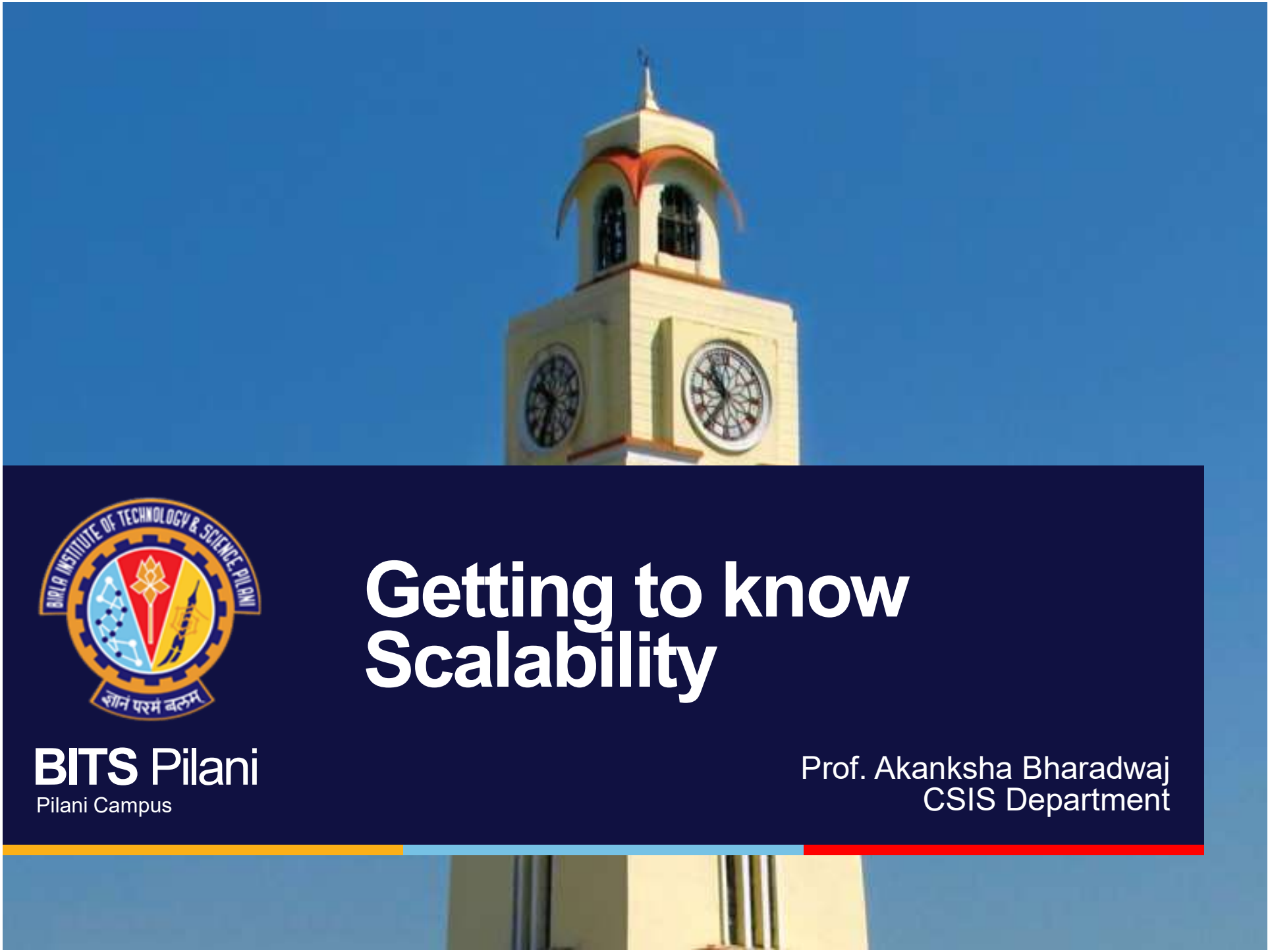

Getting to know Scalability

BITS Pilani
Pilani Campus

Prof. Akanksha Bharadwaj
CSIS Department

innovate achieve lead

BITS Pilani

Pilani Campus

SE ZG583, Scalable Services
Lecture No. 1

innovate achieve lead

BITS Pilani
Pilani Campus

Introduction to Performance, Consistency and availability

CAP Theorem

- The **CAP theorem** states that a distributed system can only guarantee two out of these three characteristics: Consistency, Availability, and Partition Tolerance.

