Scaling: 10K RPM vs Million RPM

Exploring the Differences

08

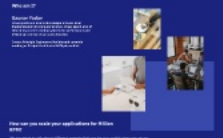
Rate Limiting, retries and circuit breakers will become new normal

10

Do you know your tail?

11

Let's talk about about Database



09

Alerts Metrics and APMs become necessity

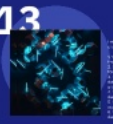
Introduction

Writing an app to run at 10K RPM (requests per minute) isn't too difficult. 10K RPM equals about 170 RPS (Requests per second). A basic MVP (minimum viable product) with key features and good code can achieve this easily.

You will need around 10 core for this all of this very efficiently. [Obviously it depends on task to task, here I am discussing about very basic tasks like login, listing, updates and basic calculations].

12

How to choose right Database?



13

How to choose right Database?

01

Do you really understanding your application

02

Caching

05

Connection Pooling

04

Stay alive we need to do handshake once you started talking: Keep Alive

07

Polling vs Pushing

03

DNS Caching

06

CDN will save your internal systems

Thats it for now.

Stay away from over engineering, if you engineering is not aligned with your business you will be wasting a lot of resource which can actually crete impact.

Everyline of code or architecture change that you do. Always keep in mind how will this imapct my CPU, memory, disk and network.

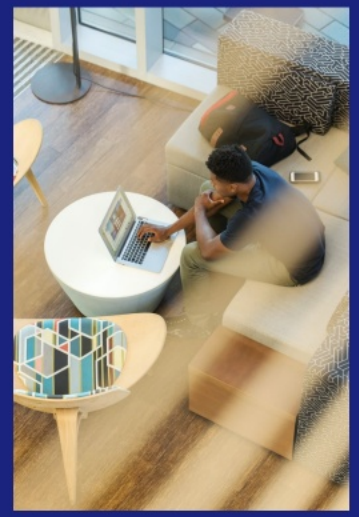
Questions or Suggestions

Who am I?

Gaurav Yadav

I love sports and love to dive deeper in lower level implementation in computer science. I have spend a lot of time of my career in tuning systems for performance and efficiency in terms of cost and utilization

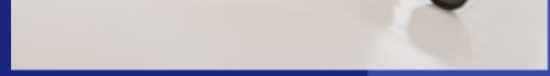
I was a Principle Engineer at OlaCabs and currently working as Pricipal Architect with Physicswallah



Introduction

Writing an app to run at 10K RPM (requests per minute) isn't too difficult. 10K RPM equals about 170 RPS (Requests per second). A basic MVP (minimum viable product) with key features and good code can achieve this easily.

You will need around 10 core for this all of this very efficiently. [Obviously it depends on task to task, here I am discussing about very basic tasks like login, listing, updates and basic calculations].



How can you scale your applications for Million RPM?

We are going to talk about different aspects that can be very useful when you strive to go for next step in your scaling journey.

Assumptions for Distributed systems:

Network is prone to disturbance. It can go down or become slow.
Your machine/VMs can go down.

Scaling: 10K RPM vs Million RPM

Exploring the Differences

08

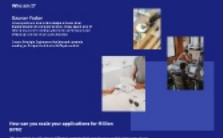
Rate Limiting, retries and circuit breakers will become new normal

10

Do you know your tail?

11

Let's talk about about Database



09

Alerts Metrics and APMs become necessity

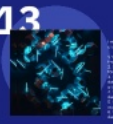
Introduction

Writing an app to run at 10K RPM (requests per minute) isn't too difficult. 10K RPM equals about 170 RPS (Requests per second). A basic MVP (minimum viable product) with key features and good code can achieve this easily.

You will need around 10 core for this all of this very efficiently. [Obviously it depends on task to task, here I am discussing about very basic tasks like login, listing, updates and basic calculations].

12

How to choose right Database?



13

How to choose right Database?

01

Do you really understanding your application

02

Caching

05

Connection Pooling

04

Stay alive we need to do handshake once you started talking: Keep Alive

07

Polling vs Pushing

03

DNS Caching

06

CDN will save your internal systems

Thats it for now.

Stay away from over engineering, if you engineering is not aligned with your business you will be wasting a lot of resource which can actually crete impact.

Everyline of code or architecture change that you do. Always keep in mind how will this imapct my CPU, memory, disk and network.

