System Design Guide: 10 Case Studies with Illustrated Solutions

July 2025

Contents

1 Introduction **2**

2 Case Studies **2**

2.1 Case Study 1: URL Shortening Service (e.g., TinyURL) 2
2.2 Case Study 2: Social Media Feed (e.g., Twitter) 2
2.3 Case Study 3: Ride-Sharing Service (e.g., Uber) 3
2.4 Case Study 4: Video Streaming Service (e.g., YouTube) 4
2.5 Case Study 5: Chat Application (e.g., WhatsApp) 5
2.6 Case Study 6: E-Commerce Platform (e.g., Amazon) 5
2.7 Case Study 7: Notification System 6
2.8 Case Study 8: File Storage System (e.g., Dropbox) 6
2.9 Case Study 9: Search Engine (e.g., Google) 7
2.10 Case Study 10: Real-Time Analytics System 8

1 Introduction

This guide presents 10 system design case studies, each addressing a real-world problem with detailed solutions and architectural diagrams. Designed for software engineers and architects, it covers scalable, reliable, and efficient system designs, illustrating key concepts like load balancing, caching, and database sharding.

2 Case Studies

2.1 Case Study 1: URL Shortening Service (e.g., TinyURL)

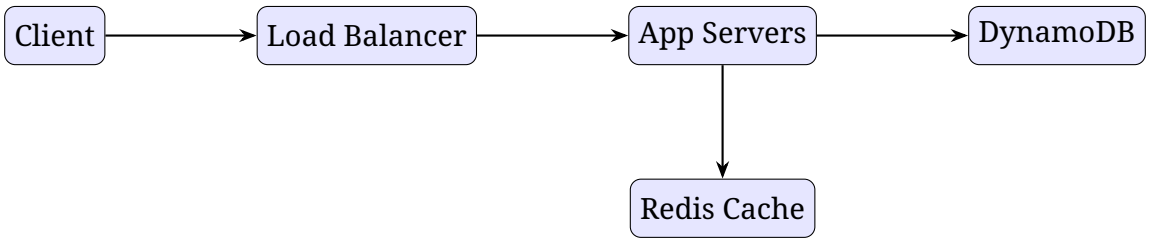
Problem: Design a service to shorten long URLs and redirect users to the original URL.
Requirements:

- Generate a unique short URL for each long URL.
- Handle high read and write traffic.
- Ensure low latency for redirection.

Solution:

- **API**: REST endpoints (`POST /shorten`, `GET /{shortURL}`).
- **Hash Function**: Base62 encoding for short URLs.
- **Database**: NoSQL (e.g., DynamoDB) for URL mappings.
- **Caching**: Redis for frequent redirects.
- **Load Balancer**: Distribute traffic across servers.

Diagram:



Explanation: The load balancer distributes requests to app servers, which check the cache before querying the database. Base62 ensures compact, unique short URLs.

2.2 Case Study 2: Social Media Feed (e.g., Twitter)

Problem: Design a system to display a user's feed with posts from followed users.
Requirements:

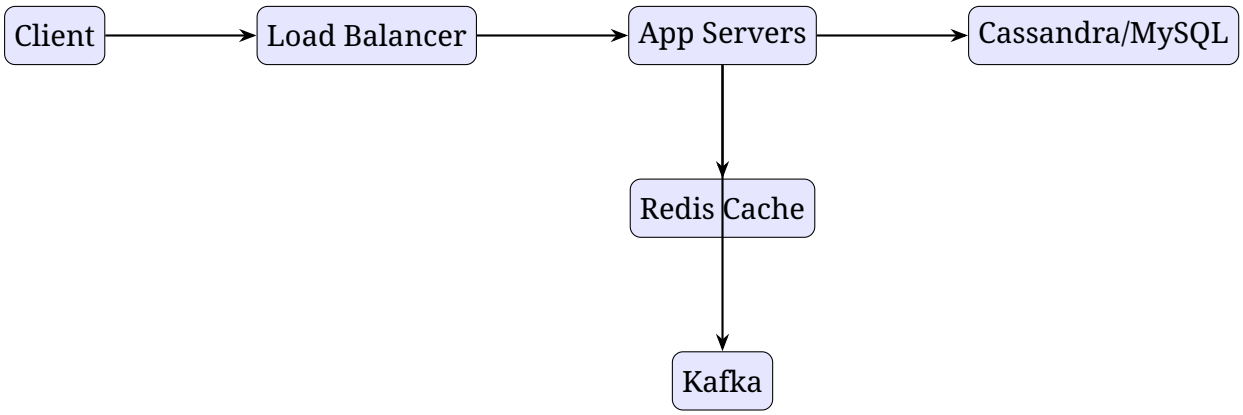
- Low-latency feed generation.
- Handle millions of users and posts.
- Support real-time updates.