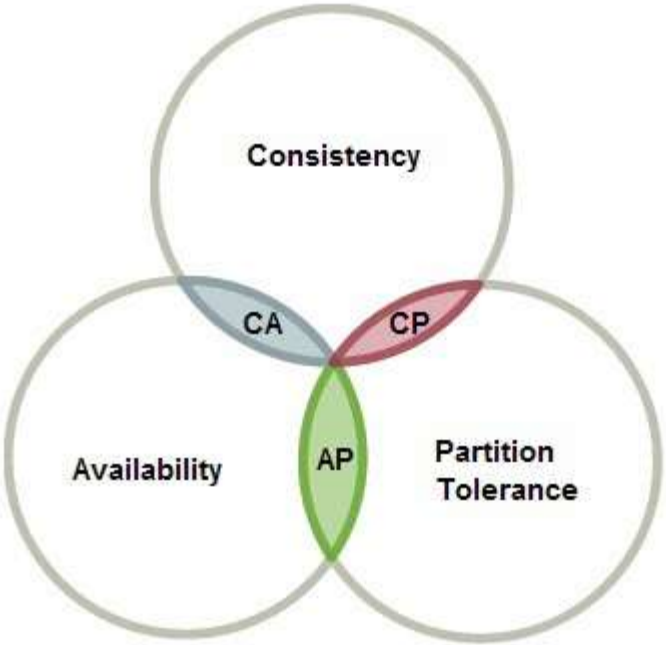

Image: Towardsdatascience

CAP Theorem

- **Consistency**: Consistency means that all clients see the same data at the same time, no matter which node they connect to.
- **Availability**: Availability means that any client making a request for data gets a response, even if one or more nodes are down.
- **Partition tolerance:** A *partition* is a communication break within a distributed system—a lost or temporarily delayed connection between two nodes.

CAP Examples

System Type	Typical CAP Priorities	Example Approaches
Relational Databases	Consistency + Availability	Single-machine or tightly-coupled clusters; limited scalability
Modern NoSQL Databases	Availability + Partition Tolerance	Eventual consistency; trades off strong consistency for scalability (e.g., Cassandra, DynamoDB)
Financial/Transactional	Consistency + Partition Tolerance	May become temporarily unavailable during partitions (e.g., some bank systems)

Eventual Vs Strong Consistency

- Eventual consistency is a consistency model that enables the data store to be highly available.
- Strong Consistency simply means that all the nodes across the world should contain the same value for an entity at any point in time.
- Eventual consistency lets real-world systems like social platforms, online retail, and distributed databases provide a fast and reliable experience on a global scale, even if updates take a moment to appear everywhere.

Performance

- The topmost reason for performance concerns is that the tasks we set our systems to perform have become much more complex over a period of time

- Can be measured using response time, throughput etc.

Availability of a system

- Availability refers to a property of software that it is there and ready to carry out its task when you need it to be.
- Strategies to Increase System Availability
  - Redundancy
  - Failover Mechanisms
  - Load Balancing
  - Regular Maintenance

$$\text{Availability} = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}}$$

What is scalability?

