Calculating Source Line Level Energy Information for Android Applications

Ding Li, Shuai Hao, William G.J. Halfond, Ramesh Govindan

Department of Computer Science
University of Southern California
Motivation: App Energy Consumption

- Battery power is limited on smart phones
- Developers lack fine-grained energy feedback
- Other approaches have critical limitations
Direct Measurement

Measure with power meters

- Does not sample fast enough
  - 10 KHz vs. 1GHz
- Cannot detect runtime events
  - Thread Switching
  - Garbage Collection (GC)

Does not provide source line level information
Other Approaches

• Cycle-level simulators
  – Cannot run market apps realistically

• Model on OS features
  – Does not provide source line level granularity

• Estimate energy with models and runtime information, for example, eLens
  – Building a good model is expensive
Road Map of Problem Solving

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expensive model</td>
<td>Measure instead of modeling</td>
</tr>
<tr>
<td>Insufficient sampling speed</td>
<td>Isolate source line energy with regression</td>
</tr>
<tr>
<td>Account for runtime events</td>
<td>Path information analysis</td>
</tr>
<tr>
<td>Capture runtime information</td>
<td>Efficient Instrumentation</td>
</tr>
</tbody>
</table>
Architecture of vLens

Runtime Measurement Phase

1. App Instrumenter
2. Power Measurement Platform
3. Path Adjuster
4. Analyzer
5. Annotator

Offline Analysis Phase

Annotated Code
Energy Report

Test Cases
App

Insufficient data?
Path Energy
1. App Instrumenter

Add probes to record runtime information

- Record the execution path
  - Efficient Path Profiling [Ball & Larus]
- The entry and exit time of APIs
  - Instrument APIs
2. Power Measurement Platform

Run apps and collect energy and path information

- **LEAP**: run the app
  - Atom N550 platform
  - Runs Android x86 3.2

- **DAQ**: measure the energy
  - Samples at a high frequency
  - Synchronizes time stamps to LEAP
  - Multiple components: CPU, RAM, WIFI, GPS
3. Path Adjuster

Handle problematic energy costs prior to regression analysis

• Calculate energy of API calls
• Assign tail energy to corresponding API calls
  – Tail Energy: energy caused by an API but expended after return of the API
• Distribute energy among concurrent internal threads
3. Path Adjuster: API Energy Calculation

Calculate and remove non-linear API cost

Sum up energy between entry and exit time stamps

\[
E_{t3,t4} = \frac{t2 - t1}{t4 - t3}
\]

Minimal sampling interval
3. Path Adjuster: Tail Energy

Assign tail energy to corresponding API

$$TE_{API_1} = \frac{T_s(API_2) - T_E(API_1)}{T_{tail}} E_{tail}$$
3. Path Adjuster: Internal Multi-thread

Split energy among concurrent internal threads

\[ E_{T1} = E_{t1,t2} + \frac{1}{2} E_{t2,t3} + \frac{1}{3} E_{t3,t4} \]
4. Analyzer

Calculate source line level energy information

- Distribute energy to bytecodes
  - Use robust regression techniques
- Eliminate GC and external thread switching
  - Residual-based outlier detection
4. Analyzer: Robust Regression

\[\sum_i \varphi(y_i - \sum_k x_{ik} \theta_k) x_{ij} = 0\]

Iterative

\[\varphi_k = \begin{cases} 
  x(\kappa \sigma - x^2)^2, & -\kappa \sigma < x < \kappa \sigma \\
  0, & \text{otherwise}
\end{cases}\]
5. Annotator
Evaluation

RQ1: What is the time cost of our approach?

RQ2: How accurately can our approach calculate the energy consumption of apps?
# Apps for Evaluation

All of our apps are from the Google Play app market

<table>
<thead>
<tr>
<th>App</th>
<th>#Classes</th>
<th>#Methods</th>
<th>#Bytecodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC Reader</td>
<td>590</td>
<td>4,923</td>
<td>293,910</td>
</tr>
<tr>
<td>Bubble Blaster II</td>
<td>932</td>
<td>6,060</td>
<td>398,437</td>
</tr>
<tr>
<td>Classic Alchemy</td>
<td>751</td>
<td>4,434</td>
<td>467,099</td>
</tr>
<tr>
<td>Skyfire</td>
<td>684</td>
<td>3,976</td>
<td>274,196</td>
</tr>
<tr>
<td>Textgram</td>
<td>632</td>
<td>5,315</td>
<td>244,940</td>
</tr>
</tbody>
</table>
RQ1: Analysis Time Cost

- **Pre-execution**: Efficient path profiling instrumentation time: 7.5 min
- **In-execution**: Runtime overhead: 4%
- **Post-execution**: Path & regression analysis time: 1.97 min
RQ2: Accuracy

1. API energy measurements
2. Bytecode energy distribution
   – Application level
   – Path level
3. GC and thread switching detection

DAQ cannot get source line level information
Accuracy of the API Energy Measurements

The average difference is 9%

24 APIs represent 70% of total API energy
Accuracy of Bytecode Energy Distribution

• $R^2$: Path level
  – Well used statistical metric
  – Describe the quality of regression model

• AEE: Application level accuracy
  – Difference between the result from our model to the measured ground truth
Accuracy of Bytecode Energy Distribution

vLens is accurate on both application level and path level

<table>
<thead>
<tr>
<th>App</th>
<th>Accuracy</th>
<th>R²</th>
<th>AEE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBC Reader</td>
<td></td>
<td>0.94</td>
<td>6.5</td>
</tr>
<tr>
<td>Bubble Blaster II</td>
<td></td>
<td>0.90</td>
<td>8.6</td>
</tr>
<tr>
<td>Classic Alchemy</td>
<td></td>
<td>0.93</td>
<td>3.4</td>
</tr>
<tr>
<td>Skyfire</td>
<td></td>
<td>0.99</td>
<td>4.8</td>
</tr>
<tr>
<td>Textgram</td>
<td></td>
<td>0.92</td>
<td>6.3</td>
</tr>
</tbody>
</table>
Accuracy of Outlier Detection

• Detect outliers caused by GC and external thread switching
• Cannot capture real GC and external thread switching
• Seed GC and thread twitching at random positions per app
  – 200 `System.gc()`
  – 200 `Thread.start()` and `Thread.join()`

All seeded events are identified by outlier detection
vLens: Calculating Source Line Level Energy Consumption

- Combines program analysis and statistical analysis
- High accuracy
  - 9% error
- High granularity
  - On source line level
- Low overhead
Thank you
Definition of AEE and $R^2$

$$AEE = \frac{\left| \sum \hat{y}_i - \sum y_i \right|}{\sum y_i}$$

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$
Bytecode Profiling

\[ \text{add} = \text{load a} \quad \text{load a} \]
\[ \text{load b} \quad \text{load b} \]
\[ \text{add} \]
\[ \text{pop} \quad \text{pop} \]
vLens vs. eLens

• vLens:
  – Have energy measurement equipment
  – No model
  – Needs to handle run time events, such as GC, external thread switching, concurrent app thread

• eLens:
  – No energy measurement equipment
  – Needs to build model
  – Does not need to handle run time events.