PocketData Benchmark

Naveen, Sankar, Saravanan, Sathish

What did we do last week?

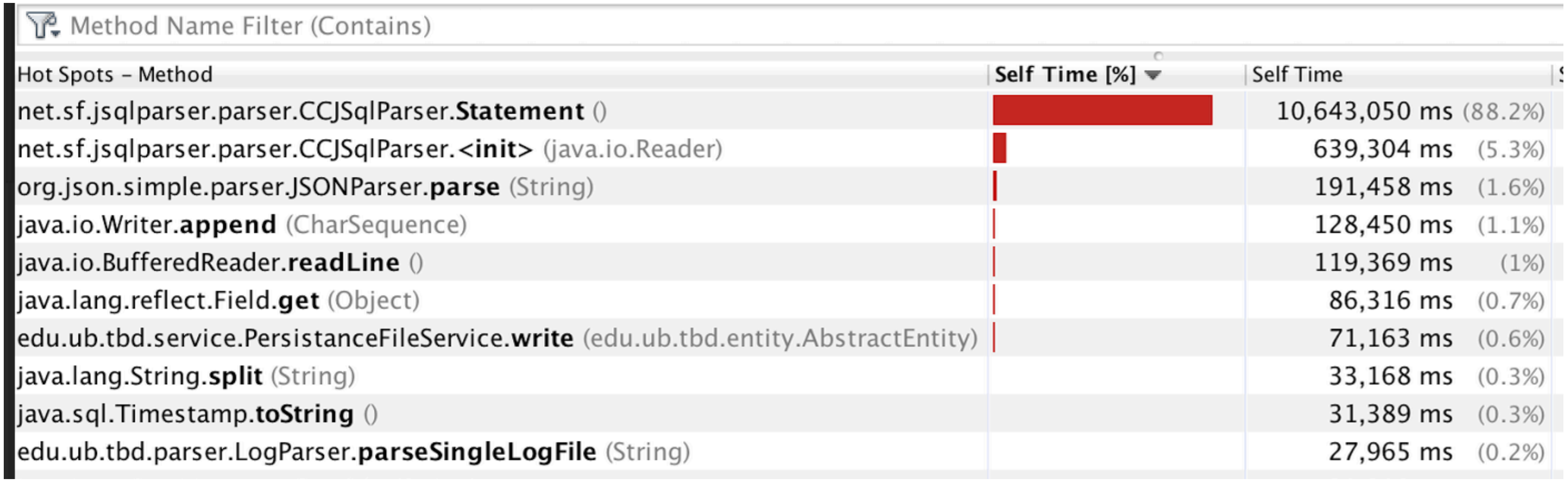
- Started extracting data from phone log.
- 93b46e40acbf1167ab0ad421b73761f16b74b562 1426486358667 1426486358667.0 2015-03-16 06:12:38.667574 13274 13274 I SQLite-Query-PhoneLab {"Counter":0,"LogFormat":"1.0",**"AppName":"Google Play Books"**,"Action":"**APP_NAME**","PackageName":"com.google.android.apps.books"}
- 93b46e40acbf1167ab0ad421b73761f16b74b562 1426477052477 1426477052477.1 2015-03-16 03:37:32.477583 13274 13274 I SQLite-Query-PhoneLab {"Counter":561,"LogFormat":"1.0","Time":243698,"Arguments":"null","Results":"SELECT value FROM ScheduledTaskProto ORDER BY sortingValue ASC, insertionOrder ASC","Action":"SELECT","Rows returned":3}
- {"Counter":27,"LogFormat":"1.0","Time":129375,"Results":"SQLiteProgram: INSERT INTO carriers(bearer,authtype,carrier_enabled,protocol,mmsproxy,roaming_protocol,numeric,mcc,type,mmsc,password,mvno_match_data,mvno_type,name,server,mnc,apn,user,mmsport) VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)","Action":"INSERT",**"Arguments(hashCoded)"**:"0,-1,1231,2343,-1630285132,2343,47743056,49679,-697368471,155249537,117478,0,0,1755004508,42,1537,1954370557,117478,1784,"}
- {"Counter":5,"LogFormat":"1.0","Time":149843,"Arguments":"null","Results":"**PRAGMA** table_info(name)","Action":"SELECT","Rows returned":0}

How did we parse it?

- Switched to Java from Python
  - Java is faster. Need to parse more than a GB
  - Better support with jsqlparser
- Extracted features like number of projected columns, table count etc.,
  - Made use of jsqlparser visitor pattern
- Modified jsqlparser to allow PRAGMA and INSERT or REPLACE queries (for future).

Challenges faced

- It took 3.5 hours to parse and extract log files.
- Future : To run parallel for users.



