System Design Questions with Solutions for Engineering Managers

This document provides comprehensive system design questions along with detailed solutions, frameworks, and EM-specific guidance for Engineering Manager interviews.

Table of Contents

System Design Interview Framework

The 5-Step Approach for EMs

Step 1: Requirements Clarification (5-10 minutes)

What to Ask:
- Functional requirements (features, user actions)
- Non-functional requirements (scale, performance, availability)
- Constraints (budget, timeline, existing systems)
- Success metrics and business goals

EM Focus: Show business acumen and stakeholder thinking.

Step 2: High-Level Design (10-15 minutes)

What to Do:
- Draw major components and data flow
- Start simple, avoid premature optimization
- Focus on core user journey
- Identify key services and databases

EM Focus: Demonstrate architectural thinking and communication skills.

Step 3: Deep Dive (15-20 minutes)

What to Cover:
- Choose 1-2 critical components
- Discuss data models and algorithms
- Address specific technical challenges
- Show technical depth

EM Focus: Balance technical details with team and process considerations.

Step 4: Scale and Optimize (10-15 minutes)

What to Address:
- Identify bottlenecks

- Discuss scaling strategies
- Add caching, load balancing, partitioning
- Consider global distribution

EM Focus: Show operational thinking and team coordination needs.

Step 5: Edge Cases and Operations (5-10 minutes)

What to Consider:
- Failure scenarios and recovery
- Monitoring and alerting
- Security and compliance
- Maintenance and updates

EM Focus: Demonstrate risk management and operational leadership.

Detailed Solutions by Category

Data & AI/ML Systems

Question: Design Netflix's Content Recommendation System

Requirements Clarification:
- **Scale**: 200M+ users, 15K+ titles, billions of interactions daily
- **Features**: Personalized recommendations, trending content, similar titles
- **Performance**: <100ms response time, 99.9% availability
- **Business Goals**: Increase engagement, reduce churn, optimize content spend

High-Level Architecture:

```
[User] → [API Gateway] → [Recommendation Service] → [ML Models]
                              ↓
[Content Service] ← [Feature Store] ← [Data Pipeline]
                              ↓
[User Profile Service] ← [Real-time Events] ← [Streaming Data]
```

Deep Dive - ML Pipeline:

1. **Data Collection:**
   - User interactions (views, ratings, searches, time spent)
   - Content metadata (genre, actors, release date, duration)
   - Contextual data (time, device, location)

2. **Feature Engineering:**
   - User embeddings (viewing history, preferences)
   - Content embeddings (collaborative filtering, content-based)
   - Contextual features (time of day, device type)

3. **Model Architecture:**
   - **Collaborative Filtering**: Matrix factorization for user-item interactions
   - **Content-Based**: Deep learning on content features
   - **Hybrid Model**: Ensemble combining multiple approaches
   - **Real-time Personalization**: Online learning for immediate feedback

4. **Serving Infrastructure:**
   - **Model Serving**: TensorFlow Serving or similar for low-latency inference
   - **Feature Store**: Redis/Cassandra for fast feature lookup
   - **A/B Testing**: Framework for model experimentation
   - **Caching**: Multi-level caching for popular recommendations

Scaling Considerations:
- **Horizontal Scaling**: Microservices architecture with independent scaling
- **Data Partitioning**: Shard by user ID for user data, content ID for content data
- **Caching Strategy**: L1 (CDN), L2 (Application), L3 (Database)
- **Global Distribution**: Regional deployments with content localization

EM Leadership Considerations:
- **Team Structure**: ML engineers, data scientists, backend engineers, data engineers
- **Cross-functional Collaboration**: Product (metrics), Content (catalog), Data (infrastructure)
- **Experimentation Culture**: A/B testing framework, metrics-driven decisions
- **Technical Debt**: Balance model complexity with maintainability
- **Talent Development**: Upskill team on latest ML techniques

