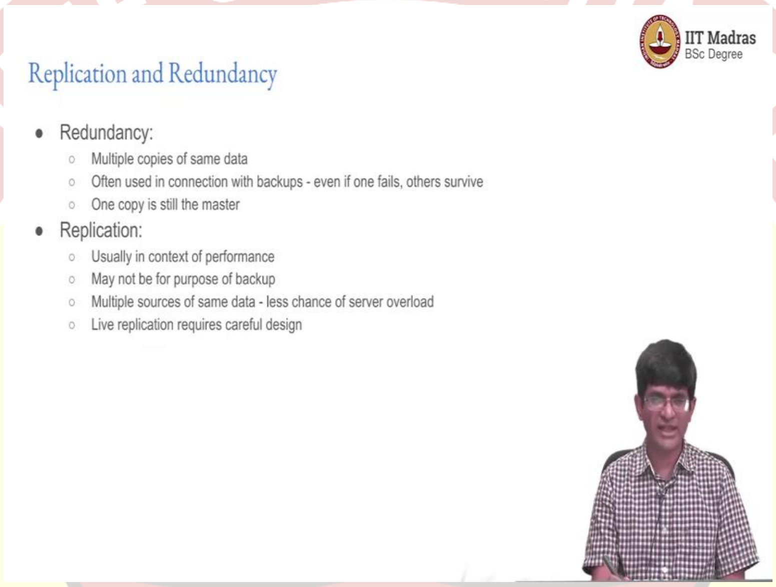
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
सिद्धिर्भवति कर्मजा

IIT Madras
ONLINE DEGREE

Modern Application Development - 1
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Scaling

Hello, everyone and welcome to this course on Modern Application Development and all of that brings us to the topic of Scaling, so what do we mean by scaling in the context of a database.

(Refer Slide Time: 00:25)



Replication and Redundancy

- Redundancy:
  - Multiple copies of same data
  - Often used in connection with backups - even if one fails, others survive
  - One copy is still the master
- Replication:
  - Usually in context of performance
  - May not be for purpose of backup
  - Multiple sources of same data - less chance of server overload
  - Live replication requires careful design

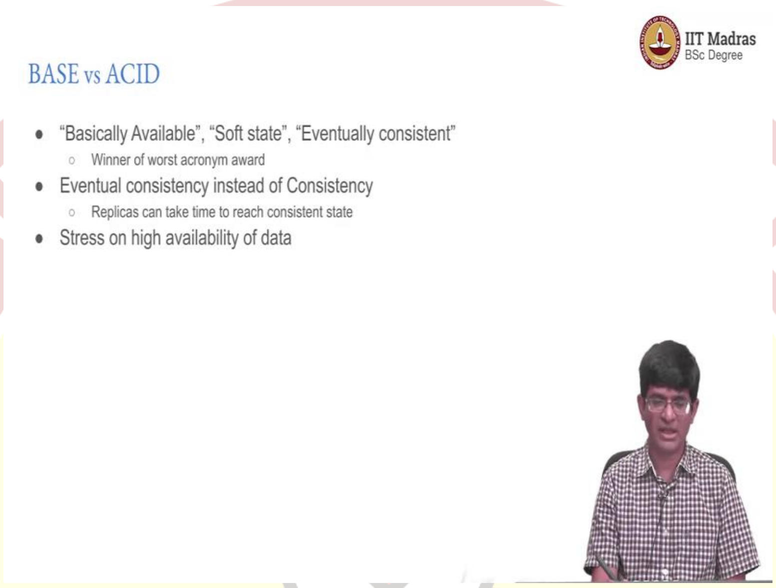
The first thing we need to think about is the notion of replication of data. So, you might want to have multiple copies of data for different reasons. One of them is so called redundancy and redundancy is usually used for things like backups, let us say that there is a chance my laptop fails, I would like to have a copy of all my photos and documents somewhere else, so that even if the laptop fails, I can retrieve it from somewhere else.

Now, I am not going to be accessing that other location all the time. So, the sort of speed at which I access it or how efficient is it to access it is not really the issue. What is important is if one fails, the other should survive.

On the other hand, in the context of things like databases, I might want to do something called replication of data and replication over here is used purely in the context of performance, it is not really for the purpose of a backup. So as an example, I might take two servers, which are sort of identically configured and somehow get the data replicated onto both of them.

What is good about it, any read queries, for example can be answered by either one of them, I do not care and the answer is going to be the same. But they are both located in the same data centre. So, what if the data centre catches fire, they are both lost, So, the redundancy aspect is not really coming into the picture here. Now, live replication of data like this requires very careful design of the database, the data structures and various other things as well.