Questions or Suggestions

01

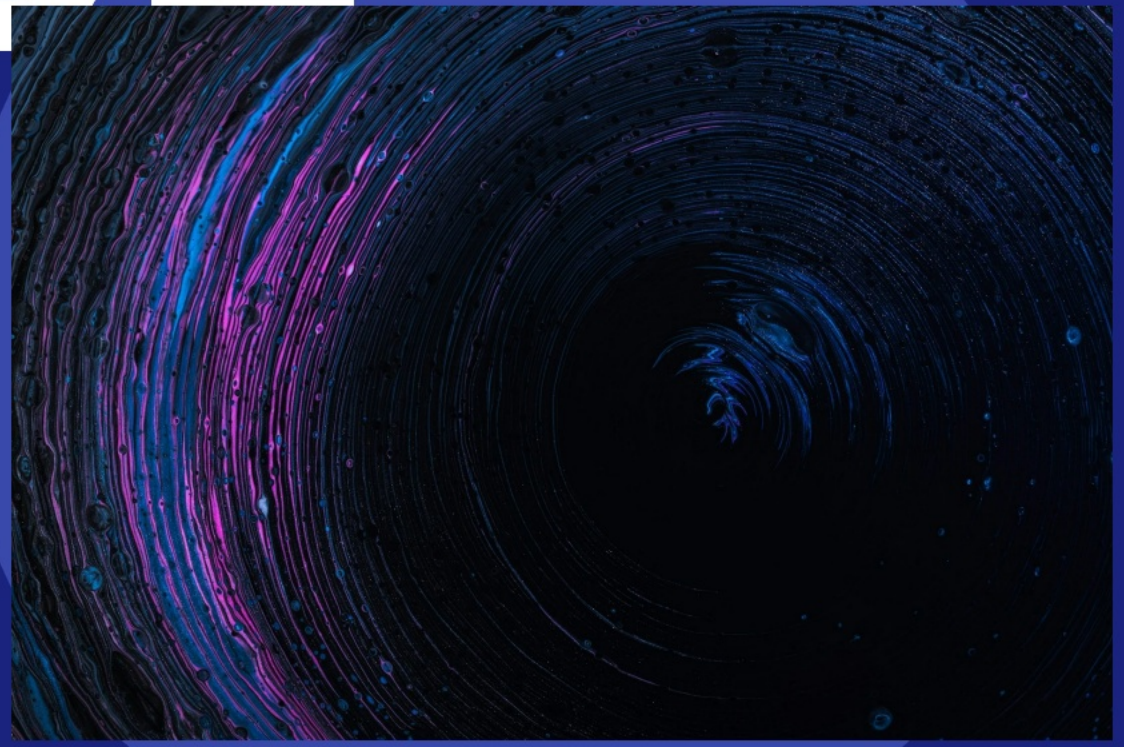
Do you really understanding your application

Try answering below 2 questions:
1. Is you application read heavy or write heavy
2. Is you application compute heavy or IO heavy

Answer these and write it down as it will help you in further slides.

Facts:
Most of the applications in the world are IO heavy. Lets see how?

02



Caching

Caching is the first thing that comes to my mind if anyone asks how to scale my application.

Caching saves your databases which are the most critical stateful part of your applcation.

The most problematic part of databases is that they are stateful.

Why is the stateful nature a problem?
This is the very thing which makes horizontal scaling tricky?

