Energy-Aware, Security-Conscious Code Offloading for the Mobile Cloud

Kirill Mechitov, Atul Sandur, and Gul Agha

October 12, 2016
Background: Mobile Cloud Computing

- Cloud Computing
  - Access to virtually unlimited, elastic computing and storage
  - Efficient, scalable, affordable

- Cloud + Mobile
  - Enable new functionality
  - Remove mobile device limitations
  - Improve performance
  - Reduce energy consumption

- How to implement?
  - Offloading
IMCM framework

- Application Component Distribution
- Elasticity Manager
- Application Target Goal
- Org/App/User Policy

- System Monitor
  - Application actions
  - Network parameters
  - User context
  - Application profiling
  - Energy estimator

- System Properties

- Offloading Plan
  - Target goal
  - Profiled exec
  - Profiled comm

- Decision Maker

- Policy Manager
  - Application Policy
  - Access Restrictions
  - User preferences

- Max app performance
- Min mobile energy consumption
- Min cloud cost
- Min network data usage

Org/App/User Policy

Application Target Goal

Elasticity Manager

Application Component Distribution
IMCM design and implementation

- Actor programming model
  - Natural Concurrency
  - Decentralization
  - No data races
  - Elasticity
  - Location transparency
    -> Transparent migration

- Implementation: SALSA
  - Full actor semantics
  - Lightweight actors
  - Migration support
  - Portability (Java-based)
IMCM framework

Application Component

Distribution

Elasticity Manager

- Max app performance
- Min mobile energy consumption
- Min cloud cost
- Min network data usage

Application Target Goal

Org/App/User Policy

- Application Policy
- Access Restrictions
- User preferences

System Monitor

Application actions
Network parameters
User context
Application profiling
Energy estimator

System Properties

Elasticity Manager

Offloading Plan

- Target goal
- Profiled exec
- Profiled comm

Policy Manager

Decision Maker

Application Component Distribution

Org/App/User Policy

Application Target Goal

System Monitor
IMCM system monitor

- **What to monitor?**
  - Application actions
    - At the level of actor primitives (create, migrate, send/receive)
  - System actions
    - Offloading decisions, costs
  - System/environment conditions
    - Performance, energy, connection speed, resource availability

- **When to notify?**
  - Changes in monitored parameters
  - Changes in org/user/system policies
  - Violation of existing policies
Application actions

- The actor system is involved when:
  - Actor sends a message
  - Message arrives
  - Message handler is executed
  - Message handler finishes execution
  - Actor creation initiated (local)
  - Actor creation completed (local or remote)
  - Migration is initiated (local)
  - Migration request received (remote)
  - Migration completed (remote)

- Can intervene and control app behavior at these points *without modifying application code*
The actor system is involved in:

- Offloading decision (when, where, and which actors)
- System-initiated migration
- System-initiated actor creation
Metadata: actor attributes

- Describe properties of actors
  - Creating user
  - Creating actor
  - Current location
  - Etc.

- Two types of attributes:
  - Static – do not change at runtime
    - E.g., creating actor
  - Dynamic – can change
    - E.g., location
Metadata: dynamically updated structures

- Message send graph
  - “Call graph”

- Actor creation graph
  - For permission/restriction inheritance

- Actor name knowledge set
  - Who can send messages to this actor
  - Actor semantics: actor names cannot be guessed
IMCM framework

Application Component Distribution

Elasticity Manager

Application Target Goal

Org/App/User Policy

System Monitor

Policy Manager

Decision Maker

Application actions
Network parameters
User context
Application profiling
Energy estimator

System Properties

• Max app performance
• Min mobile energy consumption
• Min cloud cost
• Min network data usage

Application Policy
Access Restrictions
User preferences

Offloading Plan
Target goal
Profiled exec
Profiled comm

Application Component Distribution
Action control system

- Requirements
  - Fine granularity
  - Support policies defined by organization, developer, and end-user
  - Covers all possible app and system actions

- Action control system based on:
  - The requested action
  - The specified policy
  - Attributes of the requester
  - Attributes of the recipient
  - Attributes of the resource
The authorization model includes the following parties:
- Owner organization (cloud space and/or data)
- Developers of the application that uses cloud resources
- End-users of the application

Who defines authorization policy?
- The organization that owns data and cloud resources (organization-wide policy)
- The application developer can further restrict these organization-wide accesses (application-specific policy)
- The application user (user-specific policy)
Policy definition

- Authorization policy
  - Hard (immutable)
    - E.g., cloud provider policy
  - Soft (customizable)
    - E.g., user-defined policy

- Grammar to define rules
  - Attribute based
  - Static & dynamic binding

- Rule structure:
  - \(<Subject, Object, Action, Sign, Type>\)
Implementation: action monitoring

- Rules from multiple applicable policies are merged
  - Organization, developer, end-user policies
  - Hard policy rules take precedence over soft policy rules

- Rules checked for every invocation of each app and system action, so must be efficient
  - Sort rules to prioritize early DENY decision
  - Sort rules to prioritize static-only checks
  - Cache lookups to dramatically speed up most frequent actions
  - Invalidate cache on dynamic attribute change
  - Use dynamic metadata to speed up lookups
Implementation: heuristics

- In typical applications, most of the time is spent in the message handler.

- Actor migration is quite infrequent.

- Local actor creation is much more frequent than remote actor creation.

