Adam Korczak

VIVA Finance

24 July 2024

1)

If the payment events came in via webhook, the new system would primarily involve three portions: the webhook endpoint, event processing, and a CSV being generated daily. Our webhook endpoint would be designed to receive POST requests from the payment provider. Since we know that these payment notification events would come in a fairly large batch in a short time regularly every day, we can opt to implement a message queue (Amazon SQS). My thought process behind this decision is that the message queue will better handle a large influx of events coming in at once. This is largely due to its FIFO order handling messages so that none are lost in a faster burst (1,000 - 10,000 in 30 minutes), along with its decoupling of producers and consumers being able to store the payment events which mitigates risks of losing data in the events that the receiving server suffers downtime (which likely is of the essence because it is working with financial and payment data).

At the end of the 30-minute period and retrieval of payment events, a separate job will aggregate all the data and generate the CSV file. The calculations and scheduling of the job can be handled by AWS Lambda functions and uploaded to a cloud storage to S3 with an additional backup for easy retrieval every day.

2)

a) Using the mentioned idea from question 1 with a message queue, we can update historical records for each borrower to a relational database. Specifically, past payroll dates and other relevant outstanding financial information. In using a relational database we can create schema with relationships that clearly define structures and common data/keys between tables.Furthermore, with a mass amount of data (100,000+ loans as mentioned) we can take advantage of a relational database's ability to normalize data and cut down on redundancies on common shared keys. We can use a batch processing system to periodically parse through all the data on payment timelines and analyze it with libraries such as Pandas or NumPy to pick up on patterns. Once historical patterns are found, we can tie pay rates (biweekly, monthly, etc) to specific job roles and make predictions on the frequency of payments.

b) To identify loans that didn't receive their expected payment on the payroll date we can use AWS Lambda to run serverless scheduled jobs that can first fetch the loans and expected payroll dates from the database mentioned in part a. From here we can query the database to check whether payment was received for each loan on the expected date. For a scheduled job with a large volume of payments to check, we can again implement a message queue (AWS SQS) to reliably process the data for a scheduled report creation. Another bonus of the message queue here would be the decoupling of producers and consumers. Depending on any bottlenecks that arise, we can choose to add either more producers or consumers independently of each other (effectively load balanced for our massive data processing by splitting the work between them).

c) Something to consider that may be an anomaly in the data on payment timelines from part a is the existence of borrowers that may not be paid on a regular schedule, most commonly contractors and freelancers. One idea to combat this irregularity in our predictions is to apply machine learning models against patterns specific to those jobs that are contracted/freelance (how often does xyz job get paid in over 1000s of different situations).