

Introduction to Modern Application Development
Dr. Gaurav Raina
Prof. Tanmai Gopal
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 12
Lecture – 21
Scaling a database

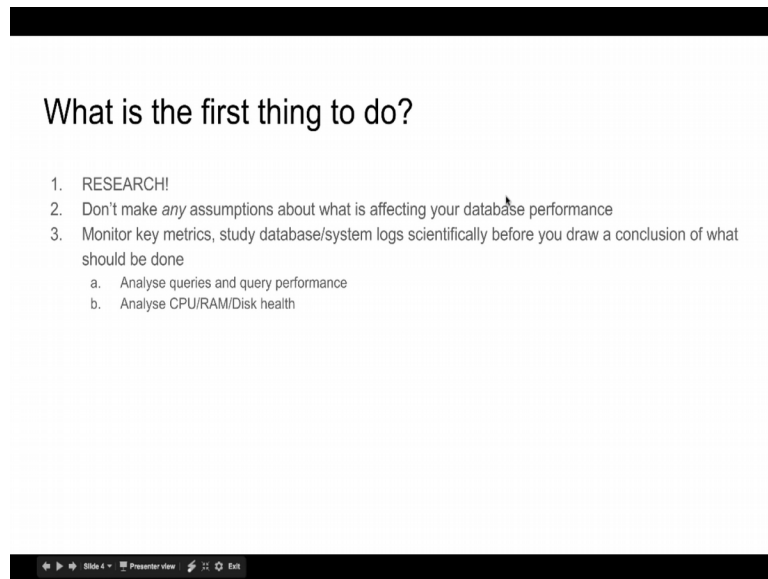
(Refer Slide Time: 00:06)

Why does a database need to scale?

1. To increase the amount of load the database can handle
2. This typically happens when amount of data generated by the activity on the webapp starts growing rapidly
3. We know that we need to scale our database when any of these start happening:
 - a. Performance starts degrading rapidly
 - b. Resources are near maximum use
 - c. Rapid growth in data size, and we can predict that we will run out of resource (especially disk)

Hi all; welcome to module 12, we are going to talk about scaling a database. So, we need to scale a database when we need to increase the amount of load that a database can handle. This usually happens when the amount of data that is being generated by the activity on the web app starts growing rapidly or when the activity on the app requires many more queries to be handled by the database. The typical signs of knowing when we need to scale a database are when performance starts suddenly degrading rapidly. So, whenever additional users come on the performance just suddenly drops we also know that it is time to scale the database when resources are very frequently near maximum use for example, the CPU, RAM or disk is maxing out. We can also predict that we need to scale when the data is growing extremely rapidly and we can see that we would run out of resource very soon.

(Refer Slide Time: 00:50)



What is the first thing to do?

1. RESEARCH!
2. Don't make *any* assumptions about what is affecting your database performance
3. Monitor key metrics, study database/system logs scientifically before you draw a conclusion of what should be done
 - a. Analyse queries and query performance
 - b. Analyse CPU/RAM/Disk health

Slide 4 - Presenter view

So, the first thing to do when we presented with the scaling problem is to do research, we should not make any assumptions about what is affecting our database performance and what the right way to scale is we need to monitor key metrics study the database system logs and do this whole process scientifically before we draw conclusion of which method should be used. So, typically we should be analysing the queries and the query performance and we should be looking at the CPU RAM and disk health of a database server.

(Refer Slide Time: 01:13)

Option 1: Performance tuning

1. The first thing that one can do is to tune the database to perform better so that more load can be handled
 - a. Eg: Increasing the speed with which each query can be performed, will increase the number of queries that can execute each second.
2. As discussed, these are the things that can be done:
 - a. Improve queries (by studying query plans and database logs)
 - b. Add indexes, remove constraints if possible
 - c. Modify the schema to better suit the common queries being made

The first thing to do is to tune a database for increase performance we already discussed this in one of the previous modules. The idea behind increasing the performance is that if you are able to make each query on the database run faster then we can perform a greater number of queries every second. The three main things that we have already discussed it can be done is that number one queries can be improved and we can do this by studying the query plans and the database logs. The second thing is to add indexes or remove constraints or to alter some of the structures in the schema to increase read or write performance of the database.

