assuming we support updates as well).

Imagine a case when one of the server goes down, misses a bunch of additions and updates, and then comes back up.

There are few ways we can approach this :

Master slave technique : There is only one active server at a time in a shard and it has a follower which keeps getting the update. When the master server goes down, the slave server takes over as the master server. Master and slave can maintain a change log with version number to make sure they are caught up.

If we are fine with all servers becoming eventually consistent, then we can have one master (taking all the write traffic) and many slaves where slaves can service the read traffic as well.
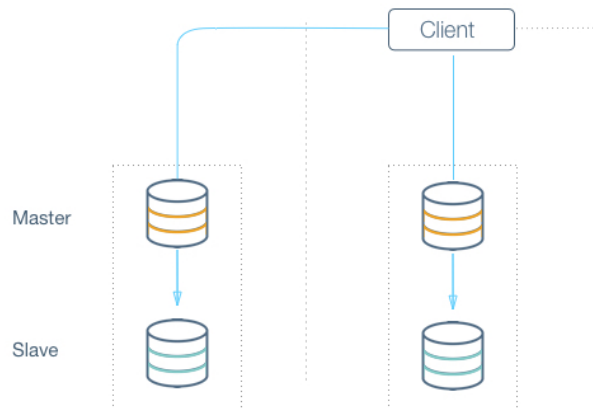
4 ()

✓ ◀ 🏆 **You have now mastered this problem!**

Discussion

▲ dooby_do (/profile/dooby_do) 8 months ago
10 I was asked this problem in Google's interview for L5 position.
▼ reply

> ▲ kaidul (/profile/kaidul) 6 months ago
> -3 For curiosity, did you make it?
> ▼ reply
>
> > ▲ dooby_do (/profile/dooby_do) 6 months ago
> > 4   I bagged the offer, however did not end up joining them :)
> > ▼ reply
> >
> > > ▲ prashant_upadhyay (/profile/prashant_upadhyay) 5
> > > -3 months ago
> > > ▼ i like it :D
> > >   reply

We are in beta! We would love to hear your   **Loved InterviewBit? Write us a testimonial.**

feedback.eecee (/profile/deeteecee) 26 days**(http://www.quora.com/What-is-your-review-**
         this is the real dream   **of-InterviewBit)**

0   reply
⬇

⬆ kumar955 (/profile/kumar955) 8 months ago
4   I think in most of the interviews, candidate is expected to calculate
⬇ Storage, RAM SIZE, Upstream and DownStream bandwidth, Num of Servers
    etc. (I meant we cannot assume most of things except one or two like Data
    Size).
    reply

⬆ venknar (/profile/venknar) 8 months ago
1   It is somewhat same as designing a distributed hash table.
⬇ reply

        ⬆ rui_tong (/profile/rui_tong) 4 days ago
        0   What's the difference?
        ⬇ reply

⬆ deep_saxena (/profile/deep_saxena) 8 months ago
2   How the QPS is calculated here? I am not getting the exact calculation. Can
⬇ somebody explain it ?
    reply

        ⬆ venknar (/profile/venknar) 8 months ago
        1   The way QPS is calculated is as follows : Earlier we mentioned that
        ⬇ each machine would have a RAM of 72 GB of RAM. For serving 30TB
            of cache, the number of machines required would be 30 TB / 72G
            which is close to 420. Assume that we have 420 machines to server
            30 TB of distributed cache. Now regarding the QPS the
            requirement was 10 M
            reply

        ⬆ venknar (/profile/venknar) 8 months ago
        2   Now per machine the QPS would be 10M / 420 = Approximately
        ⬇ 23000 QPS. So this meant per machine should be able to handle
            23,00 QPS. The approach is similar to how we decided on the
            number of machines based on the per machine RAM and the total
            cache size. Similarly for the QPS, it is based on the total QPS /
            number of machines.
            reply

        ⬆ venknar (/profile/venknar) 8 months ago
        1   Next assuming that a machine has to serve 23,000 QPS then we
        ⬇ look at each machine has 4 core and then we calculate the per
            request time as - CPU time available per query = 4 * 1000 * 1000 /
            23000 microseconds = 174us (Note everything is converted to

milliseconds.) So the machines have to return the query in 174 us. This is the way the QPS is derived. Then based on the read / write traffic and the latency numbers as per the https://gist.github.com /jboner/2841832, the QPS is further refined by increasing the number of machines.
reply

