CS 237
Billing Service with Kafka Middleware

•••

Dan Morgan (55086407)
Mazhar Abbass (73877540)
Siddharth Joshi (30807928)

Motivation & Goals

Motivation

Designing a complete system like a billing service, which uses message oriented middleware such as Kafka, scale it and evaluate its performance.

Goal

To Build a billing service using Kafka as middleware and server assembly of producers and consumers.

Simulate the real word billing application scenario and make different client and producer platforms.

Goal

Write and perform the test cases to evaluate the performance of the Kafka cluster.

Find parameters like Producer Latency, Consumer Latency.

Related Work

RabbitMQ

RabbitMQ is a solid, mature, general purpose message broker that supports several standardized protocols such as AMQP

It does not have horizontal scaling unlike Kafka

For testing, it has a performance tool which reports the rate at which messages are sent and received, along with the latency.

ActiveMQ

Implements the Java Message Service specification

Support for enterprise integration and Spring framework

Is has better cost performance and reliability but lacks in speed when compared to modern brokers like Kafka

Redis

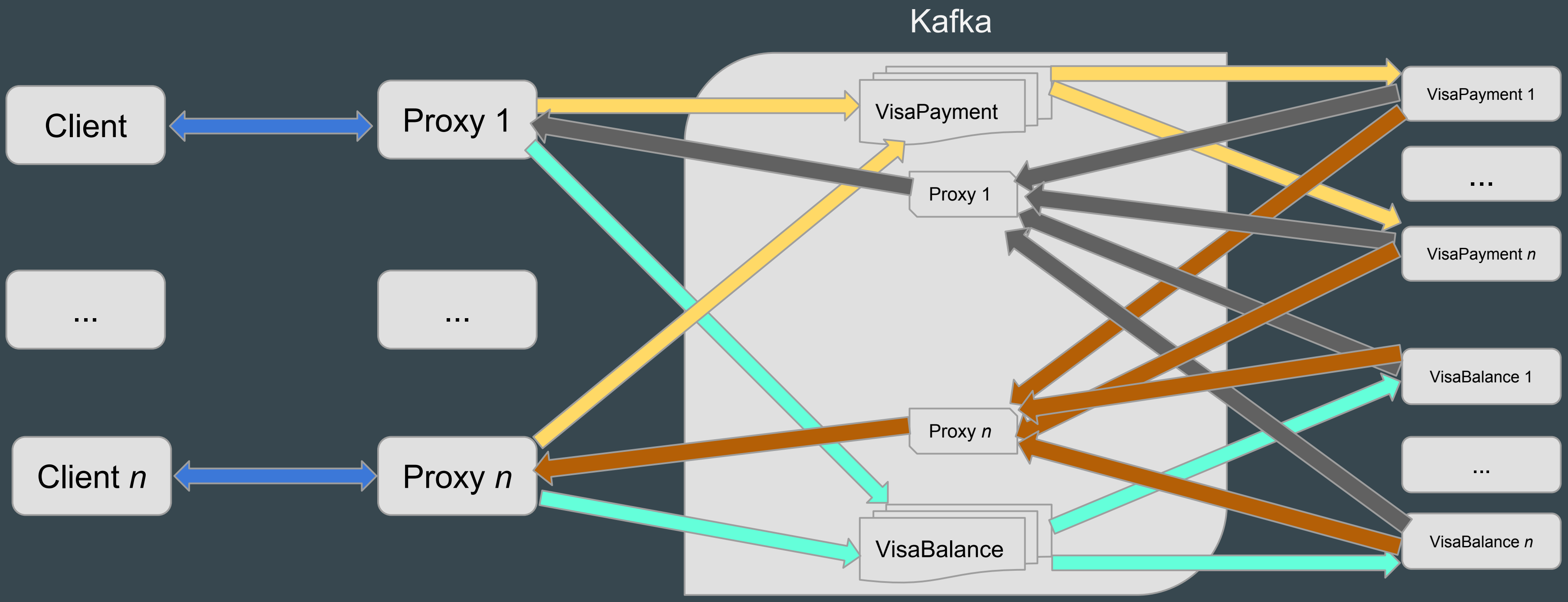
In-memory database that has pub-sub features

Better for short lived messages

No persistence but high speed

Supports data structures like string, hash tables, lists

System Prototype



Design specifics

C# Clients/Proxies

.NET Core Console Apps (x3)

- gRPC with Protobuf messages for communication between client and proxy
- Clients round-robin between proxies
- Proxies publish Protobuf messages to “Visa” Kafka topic
- Proxies consume replies from process-specific topic

Kafka Middleware

kafka_2.11-0.10.2.0

- 3 AWS servers
- 1 GB RAM
- 3 broker cluster
- Topics auto-created
- num.partitions=5
- 1 Gb Device RAM.
- 256 MB Heap memory
- 3 zookeeper ensemble
- Low leader imbalance check

C# Payment/Balance Processors

.NET Core Console Apps (x6)

- Kafka consumers from “VisaPayment” (x3) and “VisaBalance” (x3) topics
- Uses “Subscribe” API so Kafka will auto-balance partitions across consumers
- Publish response to proxy-specific topic delivered in originally consumed message

Evaluation Plan and Results

- Perform simultaneous requests per client, so “5,000” requests is 15,000 total being handled by the system
- Measure system response time from the client (time from request sent to response received)
- Perform tests with one backend processor first, then add two more and re-run
  - Kafka rebalances the topic partitions between the three
- Three processors slightly faster
  - Limited by AWS t2.micro instances; at higher rates clients and proxies would run out of memory
  - Might have been gRPC server memory leak

One Backend Processor

5,000 requests
- Shortest Response: 1.864751s
- Longest Response: 3.095911s
- Average Response: 2.378374s

10,000 requests
- Shortest Response: 1.737001s
- Longest Response: 4.198972s
- Average Response: 2.673143s

25,000 requests
- Shortest Response: 1.770683s
- Longest Response: 7.885425s
- Average Response: 4.65759s

Three Backend Processors

5,000 requests
- Shortest Response: 1.883761s
- Longest Response: 3.168537s
- Average Response: 2.416574s

10,000 requests
- Shortest Response: 1.681944s
- Longest Response: 4.292545s
- Average Response: 2.653532s

25,000 requests
- Shortest Response: 1.662826s
- Longest Response: 7.893259s
- Average Response: 4.449785s

Evaluation Plan and Results - Round 2

- Same as before, except now there are two backend topics, VisaPayment and VisaBalance
- VisaPayment has a random jitter delay of 100-1000ms before response is sent
- VisaBalance has a random jitter delay of 2-5 seconds before response is sent
- Client splits number of requests in half between payment and balance

9.5% throughput increase in 10,000 case
15.9% throughput increase in 25,000 case!

One Backend Processor

5,000 requests
- Shortest Response: 3.283162s
- Longest Response: 6.223261s
- Average Response: 4.533328s

10,000 requests
- Shortest Response: 2.91652s
- Longest Response: 8.782683s
- **Average Response: 5.122519s**

25,000 requests
- Shortest Response: 3.01406s
- Longest Response: 17.11741s
- **Average Response: 9.17651s**

Three Backend Processors

5,000 requests
- Shortest Response: 3.028235s
- Longest Response: 6.359679s
- Average Response: 4.411553s

10,000 requests
- Shortest Response: 2.738824s
- Longest Response: 8.203394s
- **Average Response: 4.637534s**

25,000 requests
- Shortest Response: 2.692739s
- Longest Response: 14.15711s
- **Average Response: 7.725362s**