Interview Questions with Detailed Answers – System Performance

---

1. What is the difference between latency and throughput?
Answer:

- **Latency** is the time it takes for a system to respond to a request (e.g., response time in milliseconds).
- **Throughput** is the number of requests the system can handle in a given time (e.g., requests per second).
- Analogy: If a highway is a system, **latency** is how fast one car can travel, while **throughput** is how many cars can pass per hour.
- These metrics often trade off — optimizing one can hurt the other.

---

2. How do SLAs, SLOs, and SLIs differ? Provide real-world examples.
Answer:

- **SLA (Service Level Agreement)**: A contractual commitment (e.g., 99.9% uptime for a paid service).
- **SLO (Service Level Objective)**: Internal performance target (e.g., 95% of API requests respond < 200ms).
- **SLI (Service Level Indicator)**: The actual measured metric (e.g., current API latency = 92% < 200ms).
- Example: An e-commerce platform might offer an SLA of 99.9% uptime, have an SLO of 99.95%, and monitor uptime via SLI metrics from uptime checks.

---

3. Why are percentiles (like P95, P99) important in performance monitoring?
Answer:

- Averages can hide outliers — 90% of requests might be fast, but 10% could be very slow.
- **P95**: 95% of requests are faster than this threshold.

- **P99**: Captures the tail latency — slowest 1% of requests that often impact user experience.
- Monitoring percentiles helps surface **real-world slowness** that affects users and drives better performance tuning.

---

4. What strategies would you use to identify a system's performance bottleneck?
Answer:

- Use **profiling** and **monitoring tools** (e.g., New Relic, Grafana, Datadog)
- Break down the request path: DB calls, service-to-service latency, cache hits/misses
- Look at resource metrics: CPU, memory, disk I/O, network latency
- Perform **load testing** to simulate pressure and expose limits
- Analyze logs and distributed traces to locate slow operations

---

5. How would you ensure responsiveness in a highly scalable system?
Answer:

- Use **asynchronous processing** for non-critical paths (e.g., background jobs)
- Implement **caching layers** (e.g., Redis, CDN) to reduce backend load
- Apply **rate limiting** and **load shedding** to maintain system health
- Ensure horizontal scalability of components (e.g., stateless services)
- Monitor **tail latencies** and auto-scale based on traffic

---

6. What tools or techniques have you used for performance testing and monitoring?
Answer:

- **Testing**: JMeter, k6, Locust for load/stress testing
- **Monitoring**: Prometheus + Grafana, New Relic, Datadog, AWS CloudWatch

- Use **APM tools** to trace requests and identify slow spans
- Use **synthetic monitoring** for uptime, and **real user monitoring (RUM)** for client-side performance
- Set alerts on SLO breaches, CPU/memory spikes, or error rates

---

7. How would you design a system to handle sudden traffic spikes?
Answer:

- Use **autoscaling** (e.g., AWS Auto Scaling Groups, Kubernetes HPA)
- **CDNs** to serve static content and reduce origin server load
- Implement **queueing systems** (e.g., Kafka, SQS) for smoothing bursts
- Apply **circuit breakers** to prevent cascading failures
- Keep services **stateless** so they can scale out quickly

---

8. Explain the trade-offs between performance and cost in cloud environments.
Answer:

- Faster performance often requires **more resources**, which means higher cost (e.g., provisioned IOPS, larger instances)
- Trade-off between **reserved vs. on-demand capacity**
- **Caching** and **batching** improve performance without linearly increasing cost
- **Auto-scaling** and **serverless** can optimize cost-per-request, but might introduce cold start latencies
- Cost-performance decisions should align with business SLAs/SLOs and traffic patterns