- Local message sends are much more frequent than remote message sends.
Example: satellite ISP customer privacy policy
Example policy

- Sample private cloud rules for restricted code and data access
  - E.g., no send, migration from private to public cloud

<table>
<thead>
<tr>
<th>Rule Number</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>ActorSystem: {Name:ActorSysPrivate1, Static (URL:174.123.78.456, Port:1362)}</td>
</tr>
<tr>
<td>2.</td>
<td>Actor: {Name:ActorPrivateGate, Static (Reference: akka.tcp://app@174.123.78.456/privateGateway, ActorSystem:&quot;ActorSysPrivate1&quot;)}</td>
</tr>
<tr>
<td>3.</td>
<td>Actor: {Name:ActorPrivateVisaDB, Static (Reference: akka.tcp://app@174.123.78.456/actor1, ActorSystem:ActorSysPrivate1)}</td>
</tr>
<tr>
<td>4.</td>
<td>Actor: {Name:ActorPrivateResidentDB, Static (Reference: akka.tcp://app@174.123.78.456/actor2, ActorSystem:ActorSysPrivate1)}</td>
</tr>
<tr>
<td>5.</td>
<td>Actor: {Name:ActorPrivateCitizenDB, Static (Reference: akka.tcp://app@174.123.78.456/actor3, ActorSystem:ActorSysPrivate1)}</td>
</tr>
<tr>
<td>6.</td>
<td>Actor: {Name:ActorPrivateVisaProcessor, Static (Reference: akka.tcp://app@174.123.78.456/actor4, ActorSystem:ActorSysPrivate1)}</td>
</tr>
<tr>
<td>7.</td>
<td>Actor: {Name:ActorPrivateResidentProcessor, Static (Reference: akka.tcp://app@174.123.78.456/actor5, ActorSystem:ActorSysPrivate1)}</td>
</tr>
<tr>
<td>8.</td>
<td>Actor: {Name:ActorPrivateCitizenProcessor, Static (Reference: akka.tcp://app@174.123.78.456/actor6, ActorSystem:ActorSysPrivate1)}</td>
</tr>
<tr>
<td>10.</td>
<td>AnonymousActorSystems: {Name:Other-ActorSys-Private, URL:174.123.78.456, Creation:FORBIDDEN}</td>
</tr>
<tr>
<td>13.</td>
<td>Rule: {Name:Private-Rule-3, Subject (Actor:ActorPrivateGate), Object (ALL), Actions: SEND-TO, RECEIVE-FROM, Permission: ALLOWED}</td>
</tr>
<tr>
<td>16.</td>
<td>ActorSystem: {Name:ActorSysPrivate2, Static (URL:174.123.78.456, Port:1369)}</td>
</tr>
</tbody>
</table>
Offloading decision

- Which components to offload, and where?

- Inputs:
  - Platform characteristics
    - available processing power, bandwidth, memory, etc.
  - Application component (actor) characteristics
    - processor, bandwidth, I/O, etc.
  - Current system configuration, resource use

- Outputs:
  - App partitioning, actor placement, actor migration
Offloading decision: energy use

- Actor-level monitoring
- Track events (actor message executions)
- Attribute overall energy use to particular actors
- Heuristic: ignore low energy use actors
Energy use estimation – simple model

- Example: face recognition app

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Voltage Drop (V)</th>
<th>Energy (mJ)</th>
<th>No. Face Detector</th>
<th>No. Feature Extractor</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0.0324</td>
<td>58350</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>0.0313</td>
<td>56340</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>161</td>
<td>0.0187</td>
<td>33700</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Component Energy Consumption

- Component | Energy Consumption (mJ)
- Face Detector | 2668
- Feature extractor | 2019
Problem: noisy data

- Hard to attribute macro energy measurements to individual actions
Carat energy monitoring framework

- The crowd
  - Instrumentation data
  - Actions and reports

- The cloud
  - AWS & Spark

- Big data
  - Raw and derived data
  - Statistical analysis
Crowd-sourced energy profiling

![Energy Profiling Chart](chart.png)

- **Battery Life Gain (%)**
- **CPU usage level**: Low, Medium, High
- **Categories**: Stationary, Movement, Screen Auto, Screen Manual, Signal Bad, Signal Average, Signal Good, Signal Excellent, Under 30C, Over 30C
CARAT-based monitoring architecture
Benchmark results

- Running the same code on more powerful HW
- Running some components in parallel
- Keeping in mind the cost of offloading

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Application Characteristic</th>
<th>Raw Speedup</th>
<th>Offload Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NQueen</td>
<td>intensive</td>
<td>73</td>
<td>56</td>
</tr>
<tr>
<td>Image</td>
<td>intensive limited limited</td>
<td>91</td>
<td>44</td>
</tr>
<tr>
<td>Trap</td>
<td>intensive limited</td>
<td>30</td>
<td>21</td>
</tr>
<tr>
<td>Virus</td>
<td>-</td>
<td>28</td>
<td>21</td>
</tr>
<tr>
<td>Rotate</td>
<td>-</td>
<td>28</td>
<td>9</td>
</tr>
<tr>
<td>ExSort</td>
<td>intensive intensive</td>
<td>46</td>
<td>36</td>
</tr>
<tr>
<td>Heat1</td>
<td>limited medium</td>
<td>31</td>
<td>29</td>
</tr>
<tr>
<td>Heat2</td>
<td>limited high</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>
IMCM framework overhead

Image Processing: Overhead of Elasticity Manager running in the background

- Image (Sequential Mode at Remote with 1 remote workers)
- Image (Parallel Mode at Remote with 8 remote workers)
- Image (Average Overhead)
IMCM framework summary

- Bridge the gap between mobile application and cloud computing
- Separate application logic from component distribution decisions
- Allow dynamic distribution of components based on run-time changes
- Support policy-based definition of security/privacy constraints
Future work

- Fine-grained energy profiling
- Energy-aware offloading optimization
- Use energy monitoring in policy-based monitoring/enforcement
Questions?