SQL Query:

```
SELECT
     COUNT(1) ,
     device_model
FROM
     app.data
WHERE
     created_at BETWEEN '2018-01-01 00:00:00'
AND '2018-01-07 23:59:59'
AND device_model NOT IN (
     SELECT
          device_model
     FROM
          app.data
     WHERE
          created_at > '2018-01-07 23:59:59'
)
GROUP BY
     device_model
```

Architecture:

DAU	Requests/ Day(15sec frequency)	Requests/ Second(15sec frequency)	Requests/ Day(10min frequency)	Requests/ Day(10min frequency)
1mil	5.76 Billion	66,666 requests/ second	144 Million	1667 requests/ second
10mil	57.6 Billion	666,666 requests/ second	1.44 Billion	16,670 requests/ second
100mil	576 Billion	6,666,666 requests/second	14.4 Billion	166,700 requests/ second

That was just for me :(
If I was selecting a backend for collecting huge amounts of location data at a high-frequency, the first thing I’d begin doing is looking at benchmarks. After some brief research Cassandra really outperforms most databases when it comes to write speed. While picking a framework for the backend, I’d definitely find something asynchronous with an asynchronous persistence layer for database access. In my own experience async-pg, an asynchronous python library was able to reach somewhere close to 900,000 rows/second on read in our internal benchmarks. I think this is something people overlook often and is a huge bottleneck when it comes to scaling. Now the backend we scaled at my current company could also handle 4000 requests/

second but that seems rudimentary compared to these numbers. I remember reading something about a company scaling to over 30million requests/day on heroku. Not sure how they achieved that but more power to them. Auto-Scalers of all natures are a gift but I wouldn’t want to give up control for ease. AWS lambda is an absolute gift when it comes to efficiently processing information. If the data isn’t necessarily needed immediately, storing it as logs on a secondary database to be written later in an orderly queue also seems like a smart idea. Obviously for the requests some form of reverse proxy and/or micro-services load balancing would take care of that issue.

Android/Part-2 of that question:

I really appreciate this not being a boring coding test. While I surely wish I had more time to work on it, I’m gonna go ahead and apologize for some of the sloppy code. I’m on maybe 3 hours of sleep since our production database went down on Friday evening. That being said I don’t mind where the functionality of the app is at. If you click “Hello” your toast should be displayed and a job should be set to run at an interval of an hour. If you would like more frequent location updates click the HIGH_FREQUENCY button.

Timed Task -
The challenge here was to find a library that was backwards compatible. If the minVersion were 23, because of my familiarity with FirebaseJobDispatcher I would have used that. Instead I picked a library created by the people over at Evernote called Android-Job. This library allows the jobs to be run persistent through a reboot and is pretty responsive. The only downside was that the periodic job has a minimum interval of 10 minutes. To overcome this challenge I decided to create an FCMService to get push notifications every 20 seconds. Since I didn’t have the time to setup a backend to store the FCMids of the devices, I programmed the app to respond to something called topic alerts. I’m currently running these FCM notification requests from a script on my computer. I’ll have this script running for the next 48 hours. My primary motivation for using the FCMService was that for a brief period of 3 minutes after you app gets a notification, your app is whitelisted and given thread priority. Sadly google decided to kill a lot of the background execution functionality but this still allows for a lot of processing to

occur in that time. Ideally I wouldn’t use a library for something of this nature but cross compatibility here would mean implementing JobScheduler, AlarmManager and GCMNetworkManager.

Location -
I didn’t get as far into my solution as much as I would have liked to but the current version pulls your location from the google location-service. It makes a request for a location and momentarily a location is delivered to you. For me, it’s been unique more often than not but that isn’t necessarily supposed to be the case(Ideally we’d check the timestamps and only request a new location when necessary. This is also how we’d filter any bad location data.) To track location efficiently, one would have to leverage all that android has to offer. Ideally you’d want to limit your location gathering from google unless it was absolutely necessary. i.e when you’re connected to a Wifi-network we can assume you’re still in that place until the WifiStatusChanged listener is triggered and you alter the state of gathering. Sensors are also another great tool to use in approaching this porblem. Detecting motion and estimating distance moved isn’t the hardest thing to. This also allows for location updates based on changes on the client-side(Something google really wants us to do). Finally, having experimented with bluetooth a lot, I know bluetooth allows you access to all the BLE peripherals nearby. If you were to store these each time, You could eliminate a lot of your location requests because you’d already know where the user was with minimal processing optimizing battery usage.

Persistent through restart -
Unfortunately I couldn’t come up with a cross-version solution for this issue. If we had this coding test 2 months ago, it would have been very easy to do but since android made those changes to background execution, it’s virtually impossible to launch a service with your app in the background. If you were willing to display a foregroundNotification, this problem goes away and it’s just like pre April 25th. The solutions I had for this issue were two listeners for a RESTART and BOOT_COMPLETE intent. These usually re-launched the service, a solution that would work on most phones but didn’t work on my test device. Also to handle the scenario of when the user closes the app, on the service.onDestroy method, I broadcast a call to the same

receiver listening for those intents. I’d love to hear your thoughts on the background execution situation right now.

Local Storage-

Since I worked on a security application for Enterprise solutions, all of our data was encrypted on the device. Public/Private keys were used for all communications and that was fun to optimize. Through this, I learned that sharedPreferences was actually really efficient if you accessed and cached your objects appropriately. Although I’ve heard great things about the SQLite solution, I’m yet to try it or see any performance benchmarks on it. Internal cache files also seem relevant for certain scenarios. When it comes to huge chunks of location data, if instead of making calls you simply gathered data all day until the phone was charged, you could optimize battery usage. This is another reason I love the approach of sending push notifications as a ping constantly. The background processing doesn’t take up too much battery and you get to control all aspects of your location detection protocol. If you took all the data you have and added some Bluetooth, Wifi and Sensor data, you could write a programmatic AI to efficiently gather data on a case by case basis and with a little reinforced learning, every single user would be treated uniquely to satisfy whatever objective we want(accuracy/battery life)

Post-Script-

I would love any feedback you have on my approach and thoughts on what would might have done differently. My kotlin is pretty weak so please ignore the silly syntactical abominations you’ve been forced to read. I absolutely loved the challenge of going at the background execution problem once again. Sadly, I didn’t quite come out on top of it this time. That being said, given enough time, I think all problems have a solution. Once again, I’d love to absolutely thank you for this coding test. As much as I love my binary search trees, a real-world problem relevant to the position was a refreshing change. I think I could do some great work with you guys, not just in terms of Android but in terms of Machine Learning as well. I’m literally salivating thinking about all that data. If you could actually filter and present that data as value propositions for Companies, the possibilities are endless. Anyways, I look forward to hearing from you. Thanks for taking the time to read this. Apologies again for the run-on sentences. I hope you have a wonderful Sunday morning!