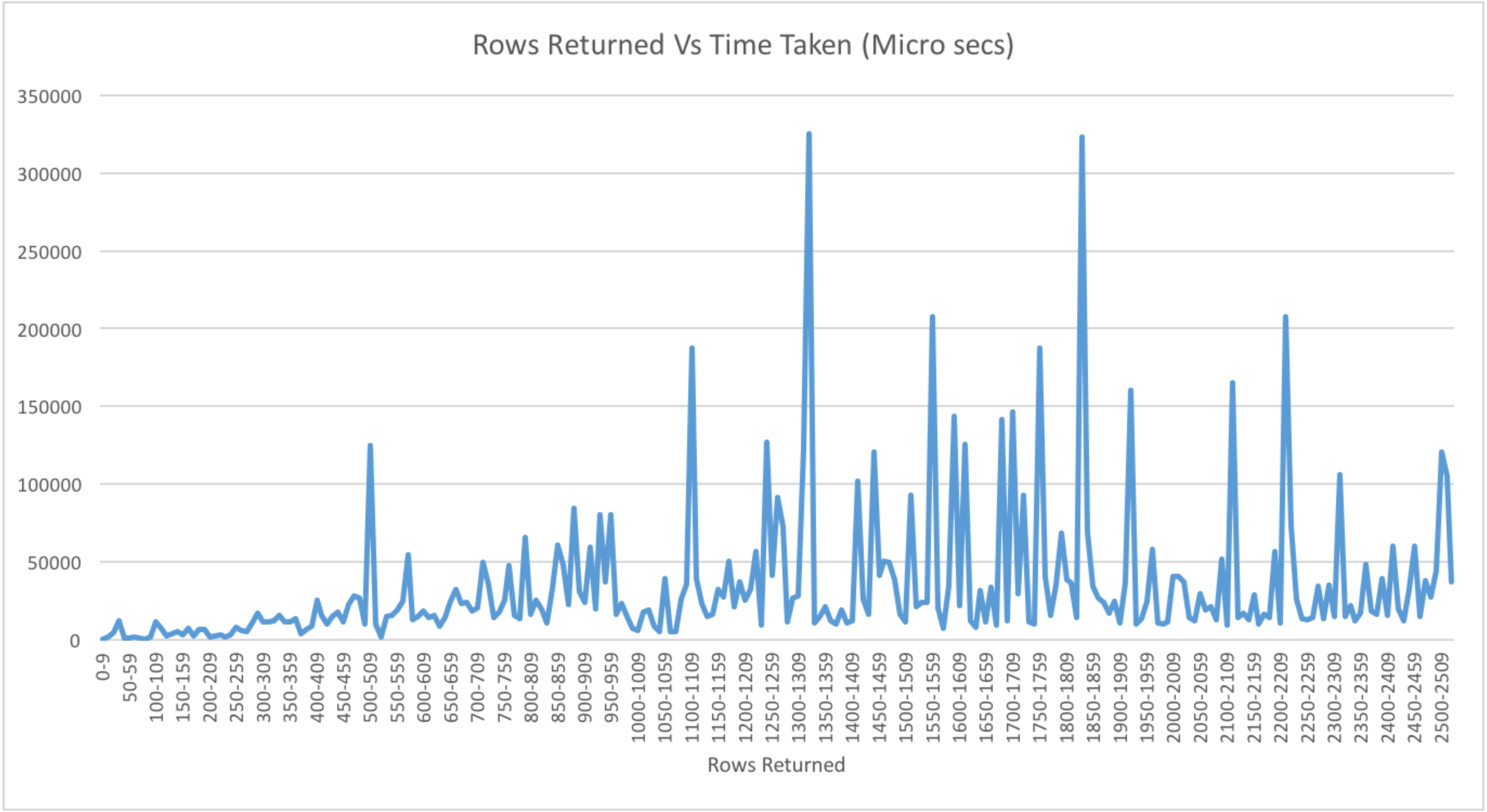
Method Name Filter (Contains)

Hot Spots – Method	Self Time [%]	Self Time
net.sf.jsqlparser.parser.CCJSqlParser.**Statement** ()		10,643,050 ms (88.2%)
net.sf.jsqlparser.parser.CCJSqlParser.**<init>** (java.io.Reader)		639,304 ms (5.3%)
org.json.simple.parser.JSONParser.**parse** (String)		191,458 ms (1.6%)
java.io.Writer.**append** (CharSequence)		128,450 ms (1.1%)
java.io.BufferedReader.**readLine** ()		119,369 ms (1%)
java.lang.reflect.Field.**get** (Object)		86,316 ms (0.7%)
edu.ub.tbd.service.PersistanceFileService.**write** (edu.ub.tbd.entity.AbstractEntity)		71,163 ms (0.6%)
java.lang.String.**split** (String)		33,168 ms (0.3%)
java.sql.Timestamp.**toString** ()		31,389 ms (0.3%)
edu.ub.tbd.parser.LogParser.**parseSingleLogFile** (String)		27,965 ms (0.2%)

Challenges [contd…]

- Unable to find app names for 32 million queries.
  - Something wrong with our logic
- Coming up with features without data available.

Results



Rows Returned Vs Time Taken (Micro secs)
350000
300000
250000
200000
150000
100000
50000
0
0-9
50-59
100-109
150-159
200-209
250-259
300-309
350-359
400-409
450-459
500-509
550-559
600-609
650-659
700-709
750-759
800-809
850-859
900-909
950-959
1000-1009
1050-1059
1100-1109
1150-1159
1200-1209
1250-1259
1300-1309
1350-1359
1400-1409
1450-1459
1500-1509
1550-1559
1600-1609
1650-1659
1700-1709
1750-1759
1800-1809
1850-1859
1900-1909
1950-1959
2000-2009
2050-2059
2100-2109
2150-2159
2200-2209
2250-2259
2300-2309
2350-2359
2400-2409
2450-2459
2500-2509
Rows Returned

Next week plan

- Play around with the extracted data to come up with interesting features.
- Extract more features from the log files.