50+ System Design Interview Questions and Answers

(Focus: Load Balancing + Scalable File/Image Server)

🔹 Section 1 — Fundamentals of Load Balancing

---

1. What is Load Balancing?

Answer:
Load balancing is the process of distributing incoming network traffic across multiple backend servers to ensure no single server becomes a bottleneck.
It improves performance, reliability, and scalability of distributed systems.
Example: Nginx distributing requests across multiple app servers.

---

2. Why is Load Balancing needed?

Answer:
To handle increasing traffic, prevent downtime, and optimize resource utilization. It ensures:

- High availability (one server down → others take over)
- Fault tolerance
- Better throughput and response times

---

3. What are the types of Load Balancing?

Answer:

1. **Hardware Load Balancer** – Proprietary devices (e.g., F5, Citrix ADC)
2. **Software Load Balancer** – Open-source or cloud-based (e.g., Nginx, HAProxy, AWS ELB)
3. **DNS-based Load Balancing** – DNS resolves to multiple IPs

---

4. What's the difference between horizontal and vertical scaling?

Answer:

- **Vertical scaling** → Add resources (CPU/RAM) to one server.
- **Horizontal scaling** → Add more servers and distribute load.
  Horizontal scaling is more fault-tolerant but complex; vertical is simple but limited by hardware.

---

5. What is a Reverse Proxy?

Answer:
A reverse proxy sits in front of backend servers and routes client requests to them. It hides the internal architecture and can provide caching, SSL termination, and load balancing.
Example: Nginx, HAProxy.

---

6. What are common load balancing algorithms?

Answer:

- **Round Robin:** Evenly distributes requests sequentially.
- **Least Connections:** Sends requests to the server with the fewest active connections.
- **IP Hash:** Routes based on client IP (ensures session stickiness).
- **Weighted Round Robin:** Prioritizes servers with higher capacity.
- **Consistent Hashing:** Used for distributed caching (minimizes key remapping).

---

7. What's the difference between Layer 4 and Layer 7 Load Balancing?

Answer:

- **Layer 4 (Transport):** Routes based on IP and TCP/UDP ports (fast, less flexible).
- **Layer 7 (Application):** Routes based on HTTP headers, cookies, or content.
  Example: Route `/api` to backend A and `/images` to backend B.

---

8. What are sticky sessions (session affinity)?

Answer:
Sticky sessions ensure all requests from a user go to the same backend server—useful when session data is stored locally.
Implemented using cookies or IP hashing.

---

9. How does a load balancer detect server failure?

Answer:
Through **health checks** (ping, TCP connection, or HTTP response).
If a server fails, the LB removes it from rotation until it recovers.

---

10. What happens if the Load Balancer itself fails?

Answer:
It becomes a single point of failure. To avoid this:

- Deploy multiple LBs in **active-passive** or **active-active** mode.
- Use **DNS round-robin** or **floating IP** failover mechanisms.

🔹 Section 2 — Load Balancing Implementation

---

11. What is HAProxy and how does it differ from Nginx?

Answer:

- **HAProxy** specializes in high-performance TCP/HTTP load balancing.
- **Nginx** can act as a web server, reverse proxy, and load balancer.
  HAProxy supports dynamic server health checks better; Nginx is more lightweight.

---

12. How do you configure Nginx for load balancing?

Answer:

```
upstream backend {
  server app1.example.com;
  server app2.example.com;
}
```

```
server {
  listen 80;
  location / {
    proxy_pass http://backend;
  }
}
```

This configuration uses **round robin** by default.

---

13. What is Weighted Load Balancing?

Answer:
Assigns weight to each backend server. Example:

```
upstream backend {
  server app1.example.com weight=3;
  server app2.example.com weight=1;
}
```

App1 gets 3x more requests than app2.

---

14. How do CDNs complement load balancers?

Answer:
A CDN caches static content (images, CSS, JS) closer to users, reducing traffic to the load balancer and backend servers.
LB handles dynamic requests; CDN handles static.