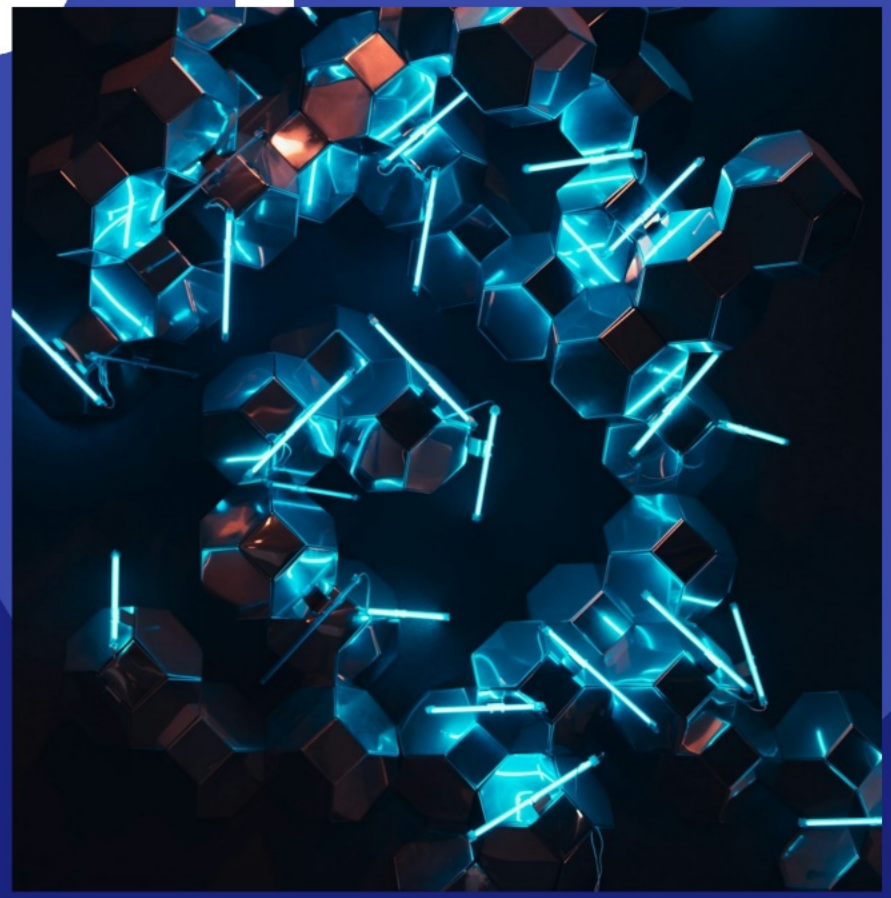
03

DNS Caching

Every connection that you try to make to any dependency involves this step. Before start the three way handshake, DNS queries are fired. When working at Million RPM scale this can cause trouble at few places.

You can run DNS cache servers or add DNS caching in your applications.

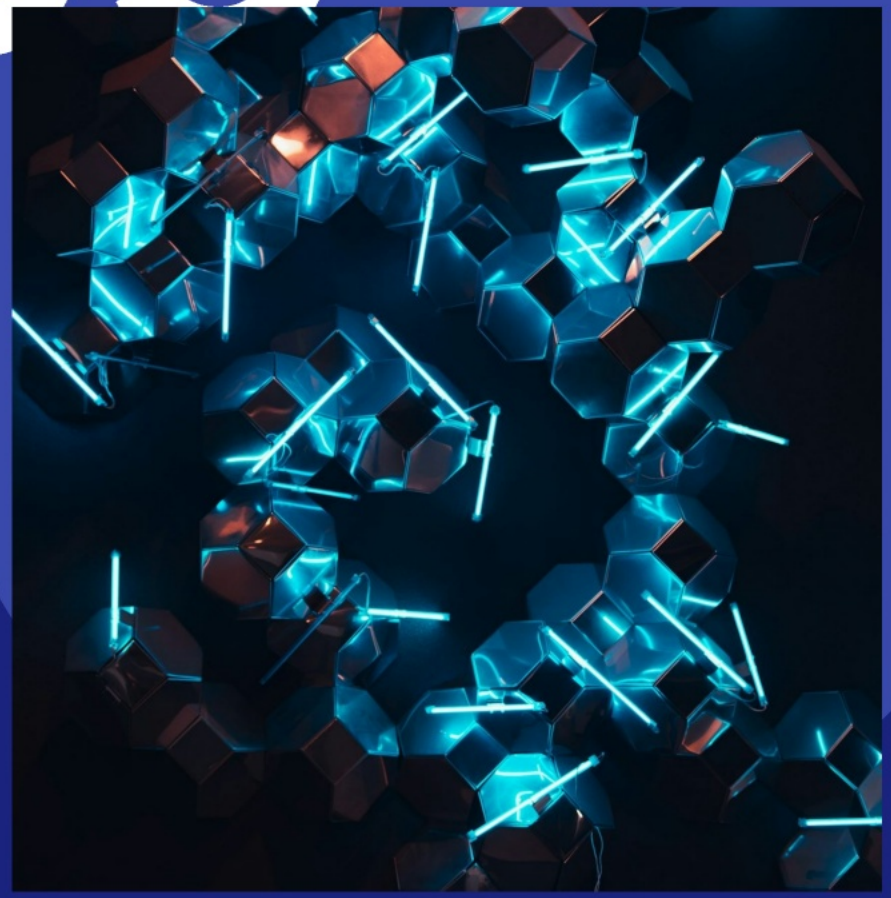
04



Stay alive no need to do handshake once you started talking: Keep Alive

This is kind of an extension of connection pooling. When you are making request to dependencies, you can request the server to keep this connection open.

