#ASLI ENGINEERING

GitHub Outage: Downtime due to reversing an Index

SWIPE

BY

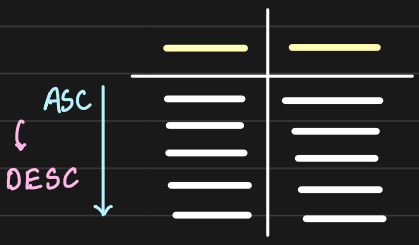
ARPIT BHAYANI

Dissecting GitHub Outage

Downtime due to "reversing" an index

What happened ?

GitHub ran a database migration
to flip the order of an index
to improve query performance



Say, we have a multi-column index on a table

commits

id	user_id	date

Most common use case:

Get commits on a repository
in descending order

Say, now we want

Get commits ordered by date and for each day order by user

↓ DESC ↓ ASC

This query will be inefficient because
all indexes on MySQL are ASC ordered
by default.

Hence to answer the query efficiently

We would need to have an index which is
ordered by Date in DESC order → Supported by MySQL 8
and UserID in ASC order

Hence, we have to "flip" the index

Reversing the index caused Full Table Scan
for a query generated by their ORM (Active Record) — load on the DB

1. Creating a new index itself would require a Full Table Scan
2. Say, you changed the order of the index, but

1. some other query needed DATE, USERID ASC

which is much more frequent would now become inefficient

2. some queries might not use this new index
because DB engine thought it would be "inefficient"
↳ because of the query we provided
↳ or its own estimations

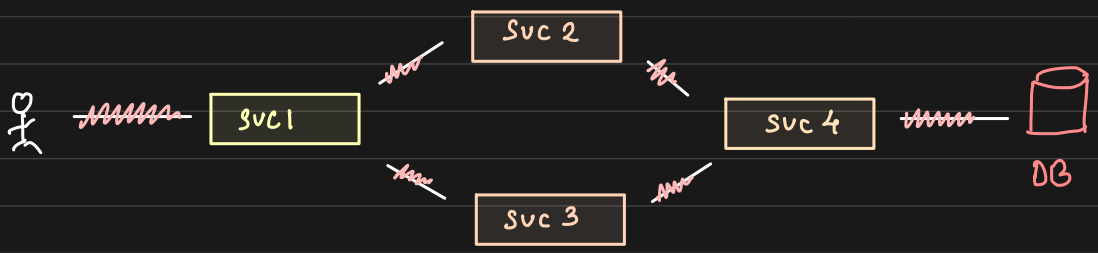
We can use INDEX HINTS to tell our DB to use a particular index for a particular query

```
SELECT .....
FROM .....
USE INDEX (idx_.....);
```

Cascading Effect

Because a query was doing a Full Table Scan

- ↳ Put load on the DB
- ↳ Response time of the service increased
- ↳ lead to timeouts of the request
- ↳ Response time of depending services increased
- ↳ timeouts cascaded !!



Note: Whenever there is a synchronous dependency b/w services there is always a chance of Cascading Failures

Key Takeaways:

1. Do not blindly trust your ORM (Active Record, Djongo ORM, SQLAlchemy, Hibernate)

   ORM makes our lives simpler, but they may not generate the most optimized SQL queries

   Periodically audit the Queries generated by ORMs

2. Always check the Query Execution Plan

   Run your migration on staging / QA environment

   and see the Query execution plan for related Queries

   Check for any abberation or deviation

3. Audit the queries and indexes they use

   prepare an inventory of queries and indexes they use

   So that we can quickly test for regressions