Scalability and Performance in System Design

---

1. What is Scalability?

Scalability is a system’s ability to handle **increased load** without compromising performance.

- **Vertical Scaling:** Add more resources (CPU, RAM) to a single machine.
  E.g., upgrading from a 4-core to 16-core server.
- **Horizontal Scaling:** Add more machines to distribute the load.
  E.g., adding more application servers behind a load balancer.

---

2. What is Performance?

Performance refers to **how fast a system responds** to a given request.

- **Latency:** Time taken to process a single request.
  Lower is better.
- **Throughput:** Number of requests processed per second.
  Higher is better.

---

3. Key Metrics to Monitor

Metric	Description
Latency	Time to complete a request
Throughput	Requests per second
Load	Number of concurrent users
CPU/Memory Usage	Resource consumption
Error Rate	Number of failed requests
P99 Latency	99th percentile response time

---

4. Techniques to Improve Performance

Caching

- Use Redis or Memcached to store frequently accessed data.
- Reduces database load and improves response time.

Database Optimization

- Use proper indexing.
- Normalize or denormalize based on access patterns.
- Optimize queries and schema design.

Concurrency & Parallelism

- Use multi-threading or async operations to handle more requests concurrently.

Load Balancing

- Distributes incoming traffic across multiple servers.
- Common tools: NGINX, HAProxy, AWS ELB.

Queueing Systems

- Use message queues (e.g., Kafka, RabbitMQ) to handle asynchronous workloads.

---

5. Patterns for Scalability

Replication

- Duplicate data across servers for reliability and faster access.

Sharding

- Split large databases into smaller parts to spread the load.

CDN (Content Delivery Network)

- Deliver static content from servers close to users (e.g., Cloudflare, Akamai).

Microservices

- Break system into smaller services to scale independently.

---

6. Scalability Trade-offs

- **Consistency vs Availability (CAP Theorem)**
- **Latency vs Accuracy (e.g., eventual consistency)**
- **Read vs Write Optimization**

---

7. Tools & Technologies

Purpose	Tools
Caching	Redis, Memcached
Load Balancing	NGINX, HAProxy, AWS ELB
Queues	Kafka, RabbitMQ, SQS
Monitoring	Prometheus, Grafana, Datadog
Profiling	NewRelic, AppDynamics

---

8. Example Scenario: Scaling a Web App

Problem:

Your app slows down when active users exceed 100k.

Solution Strategy:

1. Use CDN for static assets
2. Add Redis for caching user sessions
3. Optimize DB queries and indexes
4. Add more app servers behind a load balancer
5. Move reporting jobs to background workers