ITMO University

Highload High-Availability Systems

Course Workload		Assessment form (examination/ graded test/ ungraded test)
ECTS	Hours	
3	108	Ungraded test

Discipline focuses on the principles of designing, developing, and maintaining software systems capable of handling massive data volumes and extreme user loads. The course covers key aspects: architectural patterns (microservices, CQRS, Event Sourcing), scaling strategies (horizontal and vertical), load balancing techniques, data storage and caching approaches, ensuring fault tolerance, performance monitoring, and security fundamentals. The practical part of the course is aimed at applying the acquired knowledge to create and deploy a real-world prototype of a high-load application.

Course structure:

1. Introduction to High-Load/High-Availability: Goals, Metrics, and Key Principles

1.1. Microservices: Decomposition by business capabilities, independent deployment/scaling, communication, complexity.
1.2. High load and High availability. Business consequences of downtime and slow performance. The cost of availability. Key goals of HL/HA systems.
1.3. Integration patterns: API Gateway, Backend for Frontend (BFF).
1.4. Fundamental principles: Design for Failure. Loose Coupling. Separation of Concerns.

2. Query Processing: Load Balancing, Scaling, and Architectural Styles

2.1. Load Balancing: Algorithms, Hardware, and Software Solutions (L4/L7, Nginx, HAProxy).
2.2. Caching Strategies. Distributed Caches (Redis, Memcached).
2.3. Designing Data Storage Tiers: Sharding, Replication, CAP Theorem.

3. State Management, Caching Strategies

3.1. Cache functions. Caching levels: Client. Proxy/CDN. Application server. Database. Caching product data in a marketplace. Cache Invalidation. Distributed cashes.

3.2. Database Scaling Issues: Vertical Scaling Limitations, CAP Theorem, PACELC. Replication: Master-Slave, Multi-Master. Synchronous vs. Asynchronous Replication.

4. Data under Load: Scaling Storage (OLTP)

4.1. Sharding/Partitioning: Strategies: Range, Hash, List, Geo. Key Issues.
4.2. Choosing a DBMS: Relational vs. NoSQL. Polyglot Persistence.
4.3. Problems of synchronous communications. Message Brokers. Purpose: buffering, asynchrony, decomposition, fault tolerance. Models: queues, topics. Delivery guarantees.
4.4. Stream Processing. Purpose: real-time data processing, event response. Concepts: topics, partitions, consumer groups, offsets.
4.5. Platforms: Apache Kafka, Amazon Kinesis, Pulsar. Patterns, benchmarks and challenges.

5. Asynchrony and event-driven architecture

5.1. Problems with synchronous communications: Cascading failures, latency amplification, difficulty scaling. Message brokers (Message Queues): Purpose: buffering, asynchrony, decomposition, fault tolerance.
5.2. Stream Processing: Purpose: real-time data processing, event response. Concepts: topics, partitions, consumer groups, offsets.
5.3. Platforms: Apache Kafka, Amazon Kinesis, Pulsar. Example: "Product View" and "Add to Cart" event stream for real-time recommendations and analytics.
5.4. Benchmarks and challenges: Latency, message ordering, duplicates, queue monitoring.

6. Data for Analysis: Scaling Warehousing (OLAP) and ETL/ELT

6.1. Differences between OLTP and OLAP: Requirements (ACID vs. aggregation speed), data structure (Normalized vs. Denormalized, Star/Snowflake Schema). Data Warehouse (DWH): Centralized storage, ETL/ELT processes.
6.2. Data Lakes: Storing raw data (structured, semi-structured, and unstructured) in object storage (S3, ADLS, GCS). Metadata (Hive Metastore, AWS Glue).
6.3. Big Data Processing: Computing Models (Batch - Hadoop/Spark, Stream - Spark Streaming/Flink/Kafka Streams, Interactive - Presto/Trino). Example: Daily ETL from OLTP to DWH; Streaming Event Processing for Fraud.
6.4. Integration with OLTP: Change Data Capture (CDC) for streaming changes to DWH/Lake. Example: Debezium for capturing changes from the order database.

7. Reliability in Detail: Observability, Testing, and Disaster Recovery

7.1. Observability vs. Monitoring: Logs (structured logging, aggregation - ELK, Loki), Metrics (Prometheus, StatsD, metric types), Tracing (OpenTelemetry, Jaeger, Zipkin). The "three pillars" and their relationship.
7.2. Load Testing: Tools (k6, JMeter, Locust), test planning (load profile), results analysis (bottlenecks).
7.3. Stress Testing / Chaos Engineering: Tools (Chaos Monkey, Gremlin), principles (introduce chaos gradually, have a rollback plan), Game Days practice.

7.4. Disaster Recovery (DR): Strategies: Backup & Restore, Pilot Light, Warm Standby, Multi-site Active/Active (or Hot Standby).
7.5. RPO (Recovery Point Objective) and RTO (Recovery Time Objective). Regular testing of the DR plan. Postmortems: Blameless Culture, focusing on systemic problems rather than people, identifying root causes (Root Cause Analysis), developing corrective actions.

8. Operational aspects and capacity planning

8.1. Infrastructure as Code (IaC): Configuration management (Ansible, SaltStack), provisioning (Terraform, CloudFormation), benefits (reproducibility, versioning, documentation).
8.2. CI/CD for HL/HA: Automated build, testing (unit, integration, e2e, load), and deployment. Deployment strategies (Blue/Green, Canary, Rolling). Feature flags.
8.3. Containerization and orchestration (Docker, Kubernetes): Infrastructure abstraction, lifecycle management, scaling, and self-healing. Service meshes (Istio, Linkerd) for traffic management and observability.
8.4. Capacity Planning: Load forecasting (trends, marketing promotions, seasonality). Resource usage monitoring (CPU, Memory, Network, Disk IOPS/Throughput). Model building. Budgeting.
8.5. Soft skills for an architect: Communication with the business (explaining technical solutions and costs), working with a team, managing technical debt, making architectural decisions (trade-offs).