(Refer Slide Time: 02:02)



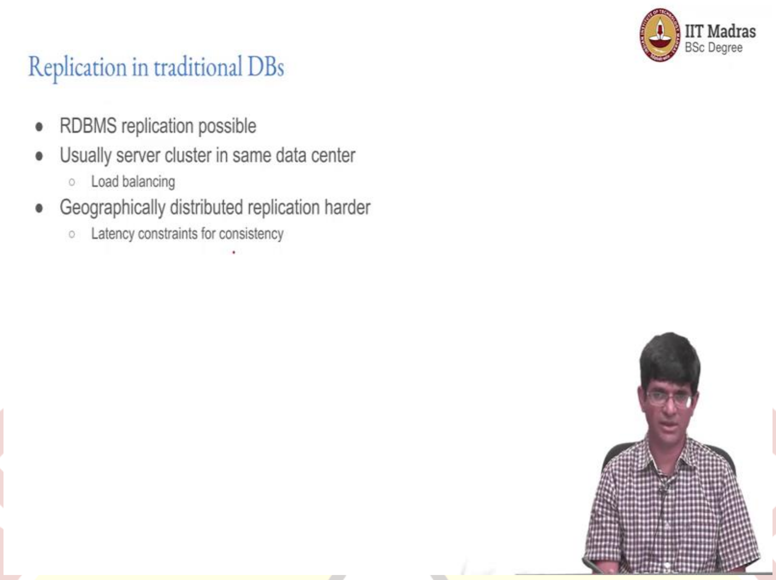
BASE vs ACID

- "Basically Available", "Soft state", "Eventually consistent"
  - Winner of worst acronym award
- Eventual consistency instead of Consistency
  - Replicas can take time to reach consistent state
- Stress on high availability of data

And once again, going back to no SQL that we talked about earlier, no SQL databases often follow something called a BASE philosophy rather than acid. BASE stands for basically available soft state and eventually consistent, classic case of desperately trying to get an acronym just so that it sounds cool.

But anyway, the important part over here is the eventual consistency and what this means is that the replicas can take time to reach a consistent state. More important than consistency is high availability of the data.

(Refer Slide Time: 02:44)

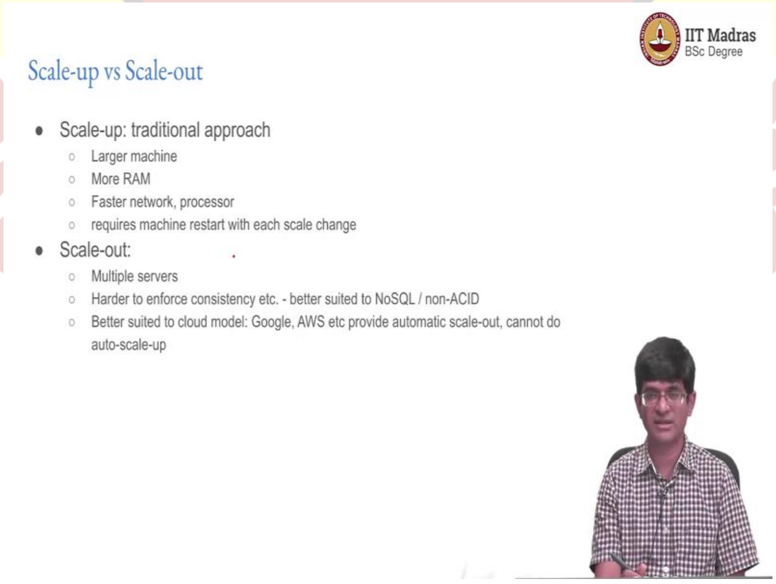


Replication in traditional DBs

- RDBMS replication possible
- Usually server cluster in same data center
  - Load balancing
- Geographically distributed replication harder
  - Latency constraints for consistency

Now, let us look at an to understand this, let us look at replication in a traditional database. RDBMS replication is possible, of course in PostgreSQL, MySQL, Oracle, of course, I mean, has been doing it for decades and usually, the way they do it is have a cluster of servers all in the same data centre, connected with a high speed network and it is used for the purpose of load balancing, replicating across a geographically distributed framework is harder to do real time replication. Because there are latency constraints and they affect how you can guarantee consistency.

