F-Secure Homework Writeup

Peter Zhang

August 4, 2016

1 Introduction

The assignment is to write a "stream processor" that analyses a stream of events from an application. I implemented the required features using Javascript on Node.js to answer the questions listed in the instructions. The following third-party Node.js packages are used:

1. `bluebird` for promises.
2. `lodash` for clean functional programming primitives.
3. `line-reader` for reading the data file.

To install these dependencies, run:

```
npm install line-reader bluebird lodash
```

All the code can be found in the `analysis.js` file in the repository:

`https://github.com/peteraldaron/fsechw`.

2 Structure

I adapted a pipeline-like design to for data processing, whose procedure is listed below:

1. The program reads in the file line-by-line using line-reader.
2. Using the built-in `JSON.parse` function, the JSON line is converted into a JS object.
3. The source of the event, or the "product name", is identified. The program now stores data in product-specific fields.
4. Using the "`event_id`" field, duplicate events are identified, since every event can only have one unique event ID. This is implemented using a map (JS object).
5. Afterwards, the types of events are being counted for the specific product.

6. Then, the program accounts for the devices seen (unique devices), as well as the first launch count from unique devices for the specific product. This stage also records the first and last seen time of the device for longest session computation.
7. The program then moves onto counting the types of OS of client, as well as their geographical origins, which can be interesting for market penetration analysis.

Given the size of the data, it may be difficult to store the entirety of the raw data in memory. Therefore, I am only storing those data that are relevant to answering the questions posed, as well as other data of interest.

3 Findings

3.1 Answer to Questions

3.1.1 How many times has a particular product has been launched during this time period? How many first-time launches can you detect for this product?

Product	Launches	first-time launches by unique device
product-a	11772	1118
product-b	168360	35085
product-c	16431	296
product-d	223	38

3.1.2 Can you detect duplicate events? How?

Yes. Through the event's event_id field, which is unique for an individual event. The appearance of events are stored in a map (JS object) and any event that appears more than once is a duplicate.

3.1.3 Do you observe anything weird in timestamps?

A couple of things. Sometimes the event timestamp is smaller than event.time.(send or create)_timestamp. This is likely due to the possibility that the client's time is not synchronized with the network. Another abnormality is that some events do not have the timestamp field. In this case, I use the event.time.send_timestamp for time accounting instead.

3.1.4 Which device has longest 'activity time' max('first event latest event')

Product-a's device with the longest activity time is e99b76bc26efd88aa31ab573ff3d8e72e3ee4b91, and the time between the first event and the last event is 25643690662 milliseconds, or 296.8 days.

Product-b's device with the longest activity time is b3ebdd401e9aebc9c11475a010d864669cd63fc8, and the time between the first event and the last event is 24875526250 milliseconds, or 287.9 days.

Product-c's device with the longest activity time is 606df2e3df770fb16f1b10a2c23e88c2d75bd3e0, and the time between the first event and the last event is 25553708036 milliseconds, or 295.7 days.

Product-d's device with the longest activity time is 2eb0d958910d6351c403875022ab679c8feb6421, and the time between the first event and the last event is 7519999165 milliseconds, or 87.0 days.

3.1.5 How would you store the data so that further processing and/or analysis would be easy?

Since the data is already in JSON format, the easiest way to store the data is putting it into MongoDB, which stores the data in JSON format and also supports advanced querying, such as counting out-of-the-box.

3.1.6 If you should prepare a maintenance time for the processor, when would you do it in order to cause minimal impact to products?

To answer this question, we need to know when are the products least active, which can be acquired by making a histogram of the hours-of-day when all the events are being sent. The hour with the least amount of events is the time when the processor should be maintained.

If the maintenance for the processor is separate for each product, then the best timings are:
product-a: 2AM,
product-b: 5AM,
product-c: 3AM,
product-d: 2AM.

If the maintenance for the processor needs to be the same for all products, then the best timing is 5AM (15891 events).

It is easy to plot the histogram bars using the final results output from the program. One will just need a plotting library.

3.1.7 What kind of statistics do you think are interesting? How could you visualize this?

I also did statistics of the country-of-origin of these events for separate products, as well as the operating system/platform count of events. These two measurements are particularly interesting because, depending on the kind of services the products provide, it might be interesting to know where the users are coming from and what kinds of devices they use, to better serve the customers. Both geographical location and platform (operating system) information is available in the JSON data.

Geographical information can be visualized through plotting "pins" on a map, such as services provided by Google Maps. The operating system data, on the other hand, can be represented by pie charts.

3.2 Other Findings

A list of findings that I have noticed:

1. product-a has predominately mobile users, while Android users contribute to the most to the number of actions.
2. product-a has the most actions from Finland, but also from other parts of Europe, the US and Malaysia (MY).
3. product-b has about a quarter of events from mobile users, while Android users contribute to the most to the amount of actions.
4. product-b has the most actions from Finland (over 90%), and the rest from other parts of Europe and the US.
5. product-c has only mobile users, and Android users have the most number of actions.
6. product-c has the most actions from Finland (over 99.9%), and the rest is from the US.
7. product-d has only iOS users.
8. all users of product-d come from Finland.
9. product-b has the most number of unique devices (36558), followed by product-a (1477), product-c (296) and product-d (38).
10. There are quite a few windows machines installed on QEMU.

4 Future Work

If more time were available, I would really like to explore a bit more into the purchase data and the implications associated with the data. I have noticed a lot of voucher purchases that have failed due to voucher code error, and it could be interesting to find out what is going on in these situations. It would also be interesting to analyze the geographical and operating-systemic correlation with (in-app) purchases. Perhaps certain users from certain geographical regions and/or certain platforms have a particular purchasing behavior? Or if several users are “power users” that contributed the most of the events comparing to other users? These are questions that can be further explored and analyzed through the data.