The third thing to do which is a little which is the hardest do in this list is to modify a schema to better suit the common queries that are being made in the database.

(Refer Slide Time: 01:55)

Option 2: Vertical scaling

1. Improve disk
 - a. Storage capacity
 - b. Increase IOPS (IO operations per second). Eg: SSD
2. Increase RAM
 - a. RAM is used by DBMS to cache heavily
 - b. Eg: Indexes are cached heavily so that we don't have to make an additional disk hit to just fetch the index
3. Increase CPU
 - a. For computationally expensive queries improving CPU processing capability can have a huge positive impact
 - b. Increasing the number of cores also has a positive impact, especially if the DBMS can exploit parallelism
4. Typically:
 - a. Improve RAM and Disk provide immediate benefits
 - b. Impact of improving CPU configuration (faster cores or more cores) depends on the DBMS

Once we realised that it is becoming too hard for us to tune the database for greater performance the next thing to typically do is to vertically scale the database server and by database server here I mean the hardware is the database server setting on. So, the three kinds of things that are done are improving the disk the RAM and the CPU capacity of the particular machine that the database server is on and improving the disk means either increasing storage capacity in case we are running out of disk space or to increase the number of IO operations per second the database can do. For example, going in for an SSD over a hard drive or going in for a better SSD.

The next thing to look at is typically the memory or the RAM DBMS use RAM very effectively because they cache a lot of stuff for example, indexes are cached by the DBMS. So, that the DBMS does not have to make a disk request to fetch the index and then a disk request to fetch the data the index is already in ram. So, the database just has to search through the index and then make a disk request to fetch the data.

Increasing the CPU could mean 2 things could mean making each core faster or good also mean increasing the number of cores that we have both of these have a different impact depending on the DBMS. Typically those from most database systems RAM and disk provide the greatest amount of benefit and the impact of improving the CPU

configuration is heavily dependent on the DBMS.

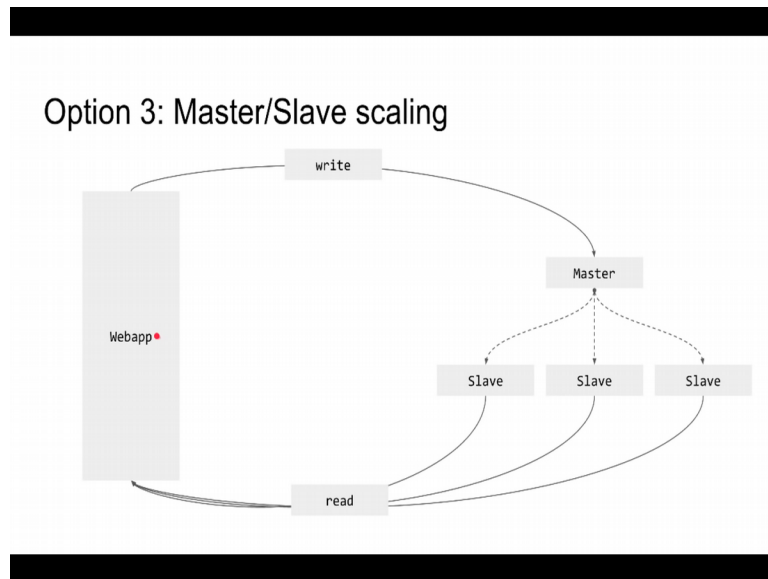
(Refer Slide Time: 03:10)

Option 3: Master/Slave scaling

1. We talked about Master/Slave replication for backup, but more important than its backup use-case, is it's use-case in basic scaling
 - a. Also, relying on master/slave for backup exclusively is dangerous because human-errors made on the master will also happen on the slave
2. Useful when write-traffic is getting heavy and is affecting read performance
3. Typically: write to master, read from slave(s)
 - a. You can also read from master, but the point was to decrease the load on master
 - b. Risk of stale data if master/slave replication is asynchronous
4. Multi-master replication is also available but scary
 - a. Generally results in a loss of transaction/ACID properties
 - b. Conflict resolution can sometimes become intractable if no of nodes/latency is high

The next step to scale the database is using master slave scaling can we talked about master slave replication for backup, but more important than it is back up use case is it is use in scaling also using a master slave for backup as the exclusive method for backup is actually little dangerous because if human errors are made on the master then they will also propagate to the slave. But coming back to scaling we typically look at master slave scaling when the right traffic is getting a little heavy and the right traffic has started to affect the read performance of a database. So, the ways typically setup is that we have a master database and a slave database the slave is getting changes and staying up to date with the master database. So, we write to the master database and then we make all the read queries on the slave databases.