(Refer Slide Time: 03:27)



Scale-up vs Scale-out

- Scale-up: traditional approach
  - Larger machine
  - More RAM
  - Faster network, processor
  - requires machine restart with each scale change
- Scale-out:
  - Multiple servers
  - Harder to enforce consistency etc. - better suited to NoSQL / non-ACID
  - Better suited to cloud model: Google, AWS etc provide automatic scale-out, cannot do auto-scale-up

Now, this sort of leads us to this thing, where we have two different notions of how you can scale when your application becomes bigger, the traditional approach is to use something called scale up and this is what Oracle, for example, excels at, you use larger machines with more RAM, faster networks, faster processors.

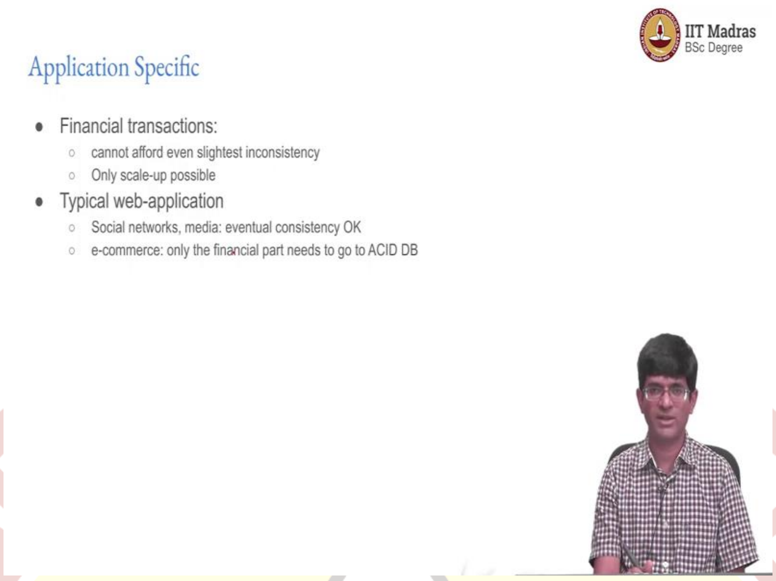
But every time you want to upscale, you want to sort of change the infrastructure of you are running out of memory, you pretty much have to restart the system, some of the servers of course, have been tuned even there. So, for example, they have a mechanism whereby you can hot swap, meaning that you can add more disks while the system is still running, you might even be able to add memory to the system, while it is still running and they even have operating systems, customised operating systems that are able to benefit from that.

A simpler in some sense approach is scaled out. Can I just make more and more copies of the server and just bring them up all over the place? I run out of space on one server fine, add one more machine and somehow let it kind of replicate and follow along with the queries. This is especially well suited to the so-called Cloud model.

Because what Google or AWS then does is in their data centres, they have a large number of machines. They create VMs for your virtual machines, each of which has a particular specification. You have run out of memory on one of them. Now, do not even bother with restarting it just create one more and add it to the pool of servers that you have. So, scale out is very well suited to the modern cloud-based approach.

The problem is, it is not particularly good for acid kind of databases. Because you have that consistency problem to maintain and that is sort of the key to why NoSQL is really popular in these kinds of applications. As long as you can accept this notion of eventual consistency, you can make use of this kind of scale out, which means just adding more servers, which can be done at runtime pretty fast, so Google App Engine or Google Compute Engine can do this automatically for you. They, it means that your application can scale very neatly, even as the number of users suddenly starts increasing.

(Refer Slide Time: 05:51)



Application Specific

- Financial transactions:
  - cannot afford even slightest inconsistency
  - Only scale-up possible
- Typical web-application
  - Social networks, media: eventual consistency OK
  - e-commerce: only the financial part needs to go to ACID DB

Now, of course like I said earlier, this is highly application specific, in particular for financial transactions it is simply not acceptable, they cannot afford even the slightest inconsistency, which is why you will find that, financial applications are probably the last ones to sort of move towards the cloud, they need all kinds of other guarantees and even in the cloud, they will need specific kinds of structures where they own the entire machine on which the information is being processed.

But on the other end, typical web application, social networks, media, eventual consistency is just fine, slight delays are not going to make any real difference. Now, even in ecommerce, interestingly, let us say Amazon or Flipkart, only the financial part needs to be in an acid compliant database, the list of items that are available for a given search could just as well be in some kind of a NoSQL database, scaling out immensely.

Let us say that, unfortunately, you click on something and then it turns out that it is already been sold out, bad luck been just ran out of these items before you clicked on the buy transaction. But once you have committed, you enter your credit card and go and press go. There should be no way that it sort of backs out and says we took your money, but we cannot send you the item. So, there is always some kind of a trade-off that you need to be aware of when you are building an app like this.