---

15. What are failover strategies in Load Balancing?

Answer:

- **Active-Passive:** Secondary LB activates on failure.
- **Active-Active:** All LBs share load; if one fails, others absorb traffic.
- **DNS Failover:** Redirects traffic to healthy instances.

---

16. What metrics should be monitored for a load balancer?

Answer:

- Request per second
- Connection errors
- Response time
- CPU/memory usage
- Server health status

---

17. How do you handle SSL termination in a load balancer?

Answer:
Terminate SSL at the LB → decrypt traffic → forward plain HTTP to backend.
Reduces backend load and centralizes certificate management.

---

18. What's the role of consistent hashing in scaling systems?

Answer:
Ensures minimal data movement when servers are added/removed.
Used in distributed caching systems (e.g., Redis cluster, CDN routing).

---

19. How do you design a load balancer for millions of requests per second?

Answer:

- Use **multi-level LBs** (DNS → Regional → App-level).
- Employ **CDN caching** for static files.
- Optimize **network throughput** using anycast IPs.
- Scale horizontally with health checks and failover.

---

20. What are common bottlenecks in load balancing setups?

Answer:

- Improper scaling of backend servers
- Session stickiness creating uneven load
- Single LB instance (SPOF)
- Poorly tuned connection timeouts

🔹 Section 3 — Scalable File/Image Server Design

---

21. What are the core components of a scalable file/image system?

Answer:

1. **Frontend/API Gateway**
2. **Load Balancer**
3. **Application Servers**
4. **Object Storage (S3/GFS)**
5. **Metadata DB**
6. **CDN for delivery**

---

22. What challenges arise in file upload systems?

Answer:

- Large file uploads
- Network interruptions
- Versioning & deduplication
- Metadata consistency
- Cost optimization (storage tiers)

---

23. How to design file upload flow efficiently?

Answer:

1. Client requests upload URL.
2. LB directs to upload service.
3. File stored in object storage.
4. Metadata (user, size, type) saved in DB.
5. CDN caches for fast retrieval.

---

24. Why use Object Storage instead of File System?

Answer:
Object storage (e.g., S3, MinIO) provides:

- Infinite horizontal scalability
- Metadata per object
- High availability & replication

---

25. How to reduce latency in file delivery?

Answer:

- Use **CDN edge caching**
- Compress images (WebP)
- Enable **HTTP/2** and **keep-alive**
- Store frequently accessed files closer to users

---

26. How does CDN handle cache invalidation?

Answer:
Through:

- **Time-based expiry (TTL)**
- **Purge APIs** for manual invalidation
- **Versioned URLs** (e.g., `image_v2.jpg`)

---

27. How do you handle file consistency in multi-region storage?

Answer:
Use **eventual consistency** with replication logs.
Synchronize metadata through message queues (Kafka).
Employ read/write regions for proximity.

---

28. How do you secure file uploads?

Answer:

- Validate MIME types and size limits
- Virus scanning before storage
- Use pre-signed URLs (AWS S3 style)
- Encrypt at rest and in transit

---

29. What is multipart upload and why use it?

Answer:
Splitting large files into smaller parts for parallel upload.
Improves reliability and allows resume-on-failure.
Example: AWS S3 Multipart Upload API.

---

30. How do you scale a file server?

Answer:

- Use **stateless upload services** behind LBs
- **Store metadata in DB**, files in object storage

- **Cache metadata** in Redis
- **Replicate storage** across regions

🔹 Section 4 — Advanced & Scenario-based

---

31. How would you ensure data durability?

Answer:

- Replicate files across multiple storage nodes
- Use erasure coding
- Employ versioning for recovery

---

32. How do you handle hot objects (popular files)?

Answer:

- Cache them at edge/CDN
- Replicate in-memory
- Use consistent hashing to avoid overloading one node

---

33. How do you measure load balancing efficiency?

Answer:
Monitor:

- Request distribution uniformity
- Latency per backend
- Server utilization variance

---

34. How would you test a load balancer setup?

Answer:

- **Load testing tools:** JMeter, k6, Locust
- Monitor metrics in Grafana
- Simulate backend failures to test failover

---

35. Explain the CAP theorem in the context of load balancing.

Answer:
Load balancers prefer **availability** and **partition tolerance**, sometimes trading off **consistency** in session data (handled externally).