Solution:

- **Fanout Service**: Push posts to followers' feeds (write-time fanout).

- **Database**: NoSQL for posts (Cassandra) and relational for user relationships (MySQL).
- **Caching**: Redis for recent posts.
- **Message Queue**: Kafka for asynchronous fanout.

Diagram:



Explanation: Fanout on write pre-computes feeds, stored in cache for fast retrieval. Kafka handles asynchronous updates.

2.3 Case Study 3: Ride-Sharing Service (e.g., Uber)

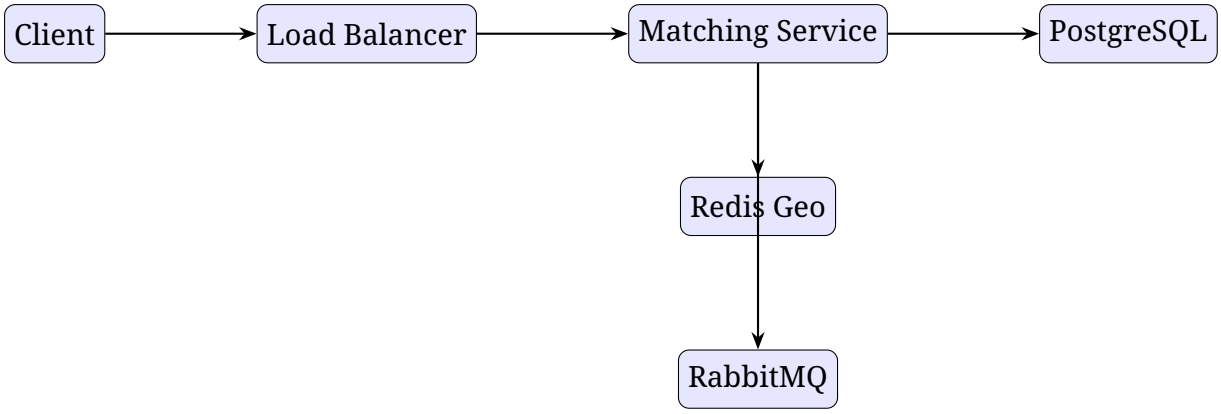
Problem: Design a system to match riders with drivers and track rides.
Requirements:

- Real-time driver-rider matching.
- Scalable location tracking.
- Handle payment processing.

Solution:

- **Geospatial Index**: Redis for driver location queries.
- **Matching Service**: Match based on proximity and availability.
- **Database**: PostgreSQL for ride data, MongoDB for logs.
- **Message Queue**: RabbitMQ for ride events.

Diagram:



Explanation: Redis geospatial indices enable fast driver lookups. RabbitMQ ensures reliable event processing.

2.4 Case Study 4: Video Streaming Service (e.g., YouTube)

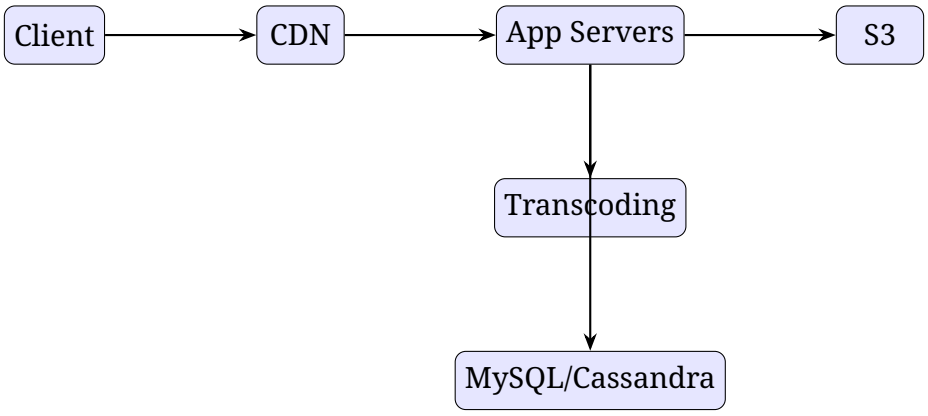
Problem: Design a system to stream videos to millions of users.
Requirements:

- Low-latency streaming.
- Support various resolutions.
- Scalable storage and delivery.

Solution:

- **CDN**: Cloudflare for video delivery.
- **Storage**: S3 for video files.
- **Transcoding Service**: Convert videos to multiple formats.
- **Database**: MySQL for metadata, Cassandra for analytics.

Diagram:



Explanation: CDN reduces latency by caching videos globally. Transcoding ensures compatibility across devices.

