Scalable Architecture

Presented By - Anil Kumar Sakala

Agenda

- What is scalability
- Designing & Architecting
- Scaling your server
- Scaling your database
- Scaling UI
- Deployment model changes
- Learning from mistakes

What is scalability

- Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth
- Vertical - Vertical scaling means that you scale by adding more power (CPU, RAM) to your existing machine. Vertical-scaling is often limited to the capacity of a single machine, scaling beyond that capacity often involves downtime and comes with an upper limit.
- Horizontal - Horizontal scaling means that you scale by adding more machines into your pool of resources. With horizontal-scaling it is often easier to scale dynamically by adding more machines into the existing pool.
- Vertical scaling (adding powerful processors and using powerful machines) is expensive when comparing to horizontal scaling. Give preference to horizontal scaling than vertical scaling. Design your application for horizontal scaling.

Designing & Architecting

- Don’t overengineer the solution . If you are expected to come out with a design for simple HR application that is used by 10 people , simple monolithic architecture is fine and there is no point in thinking about scalability , availability and other issues . One way to avoid over engineering is to simplify scope , design and implementation.

- Understand DID principle – According to DID principle design for 20x capacity , implement for 3x capacity and deploy for 1.5x capacity . It is very important to understand scalability needs and choose your frameworks and design methodologies accordingly

- Vertical scaling (adding powerful processors and powerful machines) is expensive when comparing to horizontal scaling . Give preference to horizontal scaling than vertical scaling.

Techniques to scale

- There are two basic principles that need to be followed when you want to scale a software element / component(Database / Server …etc)
  - Reduce number of request that are send to software component that needs to be scaled . Example of filling an application form.
  - Process received requests in less time. Example of processing database queries in less time. This allows more number of requests to be processed in a given time.

Scaling your server

Reduce number of requests	Increase Performance
Design your application so that it can be cloned and distributed on different machines • Stateless • Distributed session • Browser maintained sessions	Avoid network calls (Database calls) as much as possible. Don't put everything on database
Design for pluggable features. Encourage on/off functionalities	Use caching extensively
	Use asynchronous calls
	Encourage queue based architecture

Scaling your database

Reduce number of requests	Increase Performance
Split your database	Use high performing sql queries – Query tuning , Views , Materialized Views , Joins on server side
Master slave architecture when read volume is high	Avoid locks and wait times
Shared database	Avoid select for update or select * from tab;
Use other than database – log files , NO Sql	Avoid multiphase commit

Scaling your UI

Reduce number of requests	Increase Performance
Use cache extensively	Minify Javascript
Avoid DNS lookups . DNS lookups are expensive and can be reduced by reducing number of objects in UI pages . One technique is to merge image objects to single object , this will in return reduce number of calls made to srever	

Deployment model changes

- Design for rollback and replay mechanisms . Failing to design for rollback is failure . Roll backing source code can be easy job but doing a rollback on database is tough job . You need to have proper auditing fields and proper PL/SQL scripts ready for rollback . Replay and rollback mechanism need to be designed as part of your application.
- When designing tables you can use following auditing fields – SYS_SRTN_USER , SYS_CRTN_TIME , LAST_UPTD_USER , LAST_UPTD_TIME , VERSION . Maintain history of changes that are made to a record is also recommended in some cases.
- Design proper deployment mechanisms that will enable to roll back your source code to version of your liking

Learning Environment

- Building scalable applications is very hard job . You will fail couple of times before you come out with a robust scalable architecture . Doing root cause analysis for every failure and performance issues that come up in production will help you in establishing scalable architecture . In summary discuss and learn from failures.

Conclusion

- **References :**
  - Scalability Rules : 50 Principles for Scaling Web Sites
  - The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise

- **Further Reading :**
  - High performance web sites : Essential knowledge for front end engineers
  - The Art of Scalability : Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise
  - Even faster Web Sites : Performance Best Practices For Web Developers
  - Website Optimization