05



Connection Pooling

What connection pooling means is you create a set of connection with the dependency and keep on using those instead of creating the connections for each requests.

How this helps?
Each connection use some memory and CPU, CPU usage is high when lot of connections are getting created. So creating connections at the start of the application and using them helps.

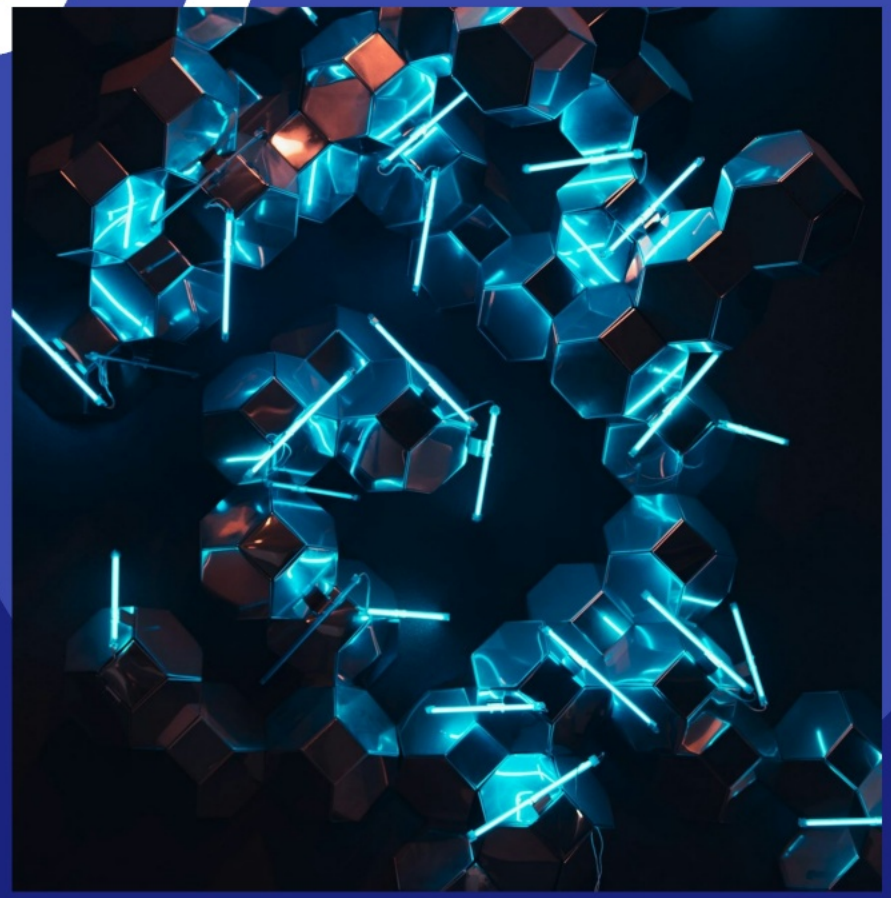
06

CDN will save your internal systems

This can bring you major win if you are able to tackle the problem of cache invalidation.

How?
CDN serves the data from the POPs and reqeusts never actually hit your gateways.

07



Polling vs Pushing

This can be a very important decision and will depend on your application. Application which needs fresh data should focus on pushing[measn socket connections] instead of polling data every second. For example: Stock market tickers

08

Rate Limiting, retries and cricuit breakers will become new normal

Rate limting is to make sure that your apps are not overloaded during spikes and shed load in case of unwanted traffic.

Circuit breaker is to make sure that in case of any issue with services, there is a graceful way of handling the error.

Retries are important so that in any of the above cases you can reqeust the resource back again and get the intended response. This can also lead to request amplification if not tackled properly.

09

Alerts Metrics and APMs become necessity

This is nothing new but to reiterate again. Anything that you cannot measure, you will not be able to improve. As you grow in scale, there are some metrics which becomes relevant which were not earlier like p99.

Also APM will help you in finding out the hotspot in you application that need immediate attention.

You want to be informed for any CPU or Memory increase because in large system the chances of cascading is very high. For example: let's say you are running 100 instances and 20 instances go down due to load. This can put load on existing 80 instances and bring them down and can stuck in loop.