Monitoring and Operations:
- **Model Performance**: Accuracy, precision, recall, engagement metrics
- **System Health**: Latency, throughput, error rates, availability
- **Business Metrics**: Click-through rate, watch time, user satisfaction
- **Data Quality**: Feature drift, model degradation, data pipeline health

---

Question: Design TikTok's For You Page Algorithm

Requirements Clarification:
- **Scale**: 1B+ users, 100M+ videos uploaded daily
- **Features**: Personalized feed, trending content, creator discovery
- **Performance**: <200ms feed generation, real-time updates
- **Business Goals**: Maximize engagement, creator monetization, content diversity

High-Level Architecture:

```
[Mobile App] → [API Gateway] → [Feed Service] → [Ranking Service]
                                      ↓
[Video Service] ← [Content Understanding] ← [ML Pipeline]
                                      ↓
[User Service] ← [Real-time Signals] ← [Event Stream]
```

Deep Dive - Ranking Algorithm:

1. **Content Understanding:**
   - **Video Analysis**: Computer vision for objects, scenes, activities
   - **Audio Analysis**: Music recognition, speech-to-text, sound classification
   - **Text Analysis**: Hashtags, captions, comments sentiment
   - **Creator Signals**: Follower count, engagement rate, content quality

2. **User Modeling:**
   - **Interaction History**: Likes, shares, comments, watch time

- **Behavioral Patterns**: Session length, time of day, content preferences
- **Social Signals**: Following relationships, friend interactions
- **Demographic Data**: Age, location, language, interests

3. **Ranking Model:**
   - **Candidate Generation**: Retrieve top 1000 videos from various sources
   - **Two-Tower Model**: User tower and video tower for efficient serving
   - **Deep Neural Network**: Multi-task learning for engagement prediction
   - **Real-time Features**: Recent interactions, trending signals

4. **Serving Pipeline:**
   - **Candidate Sources**: Following feed, trending, similar users, hashtag exploration
   - **Ranking**: Score all candidates and select top 20-50 for user
   - **Diversity**: Ensure content variety (creators, topics, formats)
   - **Freshness**: Balance popular content with new uploads

Scaling Considerations:
- **Real-time Processing**: Kafka/Pulsar for event streaming, Flink for real-time ML
- **Video Storage**: Distributed file system with CDN for global delivery
- **Model Serving**: GPU clusters for deep learning inference
- **Cache Strategy**: User feed cache, model prediction cache, content metadata cache

EM Leadership Considerations:
- **Algorithm Team**: ML engineers, data scientists, research scientists
- **Content Policy**: Balance engagement with safety and community guidelines
- **Creator Relations**: Ensure fair content distribution and creator growth
- **Global Considerations**: Cultural sensitivity, local content preferences
- **Ethical AI**: Bias detection, fairness metrics, responsible recommendations

---

Real-time & Communication Systems

Question: Design Uber's Real-time Ride Matching System

Requirements Clarification:
- **Scale**: 100M+ users, 5M+ drivers, 15M+ rides daily
- **Features**: Real-time matching, ETA calculation, dynamic pricing
- **Performance**: <5 seconds matching time, 99.99% availability
- **Business Goals**: Minimize wait time, maximize driver utilization, optimize pricing

High-Level Architecture:

```
[Rider App] → [API Gateway] → [Matching Service] → [Location Service]
[Driver App] ↗              ↓                    ↓
                    [Pricing Service] ← [Demand Prediction]
                            ↓
[Trip Service] ← [Route Optimization] ← [Maps Service]
```

Deep Dive - Matching Algorithm:

1. **Location Management:**
   - **Geospatial Indexing**: Use geohashing or R-tree for efficient location queries
   - **Real-time Updates**: WebSocket connections for driver location streaming

   - **Location Prediction**: Predict driver locations based on traffic and routes
   - **Geofencing**: City boundaries, airport zones, surge areas