(Refer Slide Time: 03:54)



So, this is what the set up looks like we have a web app on the left and the web app is making a write query to database instance which is a master. Changes from the master are propagated to the slave databases automatically these are copies of the master database and the web app does not need to read queries from the master database, but it makes the read queries from any one of the slave databases. This actually provides us a way to keep scaling the number of read queries that we need. So, if you want to increase the number of queries that we can support for read we can just add more and more slave databases.

(Refer Slide Time: 04:25)

Option 4: Sharding

1. The scariest way of scaling databases, to only be attempted in case the application really needs it
2. Sharding is a type of database partitioning that splits a large dataset into smaller 'shards' across several database instances
3. Typically done only when the writes are too many to handle by a single database instance
4. Sharding can be done manually or automatically
 - a. By splitting data according to functional or usage patterns
 - b. The DBMS itself supports a cluster. Applications typically don't have to change the way they talk to the DBMS
 - c. Very hard distributed systems challenge
 - d. Need to sacrifice on some of the traditional properties to achieve this. Eg: transactions may not work, reads might not reflect updates/writes immediately
5. Sharding is hard because:
 - a. Applications need to change the way they work. Route requests to different shards, or handle transactions in application logic
 - b. Backup/recovery/migration becomes more challenging because data is split across many shards

The last broad methodology for scaling just called Sharding; Sharding is pretty much the scariest way of scaling databases and it should only be attempted in the case of the application really needs it. Sharding is a process of splitting a large data set on a single database into smaller shards across several database instances and this is only done when one particular database instance is not able to handle the entire right traffic.

Sharding or breaking up the data into various shards is a process that can be done manually or automatically. So, we can split the data according to how the data is being used across different instances, we can also use DBMS's that support Sharding out of the box examples of some database systems like this are mongo db. These database software are fairly complicated and sophisticated pieces of core because they solve really hard distributed systems problems, they typically end up sacrificing on some traditional properties that we expect of a database to be able to achieve Sharding in a cluster. For example, transactions may not work across the cluster or reads might not be immediately available after its updates are made to the database. Sharding is hard because of 2 reasons the first reason which is fairly obvious is that applications need to change the way they work for example, applications might have to make request for reading or writing data to different shards or they might have to handle transactions in the application logic because the database cluster cannot handle it anymore.

Sharding is also hard for a more subtle reason and that reason is that backup recovery and migration also become very challenging because the data has not in one database instance anymore, but it split across many shards.

(Refer Slide Time: 06:04)

Option 4: Sharding

1. Sharding data by usage:
 - a. Take tables that are used more frequently and put them on different database instances
2. Sharding data by key:
 - a. Depending on the value of the key of a table the data is written to a different shard

So, the 2 broad ways of Sharding are Sharding data by usage. So, you can take tables that are being used more frequently than other tables and take them out and put them into separate database instances or we can shard by key in which case data inside a table is split across various database instances. So, depending on the value of a key row is put in one database instance or another for example, if you are looking at a user table then we might decide that all the users from 1 to 1 million going to shard one from users 1 million to 2 million going to shard too and so on and so forth.

(Refer Slide Time: 06:40)

Key takeaways

1. Approach database scaling very scientifically
2. Start with the simple stuff first
3. Master/Slave as a scaling mechanism for splitting read/write traffic
4. Sharding is hard, but is ultimately one of the only ways of handling the largest amounts of data

To quickly summarise the concepts that we learnt in this module we understood that should approach database scaling very scientifically we should start with all the simple stuff that we can do first master slave replication is a method for scaling and for splitting read and write traffic it works very well when the right traffic is not extremely high and the read traffic is increasing. Sharding is one of the final ways of scaling and handling the largest amount of data. Sharding also solves the scaling problem to some extent is one of the hardest things to implement.