10

Do you know your tail?

Tail here means tail latencies. Tail latencies are p99 or p99.9. You can define your own P9* to watch.

What exactly does p99 mean?

Why is it important?
Becasue at scale p99 will impact a larger audience and you should always keep a check on that.

11

Let's talk about about database

Databases are the most critical part of your applications. When we talk about scaling there few things that you have to do at application level like using connection pooling and proper indexing and writing perfromant queries. With all this in place we will still be needing to scale databases to serve.

How to scale reads?
Relication

How to scale writes?
Sharding

What else to consider?
Should you run a proxy in front of your database to handle connection and limit connections to database

12

How to choose right Database?

This can be a very critical decision. So before taking this decision you should know about few things.

1. Do you know about CAP? What do you want CA, AP or CP?
2. Do you understand the nature of you data?
3. Do you understand the nature of you queries?
4. OLTP vs OLAP
5. Columnar vs Row based

13

How to choose right Database?

Lets see few example. There is no direct roadmap for choosing the right database.

1. Very low Latency: You want low latency, stay away from disk. Inmemory databases.
2. CP vs AP: CA you can prefer MYSQL for AP you can opt for MongoDB or similar DB
3. For analytics you should opt for AP and columnar databases. Do you know why columnar databases for analytics?
4. Data type: Structured and unstructured, relational data.
5. Write Intensive: LSM based databases like cassandra, scylladb.
6. Do read about time series databases and vector databases. Trending right now. Why?

Scaling: 10K RPM vs Million RPM

Exploring the Differences

08

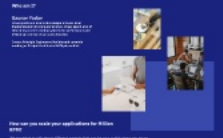
Rate Limiting, retries and circuit breakers will become new normal

10

Do you know your tail?

11

Let's talk about about Database



09

Alerts Metrics and APMs become necessity

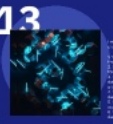
Introduction

Writing an app to run at 10K RPM (requests per minute) isn't too difficult. 10K RPM equals about 170 RPS (Requests per second). A basic MVP (minimum viable product) with key features and good code can achieve this easily.

You will need around 10 core for this all of this very efficiently. [Obviously it depends on task to task, here I am discussing about very basic tasks like login, listing, updates and basic calculations].

12

How to choose right Database?



13

How to choose right Database?

01

Do you really understanding your application

02

Caching

05

Connection Pooling

04

Stay alive we need to do handshake once you started talking: Keep Alive

07

Polling vs Pushing

03

DNS Caching

06

CDN will save your internal systems

Thats it for now.

Stay away from over engineering, if you engineering is not aligned with your business you will be wasting a lot of resource which can actually crete impact.

Everyline of code or architecture change that you do. Always keep in mind how will this imapct my CPU, memory, disk and network.

Questions or Suggestions

Thats it for now.

Stay away from over engineering, if you engineering is not aligned with your business you will be wasting a lot of resource which can actually crete impact.

Everyline of code or architecture change that you do. Always keep in mind how will this imapct my CPU, memory, disk and network.

Questions or Suggestions

Scaling: 10K RPM vs Million RPM

Exploring the Differences

08

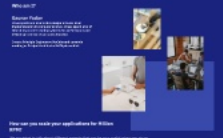
Rate Limiting, retries and circuit breakers will become new normal

10

Do you know your tail?

11

Let's talk about about Database



09

Alerts Metrics and APMs become necessity

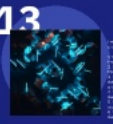
Introduction

Writing an app to run at 10K RPM (requests per minute) isn't too difficult. 10K RPM equals about 170 RPS (Requests per second). A basic MVP (minimum viable product) with key features and good code can achieve this easily.

You will need around 10 core for this all of this very efficiently. [Obviously it depends on task to task, here I am discussing about very basic tasks like login, listing, updates and basic calculations].

12

How to choose right Database?



13

How to choose right Database?

01

Do you really understanding your application

02

Caching

05

Connection Pooling

04

Stay alive we need to do handshake once you started talking: Keep Alive

07

Polling vs Pushing

03

DNS Caching

06

CDN will save your internal systems

Thats it for now.

Stay away from over engineering, if you engineering is not aligned with your business you will be wasting a lot of resource which can actually crete impact.

Everyline of code or architecture change that you do. Always keep in mind how will this imapct my CPU, memory, disk and network.

Questions or Suggestions