---

36. What is Global Load Balancing?

Answer:
Distributes traffic across multiple regions/data centers using DNS-based routing (e.g., AWS Route53, GSLB).

---

37. What are common CDN providers?

Answer:

- Cloudflare
- Akamai
- AWS CloudFront
- Fastly
- Google Cloud CDN

---

38. How do you handle image transformations (resize/compress)?

Answer:
Use **worker microservices** or **Lambda functions** triggered post-upload to process and cache resized versions.

---

39. How would you handle downtime in one region?

Answer:

- Failover routing via DNS or GSLB
- Replicate storage
- Serve stale cache from CDN

---

40. What's the difference between caching and CDN?

Answer:
Caching: server-side or in-memory (Redis).
CDN: globally distributed caching at the network edge.

---

41. What is autoscaling and how does it integrate with load balancing?

Answer:
Autoscaling adjusts backend instances based on load.
LB dynamically updates target groups to include new instances (e.g., AWS ALB + EC2 Autoscaling Group).

---

42. What's the difference between HAProxy and AWS ELB?

Answer:

- **HAProxy:** Self-managed, highly customizable.
- **AWS ELB:** Managed service with auto-scaling, health checks built-in.

---

43. What are connection draining and graceful shutdown?

Answer:
Connection draining ensures in-flight requests complete before removing a backend server from rotation during maintenance.

---

44. How can you optimize Nginx performance?

Answer:

- Use `worker_processes auto;`
- Enable `keepalive` connections
- Gzip compression
- Cache static files

---

45. What's the difference between load balancing and failover?

Answer:
Load balancing distributes load among servers; failover activates standby servers when primary fails.

---

46. How do you balance dynamic vs static content?

Answer:
Static → CDN or edge cache
Dynamic → Application layer via LB

---

47. What's the role of DNS TTL in load balancing?

Answer:
Controls how long clients cache DNS records—affects how fast failover occurs.

---

48. How would you handle DDoS attacks?

Answer:

- Use CDN with rate limiting
- WAF (Web Application Firewall)
- Blackhole routing for bad IPs

---

49. What's a health probe in Azure/AWS load balancers?

Answer:
Periodic checks (HTTP/TCP) to verify backend health and route traffic only to healthy nodes.

---

50. Design Question: How to design an image hosting service like Imgur?

Answer:
Architecture:

- LB → App Server → Object Store (S3)
- CDN for delivery
- Metadata DB for indexing
- Workers for thumbnail generation
- Cache hot images in Redis

Trade-offs:

- Optimize cost by cold storage tiering
- Handle burst uploads with queues
- Compress images dynamically

---

51. Bonus: How to ensure zero downtime deployment?

Answer:
Use **Blue-Green Deployment** or **Canary Releases** behind the LB, routing traffic gradually to new instances.

General Load Balancing & Scaling

1. What is load balancing? Why is it needed?
    - Load balancing is the process of distributing incoming network traffic across multiple backend servers, ensuring no single server becomes a bottleneck and improving reliability, performance, and scalability.
2. What's the difference between horizontal and vertical scaling?
    - Horizontal scaling adds more servers to handle load, while vertical scaling increases the resources (CPU, RAM) of a single machine.
3. Which type of scaling is more resilient to failure—and why?
    - Horizontal scaling: if one server fails, others can still serve requests, so it's more fault tolerant.
4. Name three scenarios where load balancing is critical.
    - High-traffic web applications, distributed databases, and any system where downtime is unacceptable (e.g., e-commerce checkout).