2.5 Case Study 5: Chat Application (e.g., WhatsApp)

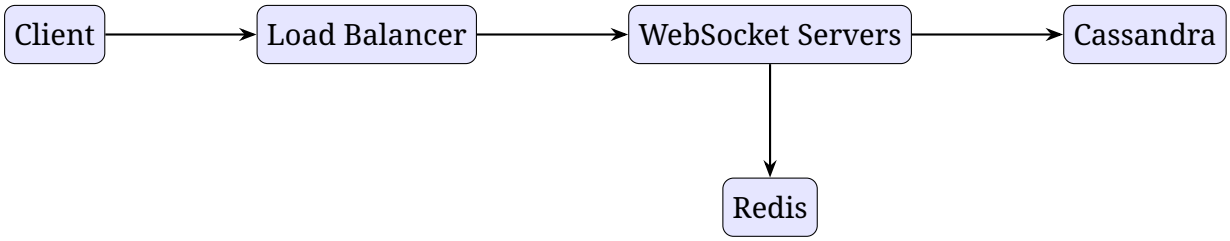
Problem: Design a real-time chat system for millions of users.
Requirements:

- Low-latency message delivery.
- Support group chats and media.
- Ensure message persistence.

Solution:

- **WebSocket**: For real-time messaging.
- **Database**: Cassandra for message storage.
- **Caching**: Redis for active sessions.
- **Load Balancer**: NGINX for WebSocket connections.

Diagram:



Explanation: WebSocket enables bidirectional communication. Cassandra ensures scalable message storage.

2.6 Case Study 6: E-Commerce Platform (e.g., Amazon)

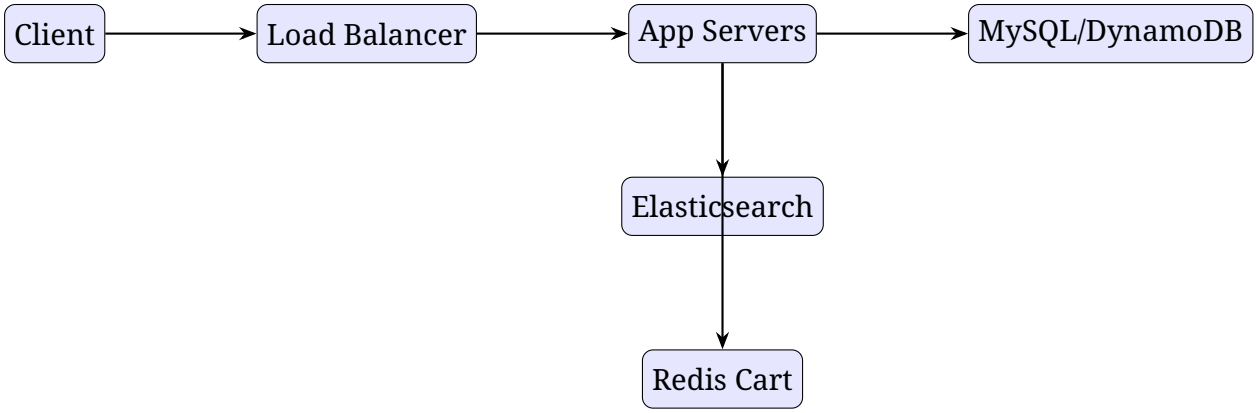
Problem: Design a scalable e-commerce system for product listings, cart, and checkout.
Requirements:

- High availability for product search.
- Handle peak traffic (e.g., Black Friday).
- Secure payment processing.

Solution:

- **Search**: Elasticsearch for product search.
- **Cart Service**: Redis for session-based carts.
- **Payment Gateway**: Integrate with Stripe.
- **Database**: MySQL for orders, DynamoDB for inventory.

Diagram:



Explanation: Elasticsearch provides fast search. Redis ensures quick cart access during peak loads.

2.7 Case Study 7: Notification System

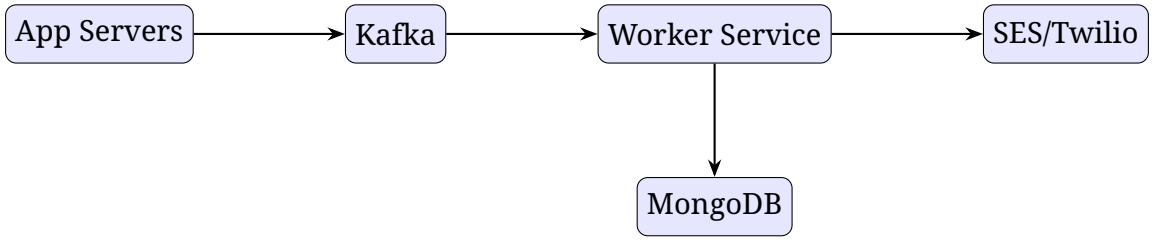
Problem: Design a system to send notifications (email, SMS, push).
Requirements:

- Support multiple channels.
- Handle high throughput.
- Ensure delivery reliability.