2. **Matching Logic:**
   - **Supply-Demand Analysis**: Real-time analysis of riders vs drivers in each area
   - **Distance Calculation**: Haversine formula with road network optimization
   - **Driver Scoring**: Rating, acceptance rate, proximity, vehicle type
   - **Batch Matching**: Process multiple requests together for optimization

3. **Real-time Processing:**
   - **Event Streaming**: Kafka for location updates, ride requests, driver status
   - **Stream Processing**: Apache Flink for real-time matching decisions
   - **State Management**: Redis for active drivers, pending requests
   - **Notification System**: Push notifications for match confirmations

4. **Optimization Algorithms:**
   - **Hungarian Algorithm**: Optimal assignment for batch matching
   - **Machine Learning**: Predict driver acceptance probability
   - **Dynamic Pricing**: Surge pricing based on supply-demand imbalance
   - **Route Optimization**: Minimize total system travel time

Scaling Considerations:
- **Geographic Sharding**: Partition by city/region for independent scaling
- **Microservices**: Separate services for matching, pricing, routing, notifications
- **Caching**: Driver locations, route calculations, pricing models
- **Load Balancing**: Distribute matching load across multiple servers

EM Leadership Considerations:
- **Marketplace Team**: Algorithm engineers, data scientists, backend engineers
- **Operations Coordination**: City operations, driver relations, customer support
- **Regulatory Compliance**: Local transportation laws, data privacy
- **Performance Optimization**: Balance matching speed with quality
- **Crisis Management**: Handle surge events, system outages, safety incidents

---

Distributed Systems & Infrastructure

Question: Design a Content Delivery Network (CDN)

Requirements Clarification:
- **Scale**: Global distribution, 100TB+ daily traffic, 1M+ requests/second
- **Features**: Static content caching, dynamic content acceleration, DDoS protection
- **Performance**: <50ms latency globally, 99.99% availability
- **Business Goals**: Reduce origin load, improve user experience, minimize costs

High-Level Architecture:

```
[Users] → [Edge Servers] → [Regional Caches] → [Origin Servers]
          ↓
[DNS Resolution] ← [Load Balancer] ← [Health Monitoring]
          ↓
[Analytics] ← [Log Aggregation] ← [Edge Metrics]
```

Deep Dive - Caching Strategy:

1. **Cache Hierarchy:**
   - **Edge Caches**: Deployed globally, serve most requests
   - **Regional Caches**: Mid-tier caches, reduce origin load
   - **Origin Shield**: Protect origin servers from cache misses
   - **Browser Cache**: Client-side caching with proper headers

2. **Cache Management:**
   - **Cache Policies**: TTL-based, LRU eviction, content-aware policies
   - **Invalidation**: Real-time purging, versioned content, cache tags
   - **Warming**: Proactive content population, predictive caching
   - **Compression**: Gzip, Brotli compression for bandwidth optimization

3. **Routing and Load Balancing:**
   - **Anycast DNS**: Route users to nearest edge server
   - **Health Checks**: Monitor server health, automatic failover
   - **Load Distribution**: Consistent hashing, weighted round-robin
   - **Traffic Shaping**: Rate limiting, DDoS protection

4. **Content Optimization:**
   - **Image Optimization**: WebP conversion, responsive images
   - **Minification**: CSS, JavaScript compression
   - **HTTP/2 & HTTP/3**: Modern protocol support
   - **Edge Computing**: Serverless functions at edge locations

Scaling Considerations:
- **Global Distribution**: 200+ edge locations across 6 continents
- **Network Capacity**: Multi-Tbps backbone, redundant connections
- **Storage Scaling**: Distributed storage systems, automatic replication
- **Monitoring**: Real-time metrics, alerting, performance analytics

EM Leadership Considerations:
- **Infrastructure Team**: Network engineers, systems engineers, SRE team
- **Vendor Management**: ISP relationships, data center partnerships
- **Cost Optimization**: Bandwidth costs, server utilization, energy efficiency
- **Security**: DDoS mitigation, SSL/TLS management, threat detection
- **Compliance**: Data sovereignty, privacy regulations, content policies