> ▲ skcoder (/profile/skcoder) 8 months ago
> 0 I am coming up with 23,000 queries / 1 sec.* 1,000,000 us/
> ▼ 1 sec = 1 query / 40us. So 40us per query service time is required. I don't see why we would calculate this based on CPU time, or why my # is so far off. Only thing I can think of is that if I multiply by 4 CPUs I have each CPU servicing 1 query every 160us (close to the 174 above)- but why would we calculate this per CPU rather than per server?
> reply
>
> > ▲ skcoder (/profile/skcoder) 8 months ago
> > 0 OK, I figured it out. Each *CPU* has to return the
> > ▼ query in 174us, but each *CPU* has to service ~6K queries/sec, not 23K. 23K is per machine.
> > reply
>
> ▲ amandeep_gautam_181 (/profile/amandeep_gautam_181) 29
> 0 days ago
> ▼ How do we get this: 4 * 1000 * 1000 as CPU time?
> reply

▲ daniele_broccolo (/profile/daniele_broccolo) 8 months ago
3 I clicked on one of the links for the implementation of the least recently
▼ used cache.... the counter started, but I did not have the time to complete the task. I think you should advice before open the problems pages
reply

> ▲ catlover (/profile/catlover) 8 months ago
> 0 Yeah, that's right. There should be an advice. It also cause problem
> ▼ when landing from google search results.
> reply

▲ amitvc (/profile/amitvc) 8 months ago
1 What about sharding algorithms ? How does the caller know which server
▼ to go for a specific key ? Like memcache uses hashing to determine which server the key resides on.
reply

We are in beta! We would love to hear your   **Loved InterviewBit? Write us a testimonial.**
feedback.   **(http://www.quora.com/What-is-your-review-**
> ▲ kousik_nath (/profile/kousik_nath) 8 months ago   **of-InterviewBit)**

1 You can use consistent hashing to determine the exact shrad for
⬇ the key. But in case of cache server failure, consistent hashing
can't alone help to recover the data. You must have replication
cache servers so that disaster recovery happens from the
replicated cache server data. You can google about consistent
hashing. There are a lot of articles about it.
reply

⬆ amitvc (/profile/amitvc) 8 months ago
1 Thanks Koushik. I am aware about consistent hashing. My
⬇ question was more about no discussion on the topic of
sharding mentioned in the problem.
reply

⬆ rashmiiiith (/profile/rashmiiiith) 8 months ago
1 I don't know how relevant to your question but
⬇ this paper is worth reading .
reply

⬆ rashmiiiith (/profile/rashmiiiith) 8 months ago
0 http://www.allthingsdistributed.com/files
⬇ /amazon-dynamo-sosp2007.pdf
reply

⬆ amitvc (/profile/amitvc) 7 months ago
1 Thank you. I believe Cassandra also uses
⬇ consistent hashing
reply

⬆ velupula (/profile/velupula) 7 months ago
1 partitioning and fault tolerance are important topics that must have been
⬇ covered in this design.
reply

⬆ aniket15 (/profile/aniket15) 5 months ago
0 Latency is always an important metric for distributed and scalable
⬇ systems. Can there be scenarios where its not that important?
reply

⬆ snehasis_ghosh (/profile/snehasis_ghosh) 13 days ago
0 Something like Dropbox / Google Drive.
⬇ reply

document I didn't intend to be public. When I delete that file from the folder, I would assume that nobody can access the file immediately. Everything else is secondary. If others have to wait a few seconds before they can access the folder, that's ok. In this case Consistency > Availability. And if it introduces some latency to guarantee Consistency, so be it.

reply

⬆ Titan99 (/profile/Titan99) 3 months ago
0 I have a doubt, please correct me if I am wrong. We divided 30 TB of data
⬇ into 420 machines having each 72GB of data. Isn't this also called sharding ? And, then we again split each machine data to 16GB shards ? Aren't we doing sharding twice, if seen from higher level ?

reply

⬆ shino_kurian (/profile/shino_kurian) 3 months ago
0 The case in discussion is either 420 machine of 72GB each or 1875
⬇ machines of 16GB each.

reply

⬆ mag (/profile/mag) 3 months ago
1 What is actually meaning of distributed here? all data spread across
⬇ multiple nodes in a cluster (sharding)? or in case all data can reside in one node and implement master-slave or master-master replication? what is the goal? availability or fastest access? as per my understanding, performance is the primary goal of a cache system and to achieve that, if we are designing a very large hash table which cannot reside on a single node and we have split it to multiple nodes, would you say it distributed or sharded cache system?

reply

⬆ niufei8888 (/profile/niufei8888) 2 months ago
0 This tutorial rocks!
⬇ reply

⬆ rkouj (/profile/rkouj) about 1 month ago
0 Scale is not more than 60k per second.
⬇ reply

⬆ Clove (/profile/Clove) about 1 month ago
-1 Has anyone solved the above problem?
⬇ reply

We are in beta! We would love to hear your   **Loved InterviewBit? Write us a testimonial.**

⬆ haandol (/profile/haandol) about 1 month ago
feedback!   **(http://www.quora.com/What-is-your-review-**
This is just amazing.   **of-InterviewBit)**

-1 reply

⬇

Write your comment here. Press Enter to submit.