5. What is a single point of failure, and how do load balancers help avoid it?
    - A single point of failure is any part of the system whose failure brings down the entire service; load balancers reduce this risk by distributing requests and supporting failover.
6. What is server health checking, and why is it important for load balancers?
    - Load balancers periodically check if servers are healthy; unhealthy servers are removed from rotation to avoid failed requests and downtime.
7. How does session persistence ("sticky sessions") impact load balancing?
    - Sticky sessions ensure a user's requests always reach the same backend server, necessary when user state isn't shared, but can lead to uneven load.
8. Describe basic traffic distribution without a load balancer versus with one.
    - Without: All requests go to a single server. With: Traffic is shared across multiple servers, preventing overload.
9. How can horizontal scaling improve application performance?
    - By adding servers, you can handle more simultaneous users and requests, reducing latency and queuing.
10. How do load balancers contribute to zero-downtime deployments?
    - They can route traffic away from servers being updated, ensuring ongoing users aren't interrupted.
11. Why might you place a load balancer behind another load balancer?
    - For redundancy (failover), global/regional routing, or to handle higher levels of load/distributed traffic.
12. What is failover? Give an example in a load balanced system.
    - Automatic handling of server failures; load balancer reroutes requests to healthy servers if one fails.
13. When would you choose vertical scaling despite its downsides?
    - If the application can't be distributed easily, has strict licensing, or needs large memory/CPU for data processing.
14. If a server behind your load balancer is slow but not failing health checks, what problems might users face?
    - Higher response times, degraded experience; advanced health checks may be needed to catch performance issues, not just outages.
15. Can load balancing help against DDoS attacks? Why or why not?
    - It helps absorb larger volumes, but is not a complete solution—application-layer DDoS may still overwhelm all backends unless mitigated.

Algorithms, HAProxy & Nginx

16. Describe the round robin load balancing algorithm. When is it best used?
    - Requests are distributed evenly in a cycle to each server. Best for stateless applications.
17. Explain consistent hashing. Why is it valuable for file/image storage?
    - Requests are assigned to a server using a hash of some property (e.g. filename), ensuring requests for the same file/user go to the same server. This minimizes data movement if servers are added/removed.
18. Compare least connections and round robin.

- Least connections sends each new request to the server with the fewest current connections, suited for servers with variable load; round robin gives equal chance to each, regardless of current usage.

19. How do you configure HAProxy for basic round robin balancing?
    - Define frontends (listen for traffic) and backends (list of servers); by default, HAProxy uses round robin.
20. What ports does HAProxy commonly use by default?
    - HAProxy can listen on any configurable port, but typically uses 80 (HTTP) and 443 (HTTPS) in web setups.
21. What's the function of an 'upstream' block in Nginx?
    - It defines a group of backend servers to which requests can be proxied.
22. How does weighted round robin work?
    - Each server is given a weight; servers with higher weights get more requests proportionally (good for unequal capacity).
23. What is session persistence, and how can it be implemented in HAProxy/Nginx?
    - Session persistence pins a user to a specific server—implemented via cookies, IP hash, or application-level stickiness.
24. Why might you use IP hash in a proxy/load balancer?
    - To always send requests from the same user (IP) to the same backend server—useful when sessions must remain sticky but not handled by cookies.
25. Give an example of an Nginx configuration to act as a reverse proxy to two file servers.
26. `text`

```
upstream backend {
  server file1.internal;
  server file2.internal;
}
server {
  listen 80;
  location /files/ {
    proxy_pass http://backend;
  }
}
```

27.

28. What is the 'frontend' section in HAProxy?
    - Specifies where and how HAProxy receives incoming traffic (IP, port, protocol).
29. What is the 'backend' section in HAProxy?
    - Lists the pool of servers to which traffic will be forwarded, along with algorithm and health check settings.
30. Explain how HAProxy performs health checks.
    - By periodically sending connection requests or HTTP checks to backends; unhealthy servers are removed from rotation.
31. Can Nginx provide Layer 7 (HTTP) load balancing? Describe briefly.

- Yes—it uses application-level (HTTP) rules, forwarding traffic based on request path, headers, or cookies.

32. How do you scale HAProxy horizontally?
    - Add a secondary load balancer in active-active or active-passive configuration; use DNS or another LB to distribute traffic between them.