---

EM Leadership Framework

Technical Leadership Dimensions

1. Architectural Decision Making

• **Technology Selection**: Choose appropriate technologies for scale and team skills

• **System Design**: Balance complexity, maintainability, and performance

• **Technical Debt**: Manage accumulation and prioritize reduction efforts

• **Standards**: Establish coding standards, review processes, documentation

2. Team Structure and Coordination

• **Team Topology**: Organize teams around business domains or technical components

- **Cross-team Dependencies**: Manage interfaces, APIs, and shared services
- **Skill Development**: Identify skill gaps and create development plans
- **Knowledge Sharing**: Establish practices for documentation and knowledge transfer

3. Process and Methodology

- **Development Process**: Agile, DevOps, continuous integration/deployment
- **Quality Assurance**: Testing strategies, code review, monitoring
- **Incident Management**: On-call processes, post-mortem culture, reliability
- **Planning**: Estimation, roadmap planning, capacity management

4. Stakeholder Management

- **Product Collaboration**: Work with PMs on requirements and priorities
- **Executive Communication**: Report on progress, risks, and resource needs
- **Customer Focus**: Understand user needs and translate to technical requirements
- **Vendor Management**: Evaluate and manage third-party services and tools

Common System Design Patterns

Scalability Patterns

1. **Horizontal Scaling**: Add more servers vs upgrading existing ones
2. **Load Balancing**: Distribute traffic across multiple servers
3. **Caching**: Multiple levels (CDN, application, database)
4. **Database Sharding**: Partition data across multiple databases
5. **Microservices**: Break monolith into independent services

Reliability Patterns

1. **Circuit Breaker**: Prevent cascade failures
2. **Bulkhead**: Isolate critical resources
3. **Timeout and Retry**: Handle transient failures
4. **Health Checks**: Monitor service health
5. **Graceful Degradation**: Maintain core functionality during failures

Data Patterns

1. **CQRS**: Separate read and write models
2. **Event Sourcing**: Store events instead of current state
3. **Data Lake**: Store raw data for analytics
4. **CDC**: Capture database changes for real-time processing
5. **Polyglot Persistence**: Use different databases for different needs

Evaluation Criteria

Technical Competency (40%)

- **System Design Skills**: Ability to design scalable, reliable systems
- **Technology Knowledge**: Understanding of modern technologies and patterns
- **Problem Solving**: Approach to breaking down complex problems
- **Trade-off Analysis**: Ability to evaluate different solutions

Leadership Capability (35%)

- **Team Management**: Experience building and leading engineering teams

- **Communication**: Ability to explain technical concepts to different audiences
- **Decision Making**: Process for making technical and organizational decisions
- **Influence**: Ability to drive consensus and influence without authority

Business Acumen (25%)

- **Product Thinking**: Understanding of user needs and business requirements
- **Resource Management**: Ability to plan and optimize resource allocation
- **Risk Management**: Identification and mitigation of technical and business risks
- **Stakeholder Management**: Effective collaboration with cross-functional teams

Red Flags to Avoid

- **Over-engineering**: Adding unnecessary complexity
- **Under-engineering**: Ignoring scalability and reliability requirements
- **Poor Communication**: Unable to explain design decisions clearly
- **Lack of Leadership**: Focusing only on technical details without team considerations
- **Inflexibility**: Unwilling to adapt design based on feedback

Success Indicators

- **Clear Communication**: Explains design decisions and trade-offs clearly
- **Systematic Approach**: Follows structured approach to system design
- **Leadership Mindset**: Considers team, process, and organizational aspects
- **Practical Solutions**: Balances ideal solutions with real-world constraints
- **Continuous Learning**: Shows awareness of current trends and best practices

---

This document provides detailed solutions and frameworks for system design interviews. Practice these patterns and adapt them to your experience and the specific company context.