COMSATS University
Islamabad
Abbottabad Campus

Assignment #2

Software Design and Architecture

Submitted By:

- Noman Shakir (FA22 - BSE - 115)

Submitted To: Sir Mukhtiar Zamin

Date: 03-01-2025

NAME: NOMAN SHAKIR
REG: FA22-BSE-115

Identifying 5 Major Architectural Problems

1. Twitter (Scalability Issues)

- ### Problem:

Twitter struggled to handle increased traffic, especially during major events like elections or global sports tournaments, leading to frequent downtimes or delays in posting and retrieving tweets.

Causes:

- **Monolithic Architecture**: All components of the application (tweet posting, user management, timelines) were tightly coupled, creating bottlenecks under heavy load.
- **Lack of Horizontal Scalability**: The monolithic system couldn't efficiently distribute load across multiple servers.
- **Single Points of Failure**: Failure in one module (e.g., timeline service) cascaded into failures across the entire system.

Solution:

- **Microservices Transition**: Twitter split its monolithic system into smaller, independently deployable services (e.g., Tweet Service, Timeline Service, Notification Service).
- **Distributed Data Storage**: Implemented distributed databases like Apache Storm to handle real-time streaming data for timelines.
- **Caching**: Used caching mechanisms (e.g., Redis) to reduce database load during peak traffic.

Prevention:

- **Scalable Design**: Adopt modular and loosely coupled architectures.
- **Load Testing**: Simulate high-traffic conditions to identify bottlenecks.
- **Monitoring and Redundancy**: Use tools like Prometheus for monitoring and ensure failover mechanisms are in place.

---

2. Netflix (Latency Issues)

Problem:

Netflix faced delays in delivering video content globally, resulting in poor user experience, especially in regions far from their centralized data centers.

Causes:

- **Centralized Data Centers**: Streaming data from a single location caused high latency for geographically distant users.
- **Network Congestion**: Increased demand for high-quality video streaming added strain on network infrastructure.

Solution:

- **Content Delivery Network (CDN)**: Deployed a globally distributed CDN (Open Connect) that stores video content closer to users, significantly reducing latency.
- **Adaptive Streaming**: Introduced dynamic bitrate adjustments to provide uninterrupted streaming even under constrained bandwidth.

Prevention:

- **Distributed Architecture**: Build systems with edge computing to bring services closer to end-users.
- **Regional Optimization**: Monitor regional demand and optimize resource allocation dynamically.

---

3. Facebook (Data Consistency Problems)

- ## Problem:

Facebook experienced issues with maintaining data consistency in its massive social graph, leading to delays in syncing friend requests, messages, and notifications.

Causes:

- **Overloaded Relational Databases**: Relational databases couldn't handle the growing volume and complexity of data.
- **Inefficient Synchronization**: Slow synchronization across distributed databases led to inconsistencies.

Solution:

- **Graph-Based Databases**: Migrated to NoSQL databases like Cassandra and HBase for scalable and efficient handling of graph data.
- **Eventual Consistency Model**: Implemented an eventual consistency model for non-critical data while maintaining strict consistency for critical operations.

Prevention:

- **Database Choice**: Use a database system that aligns with data structure and scale requirements.
- **Partitioning and Sharding**: Distribute data intelligently to avoid overloading specific nodes.

---

4. Amazon (Downtime Issues)

- ## Problem:

Amazon encountered significant downtime during high-traffic events like Black Friday and holiday sales, resulting in lost revenue and customer trust.

Causes:

- **Monolithic Architecture**: A single application managing all services became overwhelmed during traffic spikes.
- **Resource Contention**: Simultaneous requests overloaded shared resources, causing failures.

Solution:

- **Service-Oriented Architecture (SOA)**: Decoupled services like Payment, Inventory, User Management, and Order Management into independent components.
- **Load Balancing and Auto-Scaling**: Introduced Elastic Load Balancing (ELB) and auto-scaling to dynamically adjust resources based on traffic.

Prevention:

- **Stress Testing**: Conduct periodic stress tests to prepare for high traffic.
- **Redundancy**: Deploy multiple instances of critical services for failover support.

- **Monitoring**: Use real-time monitoring tools like AWS CloudWatch to detect anomalies early.

---

5. Microsoft Azure (Security Breaches)

- ## Problem:

Microsoft Azure faced security vulnerabilities in its multi-tenant cloud environments, exposing sensitive user data to breaches.

Causes:

- **Weak Tenant Isolation**: Poorly designed mechanisms to isolate tenants allowed unauthorized access.
- **Lack of Encryption**: Insufficient encryption protocols made data vulnerable during transit and at rest.

Solution:

- **Zero-Trust Architecture**: Adopted a zero-trust model requiring strict identity verification for every access request.
- **Enhanced Encryption**: Introduced end-to-end encryption and hardware-level security measures like Azure Secure Boot.

Prevention:

- **Security Audits**: Conduct regular audits to identify and fix vulnerabilities.
- **Proactive Threat Detection**: Use AI-driven tools to monitor and detect threats in real time.
- **Data Segregation**: Ensure clear boundaries between tenants using advanced isolation techniques.

NAME: NOMAN SHAKIR
REG: FA22-BSE-115

Problem: Amazon's Downtime Issues During Peak Traffic

Description of the Problem

Amazon faced significant downtime issues, particularly during high-traffic periods such as the holiday season or major sales events. This downtime directly impacted customer satisfaction and resulted in revenue loss.

Causes

1. **Monolithic Architecture**: The system was built as a single large application. Any failure in one part of the system caused the entire system to go down.
2. **Scalability Challenges**: Limited ability to handle a sudden surge in traffic due to centralized processing and lack of horizontal scalability.
3. **Single Points of Failure**: If a critical component failed, it affected the entire system.
4. **Code Base Complexity**: The tightly coupled codebase made it difficult to implement updates or fixes without impacting the entire application.

Solution

Breaking Down the Monolith

- **Microservices**: Separate services for user, order, and inventory management.
- **Independent Services**: Each service operates independently with its own logic and database.

Dynamic Service Discovery

- **Consul**: Allows dynamic registration of services.
- **API Gateway**: Routes client requests to the appropriate microservices via the discovery server.

Routing Requests

- The **API Gateway** ensures seamless routing and returns "Service Unavailable" if a service is down, maintaining system integrity.

Fault Tolerance

- Service failures do not impact other services, ensuring operational continuity.
- Faulty services are isolated without crashing the entire system.

Scalability

- High-demand services can scale horizontally (e.g., running multiple instances of inventory-service during peak sales).

Futureproofing

- New features can be added as independent services without affecting existing ones.

Prevention of Downtime

1. **Load Testing**: Simulate peak traffic to ensure scalability.
2. **Monitoring**: Use tools like Prometheus for health checks and alerts.
3. **Redundancy**: Run multiple service instances for failover.
4. **Zero-Downtime Updates**: Use rolling or blue-green deployments.
5. **Distributed Databases**: Handle traffic efficiently with databases like MongoDB or DynamoDB.