Reverse Proxy & CDN Basics

31. What is a reverse proxy?
    - A server that accepts requests from clients and forwards them to one or more backend servers, then sends the response back.
32. List three key benefits of using a reverse proxy.
    - Hides internal backend structure, enables SSL offloading, provides caching and load balancing.
33. Compare a reverse proxy and a forward proxy.
    - Reverse proxies act on behalf of servers, forward proxies act on behalf of clients.
34. Describe how SSL termination works in a reverse proxy.
    - The proxy handles incoming encrypted traffic, decrypts it, and sends plain HTTP to backend; reduces CPU load on app servers.
35. What is CDN and why use it for static file/image hosting?
    - Content Delivery Network; accelerates delivery by caching content close to users worldwide, reduces origin server load.
36. How can reverse proxies enable rate limiting?
    - By monitoring incoming connections/requests and applying limits before requests reach backends.
37. How does caching by a reverse proxy help performance?
    - The proxy can return cached responses for repeated requests, reducing backend server load and response times.
38. Describe the role of edge servers in a CDN.
    - Edge servers maintain cached copies of content in geographically distributed locations, serving users rapidly and reducing latency.
39. What happens if a file is requested and not present in an edge cache?
    - The CDN fetches it from the origin server and stores it in cache for future requests.
40. Name two CDN providers.
    - Cloudflare, Akamai, AWS CloudFront, or Fastly.

File/Image Server System Design

41. Outline the architecture of a scalable file upload/download system using load balancing.
    - Users upload/download via a load balancer (Nginx/HAProxy), which forwards requests to backend servers, files stored centrally or sharded, CDN used for fast global distribution.
42. List the main bottlenecks of a naive (single-server) file server.
    - Limited bandwidth, CPU/memory, single point of failure, poor concurrency.
43. How can storing file metadata in a database enhance scalability?

- Enables distribution of file data separately from metadata, scaling storage (for files) and lookup (database) independently.

44. When is consistent hashing especially important in a file server system?
    - When distributing file storage or cache across many nodes, so minimal data needs to be moved if nodes are added/removed.
45. How might you design an upload workflow to optimize for global users?
    - Accept uploads in a nearby region, store to shared/replicated storage, notify all CDN edge nodes, deliver via edge caching on first request.
46. What are some strategies for handling large file uploads/downloads efficiently?
    - Use chunked upload (splitting large files); resume interrupted downloads; stream files directly from storage.
47. List at least three challenges unique to file/image server scaling.
    - Synchronizing file metadata with storage, cache invalidation on updates, supporting versioning, handling hot files (popular downloads).
48. If two users upload files with the same name, how could conflicts be avoided?
    - Store files with unique IDs (UUIDs or hashes), map to original names in metadata.
49. What role does a distributed lock or consensus service (like etcd or Zookeeper) play in a high-scale file service?
    - Ensures actions like file/metadata updates are coordinated, preventing conflicts and data loss.
50. How do you secure user file downloads in a distributed system?
    - Sign URLs with short TTL, validate tokens at the backend, require authentication/authorization before file access.
51. How do you ensure file consistency across multiple regions?
    - Replicate data, use eventual consistency model, resolve conflicts, ensure metadata is synced.
52. What are possible approaches for cache invalidation in image/file servers?
    - Time-based expiry (TTL), explicit purge on file update/delete, versioned URLs for cache busting.
53. How can you monitor and alert for system failures in a distributed file/image hosting service?
    - Use centralized logging, real-time metrics (Prometheus/Grafana), and automated health checks; alert on anomalies or backend error rates.
54. Describe a stateless vs stateful backend in this architecture.
    - Stateless: does not track previous user requests (best for load balancing); stateful: maintains session or transient file information.
55. What's the trade-off between storing files in a database vs a distributed object store?
    - Databases have size/performance limits, object stores are better for large files and can scale horizontally, but may not provide transactional guarantees or fast random access.
56. How would you benchmark the scalability of your system?
    - Simulate high-concurrency uploads/downloads, measure response times, throughput, error rates, backend utilization; test with and without CDN/cache.