Primary Bottlenecks That Hurt the Scalability of our Application

We'll cover the following

- Database
- Application architecture
- Not using caching in the application wisely
- Inefficient configuration and setup of load balancers
- Adding business logic to the database
- Not picking the right database
- At the code level

There are several points in a web application that can become a bottleneck and can hurt the scalability of our application. Let’s take a look at them.

Database#

Consider that, we have an application that appears to be well architected. Everything looks good. The workload runs on multiple nodes, and it has the ability to horizontally scale.

However, the database is a poor single monolith, where just one server has the onus of handling the data requests from all the server nodes of the workload.

This scenario is a bottleneck. The server nodes work well, handle millions of requests at a point in time efficiently, yet, the response time of these requests and the latency of the application is very high due to the presence of a single database. There is only so much it can handle.

Just like workload scalability, the database needs to be scaled well.

Make wise use of database partitioning, sharding, and multiple database servers to make the module efficient.

Application architecture#

A poorly designed application's architecture can become a major bottleneck as a whole.

A common architectural mistake is not using asynchronous processes and modules whereever required rather all the processes are scheduled sequentially.

For instance, if a user uploads a document on the portal, tasks such as sending a confirmation email to the user or sending a notification to all of the subscribers/listeners to the upload event should be done asynchronously.

These tasks should be forwarded to a messaging server as opposed to doing it all sequentially and making the user wait for everything.

Not using caching in the application wisely#

Caching can be deployed at several layers of the application and it speeds up the response time by notches. It intercepts all the requests going to the database, reducing the overall load.

Use caching exhaustively throughout the application to speed up things significantly.

Inefficient configuration and setup of load balancers#

Load balancers are the gateway to our application. Using too many or too few of them impacts the latency of our application.

Adding business logic to the database#

No matter what justification anyone provides, I've never been a fan of adding business logic to the database.

The database is just not the place to put business logic. Not only does the whole application tightly coupled it puts an unnecessary load on it.

Imagine when migrating to a different database, how much code refactoring this would require.

Not picking the right database#

Picking the right database technology is vital for businesses. Need transactions and strong consistency? Pick a *relational database*. If you can do without strong consistency rather than need horizontal scalability on the fly pick a *NoSQL* database.

Trying to pull things off with a not so suitable tech always has a profound impact on the latency of the entire application in negative ways.

At the code level#

This shouldn't come as a surprise but inefficient and badly written code has the potential to take down the entire service in production, which includes:

- Using unnecessary loops or nested loops
- Writing tightly coupled code
- Not paying attention to the *Big-O complexity* while writing the code. (be ready to do a lot of firefighting in production)

In this lesson, if a few of the things are not clear to you such as strong consistency, how the message queue provides an asynchronous behavior, or how to pick the right database, don't worry. I'll discuss all that in the upcoming lessons, stay tuned.

Back

Next

Which Scalability Approach is Right fo

How to Improve and Test the Scalabili

Mark as Completed

Report an Issue