- Scalability of an architecture refers to the fact that it can scale up or down to meet increased or decreased work loads.
- Types of Scalability
  - Vertical Scalability
  - Horizontal Scalability

Need for scalable architectures

Monolithic Architecture

- Monolith means composed all in one piece.
- Traditionally, applications were built on a monolithic architecture, a model in which all modules in an application are interconnected in a single, self-contained unit.
- They’re typically complex applications that encompass several tightly coupled functions.
- When all functionality in a system had to be deployed together, we consider it a **monolith**.

Advantages of Monolith

- Simplicity
- Network latency and security

Disadvantages of Monolith

- Scalability
- Slow development
- Long deployment cycle

Principles of Scalability

1. Scale horizontally, not vertically
2. Statelessness
3. Cache
4. Load Balancing
5. Modular Architecture
6. Asynchronous Processing
7. Monitoring and Metrics
8. Resilience and Fault Tolerance

All these mainly target three areas **Availability, Performance, and Reliability**

Guidelines for Building Highly Scalable Systems

1. Avoid shared resources as they might become a bottleneck
2. Avoid slow services
3. Optimize Database Design
4. Auto-Scaling
5. Prioritize API Design
6. Ensure Data Consistency and Partitioning
7. Optimize Network Usage
8. Minimize Latency
9. Plan for Growth

Architecture's scalability requirements

- How important are the scalability requirements?
- Identify the scalability requirements early in the software life cycle so that that it allows the architectural framework to become sound enough as the development proceeds.
- System scalability criteria could include the ability to accommodate
  - Increasing number of users,
  - Increasing number of transactions per millisecond,
  - Increase in the amount of data

Challenges for Scalability

Here are some common **challenges for scalability** in real-world applications:

1. Unanticipated Traffic Spikes
2. Database Scalability
3. Resource Contention
4. Distributed System Complexities
5. Legacy Systems
6. Cost of Scalability
7. Security at Scale
8. Scalability Testing

innovate achieve lead

BITS Pilani

Pilani Campus

Case Study

High-Level Design (HLD) of YouTube

- The high-level design of YouTube is a **distributed, large-scale architecture** that supports several billion users, millions of video uploads, and hundreds of millions of searches per day.

- YouTube deals with challenges of scale, real-time video streaming, data processing, and distributed search.

Core High-Level Components

Content Delivery Network (CDN)

- CDN nodes (Edge Servers) cache videos based on user proximity and demand.
- **YouTube uses Google's CDN**, part of Google Cloud Platform (GCP).

Core High-Level Components

Video Upload and Processing

- Users upload video data in chunks (multi-part upload), using the **Google Cloud Storage** API.
- YouTube uses **FFmpeg** (a widely used multimedia processing framework) internally to transcode videos into multiple resolutions.

Core High-Level Components

Storage (Video and Metadata Storage)

- YouTube stores video data in a **distributed object storage system** using **Google Cloud Storage** (GCS)
- Metadata (video titles, descriptions, tags) is stored in **Bigtable**, a NoSQL database developed by Google.

Core High-Level Components

Content Search

- YouTube relies on **Elasticsearch**
- It's distributed, supports multi-node clusters, and is designed to handle real-time, large-scale search operations.

Core High-Level Components

Recommendation System

- It uses **machine learning models** (like collaborative filtering, deep learning, and matrix factorization techniques) trained on user data: watch history, likes, search behavior, and demographics.

Core High-Level Components

API Gateway

- **Google's API Gateway** handles routing, authentication, and rate-limiting. It connects clients to backend services while ensuring that the system remains **modular** and **scalable**.

Self Study

Example of scalable database architecture:

https://www.youtube.com/watch?v=VHELcOe1gy0

References

- Chapter-1, Monolith to Microservices, Sam Newman
- Chapter-1, Building Microservices, Sam Newman
- https://groups.csail.mit.edu/tds/papers/Gilbert/Brewer2.pdf
- http://highscalability.com/youtube-architecture
- https://dev.to/wittedtech-by-harshit/system-design-of-youtube-a-detailed-deep-dive-into-the-video-giant-5019?