#ASLI ENGINEERING

Designing Review System for Booking.com

SWIPE

BY

ARPIT BHAYANI

Designing and Scaling Booking.com's Review System for 10000 RPS

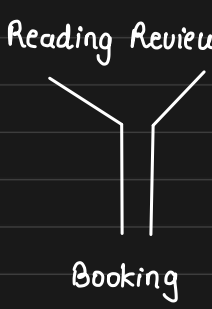
You can book flights, hotels, stays, tours, and much more on Booking and Reviews System is the core of the platform

1. users use the reviews to make an informed decision
2. reviews are authentic
   ↳ user's cannot post review without making a booking

Review system is a high-throughput system

High availability and low latency is essential because reviews act as a booking funnel

People will not book without reading reviews *



Review Service will be a simple REST based that exposed API to get, create, edit and delete reviews.

Traffic Peak ~ 10,000 rps with p99 of 50ms

↕ indicates

reviews (mostly) are served from the cache or prematerialized views on the relational databases

Amount of Data

250 million reviews and each review contains

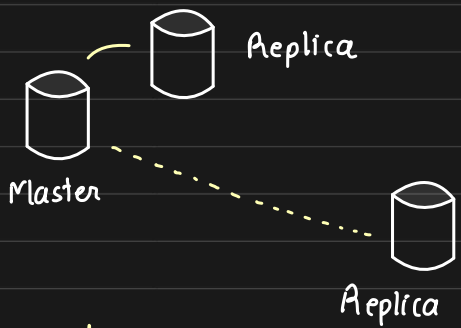
1. answers to some objective questions
2. ratings on various parameters
3. textual feedback

if we assume 2KB per review then total data will be 250M x 2KB = 500GB

To ensure response time under 50ms we have to shard the DB + have a bunch of replicas to protect us against network / hardware failures.

+ availability zone failures

* Availability is most important



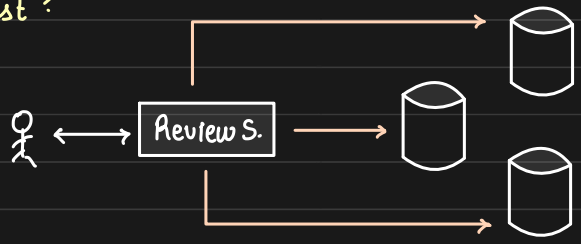
Sharding

The reviews are sharded by accomodation id so that all reviews of an accomodation are present on the same node

But, how do we route the request?

Simplest one is the Hash based

acc_id % # shard = i



challenge

as soon as we add/remove nodes

our routing function changes requiring a massive data shuffle.

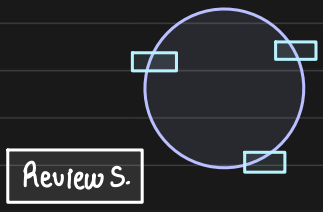
↳ re-partitioning of the entire data

Solution to this problem is pretty standard

We use Consistent Hashing

↳ data ownership

↳ with minimal data movement



* I have a blog post where I explained and implemented Consistent Hashing

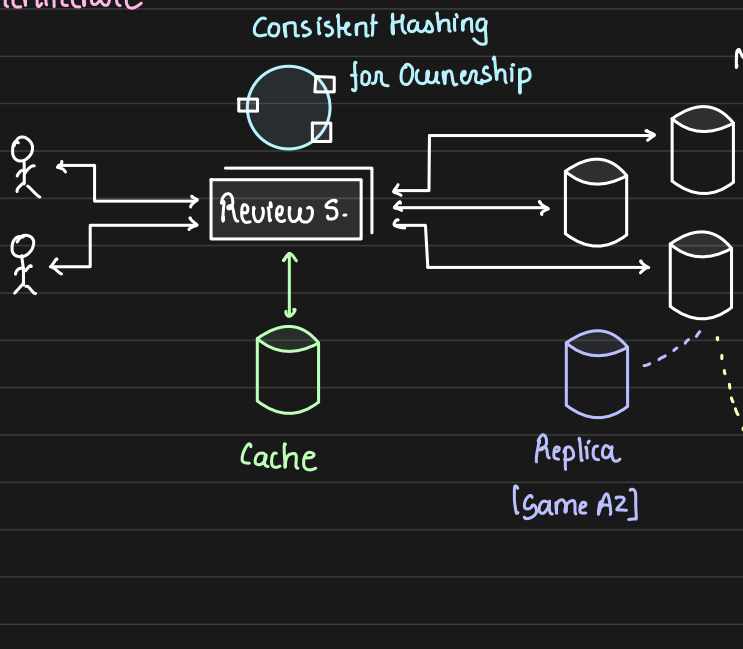
arpitbhayani.me/blogs/consistent-hashing

Practicality of resizing

1. add/remove nodes
2. copy the data that needs to be moved
3. notify Review Service to start consuming new ring

Routing

Architecture



1. Materialized Views
2. High cache hit ratio
3. Availability is critical