Ad Click Charging System - Architecture

High-Level Architecture Diagram

```
┌─────────────────┐    ┌──────────────────┐    ┌─────────────────┐
│   Web Browser   │    │  Mobile App      │    │ Third Party Pubs│
│                 │    │                  │    │ (Future Ext.)   │
└─────────┬───────┘    └─────────┬────────┘    └─────────┬───────┘
          │                       │                       │
          └───────────────────────┼───────────────────────┘
                                  │
                    ┌────────────▼─────────────┐
                    │   Load Balancer          │
                    │      (NGINX)             │
                    └────────────┬─────────────┘
                                  │
                    ┌────────────▼─────────────┐
                    │  Click Ingestion API    │
                    │     (Node.js)           │
                    └────────────┬─────────────┘
                                  │
                    ┌────────────▼─────────────┐
                    │  Message Queue          │
                    │   (Apache Kafka)        │
                    └─────────┬───┬───┬───────┘
                              │   │   │
              ┌───────────────┘   │   └───────────────┐
              │                   │                   │
    ┌─────────▼─────────┐ ┌───────▼───────┐ ┌─────────▼─────────┐
    │ Fraud Detection   │ │ Billing       │ │ Analytics         │
    │ Service           │ │ Service       │ │ Service           │
    └─────────┬─────────┘ └───────┬───────┘ └─────────┬─────────┘
              │                   │                   │
    ┌─────────▼─────────┐ ┌───────▼───────┐ ┌─────────▼─────────┐
    │ Fraud Database    │ │ Billing DB    │ │ Analytics DB      │
    │     (MySQL)       │ │    (MySQL)    │ │     (MySQL)       │
    └───────────────────┘ └───────────────┘ └───────────────────┘
```

Component Details

1. Click Ingestion Layer

This is where all the clicks will first enter the system.

- **Load Balancer (NGINX)**: Using NGINX as it's lightweight and can handle concurrent connections. It will spread the incoming click requests across multiple API instances so no single server gets overwhelmed.

- **Click Ingestion API (Node.js)**: A simple Node.js HTTP server. The primary job of this server is to validate the click data and add some additional information like timestamps and IP addresses before passing it along.
- **Rate Limiting**: Basic rate limiting to prevent Denial of Service attacks. Set it to 1000 clicks per minute per IP address

2. Message Queue (Pub-Sub Core)

Heart of the pub-sub architecture. Using Kafka for pub-sub

- **Apache Kafka**: Selected Kafka over Redis for messaging because it can handle much higher throughput and has built-in persistence. So, if the service goes down, we won't lose any click events.
- **Topics Structure**: Designed three main topics:
  - `click-events`: Raw clicks from the API (before any processing)
  - `validated-clicks`: Clicks that cleared fraud detection
  - `billing-events`: Clicks that were successfully charged
- **Publisher/Subscriber Pattern**: The ingestion API publishes to click-events, then each service subscribes to what it needs. This way if we want to add a new service later (eg reporting), just need to subscribe it to the right topics.

3. Processing Services

Split the processing into three separate services as per microservices architecture:

- **Fraud Detection Service**: This is the most complex component. Simple checks based on velocity and location. We will implement a basic fraud detection using IP analysis, click velocity checks, and user agent validation. It's not going to be as sophisticated as real ad platforms, but would catch obvious bot traffic.
- **Billing Service**: Handles the money side of things. It calculates costs based on campaign bid amounts and updates advertiser budgets. Will use transactions here since we're dealing with financial data.
- **Analytics Service**: Collects real-time metrics. Subscribes to billing events and creates hourly/daily aggregations for reporting.

4. Data Layer

Kept the database architecture simple by using MySQL everywhere:

- **MySQL Databases**: Using MySQL for all persistent storage needs:
  - **Fraud Database**: Stores click validation results and fraud scores
  - **Billing Database**: Financial transactions and campaign budgets
  - **Analytics Database**: Time-series data and pre-calculated metrics
- **Redis Cache**: Used separately for caching session data and rate limiting counters. Much faster than hitting MySQL for these frequent operations.

Design Decisions and Challenges

Why Pub-Sub Architecture

Selected pub-sub architecture for the following reasons:

- **Decoupling**: Each service operates independently. If the analytics service crashes, billing continues to work.
- **Scalability**: Can add more instances of any service without modifying others.
- **Flexibility**: Adding new features requires only subscribing to the appropriate topics.

Implementation Challenges

- **Message Ordering**: Addressed out-of-order processing by using Kafka partitioning with campaign_id as the key.
- **Exactly-Once Processing**: Prevented duplicate charges by implementing idempotency checks using unique event IDs.
- **Database Performance**: Added proper indexes and connection pooling to handle high click volumes.

Future Improvements

- **Error Handling**: Implement dead letter queues for failed message processing.
- **Monitoring**: Add comprehensive metrics and alerting beyond basic logging.
- **Testing**: Expand test coverage with more comprehensive unit and integration tests.

Technology Choices

Selected Technologies

- **Node.js**: Handles concurrent connections efficiently, suitable for high-volume click processing.

- **Apache Kafka**: Chosen over Redis and RabbitMQ for its ability to handle millions of messages with built-in replication and persistence.

- **MySQL**: Selected for operational simplicity. Can handle both transactional and analytical workloads effectively.

- **NGINX**: Lightweight load balancer with good performance characteristics and extensive documentation.

- **Docker**: Ensures consistent deployment across different environments and simplifies service management.

Performance Targets

Based on research of production ad platforms:

- 10,000 clicks per second throughput
- Under 100ms API response time
- Under 50ms fraud detection processing
- 99.9% system availability

The architecture is designed to support these targets with proper optimization and tuning.

Implementation Status

Completed Components

- Click ingestion and validation
- Kafka message publishing and consuming
- Basic fraud detection for bot traffic
- Billing calculations with budget controls
- Real-time analytics aggregation

Remaining Work

- Enhanced fraud detection models
- Comprehensive load testing
- Monitoring dashboard implementation
- Improved error handling and recovery mechanisms
- Security enhancements (authentication, encryption)

Key Insights

The pub-sub pattern provides significant scalability benefits but introduces operational complexity. Distributed system debugging requires different approaches compared to monolithic applications. The architecture demonstrates viability for production ad platforms with additional refinement and optimization.