Crypto Address Tracker

Tuesday, March 1, 2016
1:03 PM

React front end: easy to implement and allows for reactive programming. Enables encapsulation and reuse of what will likely be simple components

Backend API
Java Spring Boot REST API
- Until we get to the realm of 200 requests a second we don't really need streaming architecture for MVP
- Simple to design; easily extendable into Spring Web Flux if streaming were required
- noSQL database; slight dependency on users and transactions, but

Database Design
noSQL due to simplicity of data being stored (user + addresses + transactions), scalability, and performance

```
User {
    Username: string,
    FirstName: string,
    LastName: string,
    cryptoAddresses: [
        CryptoAddress: {
        } ,
        …]
}

CryptoAddress {
    Type: SupportedCryptoEnum,
    Address: string,
    Transactions: [
        Transaction: {
        } ,
        …]
}

Transaction: {
    Source: string,
    Destination: string,
    SameAccount: boolean
    BitCoin amount: decimal,
    BitCoin value at time of transaction: decimal
}

SupportedCryptoEnum {
    BITCOIN("Bitcoin")
}
```

Difficult components to scale:
- REST API
  - Large amount of sync requests could lock up system due to blocking database operations

Solution: move to streaming architecture that intelligently detects what user's require synchronization

Database interactions

Currently querying across entire transaction database

Should implement streaming architecture to lighten load on superfluous database interactions

Could also keep note of high activity users to pre-emptively synchronize. This would allow tighter control over database interactions by classifying users as high activity denoting daily synchronizations as well as the whether or not we need to synchronize the user's transactions to begin with

Testing:

Spring integration tests for web app testing

React component testing

Should also integrate some sort of performance testing by hosting our application in a PERF environment and testing it under production style loads

Production Monitoring:

Smarts monitoring for API

Various log monitoring solutions like Dynatrace provide increased visibility into a variety of application and infrastructure metrics

Database monitoring can also be done via Dynatrace adding visibility into login attempts and long running queries

Algorithm:

Given a list of withdrawals and deposits detect the likely transfers amongst them

Example list of transactions:

[

('Tx_1', 'wallet_1', transaction_datetime, withdrawal/deposit(represented as in/out), amount of bitcoin), …]

```
MultiMap<String, transaction> transactions = new MultiValueMap();
ArrayList<Transaction> sameUserTransactions = new ArrayList<>();
//for each transaction in passed in list

//if hashmap is empty
//add a new entry to transactions hashmap using a key hashed from amount of bitcoin, transaction_datetime, and transaction type
//else
//if hashmap contains a key of the transaction with transaction type inversed within any transactions at the key
//check to make sure wallets are different
//add transaction from HashMap to sameUserTransactions first (and then remove) and then add the current transaction to retain ordering
//if they are the same then add a new entry at the hashed key
add
//else
//add a new entry to transactions hashmap using a key hashed from amount of bitcoin, transaction_datetime, and transaction type
```

Time Space Complexity

Time Complexity would be O(N) with hashmap contains checks taking O(1) provided the hashkey is optimal

Space Complexity would be at worse O(N) in the event that we don't find a likely match and we fill the hashmap in the process.

Evolve Algorithm for timestamp similarity

In our case the changes required would be simple. When building out the key for hashing we can drop the significant figures required to get the specificity we're looking for. Drawback here is we could only account for 10 minutes variances in transaction times exactly which may be too large of an allowance.

If we were looking for tighter bounds we would need to change how we're storing data and checking for duplications. First we would use a treemultimap instead of a multimap. Keys of the transactions would now consist only of bitcoin_amount at the multimap level and at the bucket level we'll use a TreeMap of our own custom key. This key will be made up of the timestamp of the transaction and leverage an overwritten equals to method that incorporates the range timestamp search we're looking for
//from there to determine potential matches we pull the bitcoin_amount bucket and perform a contains call to find transactions within our range.
//then we check the wallets again to make sure it's not the same wallet that performed both transactions before adding them both to the matches data structure based on transaction time.