Solution:

- **Message Queue**: Kafka for notification events.
- **Worker Service**: Process notifications asynchronously.
- **External Services**: AWS SES for email, Twilio for SMS.
- **Database**: MongoDB for notification logs.

Diagram:



Explanation: Kafka decouples notification triggers from delivery, ensuring scalability. Workers handle retries for reliability.

2.8 Case Study 8: File Storage System (e.g., Dropbox)

Problem: Design a system for storing and syncing files across devices.
Requirements:

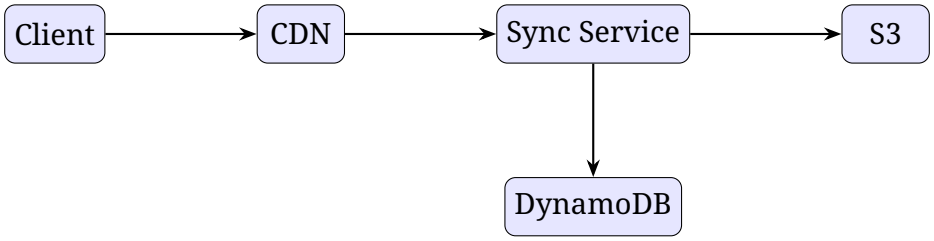
- Scalable file storage.
- Support file versioning.

- Efficient sync mechanism.

Solution:

- **Storage**: S3 for file storage.
- **Metadata**: DynamoDB for file metadata and versions.
- **Sync Service**: Delta encoding for efficient updates.
- **CDN**: For fast file access.

Diagram:



Explanation: S3 provides scalable storage, while DynamoDB tracks metadata. Delta encoding minimizes bandwidth usage.

2.9 Case Study 9: Search Engine (e.g., Google)

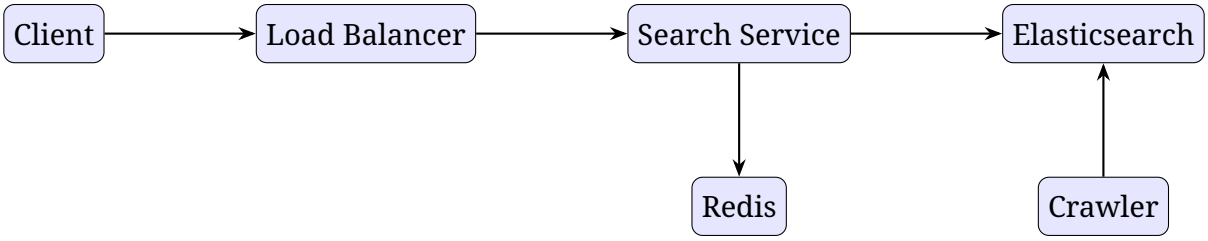
Problem: Design a system to index and search web content.
Requirements:

- Fast search results.
- Handle billions of pages.
- Support ranking and relevance.

Solution:

- **Crawler**: Fetches web content.
- **Indexer**: Elasticsearch for inverted indices.
- **Ranking Service**: Machine learning for relevance.
- **Cache**: Redis for popular queries.

Diagram:



Explanation: Elasticsearch enables fast searches. Caching popular queries improves performance.

2.10 Case Study 10: Real-Time Analytics System

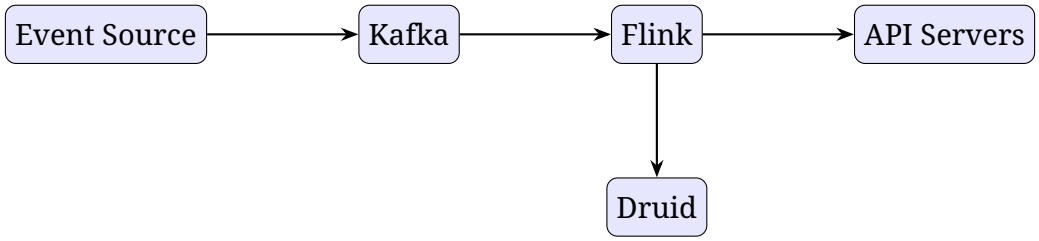
Problem: Design a system to process and analyze real-time event data.
Requirements:

- Handle high event throughput.
- Provide real-time insights.
- Support historical analysis.

Solution:

- **Stream Processing**: Apache Flink for real-time analytics.
- **Storage**: Druid for time-series data.
- **Queue**: Kafka for event ingestion.
- **API**: Serve aggregated results.

Diagram:



Explanation: Kafka ingests events, Flink processes them in real-time, and Druid stores results for querying.