Dynamic Fine-Grained Code Offloading in Mobile Cloud Applications

Kirill Mechitov, Atul Sandur, and Gul Agha

April 6, 2016
Outline

- Offloading optimization in the IMCM Framework
  - Elasticity Manager component
  - Mobile-side profiling

- Choosing a good initial configuration for the deployment
  - Model checking tool for improving “cold start” performance
Background: Mobile Cloud Computing (MCC)

- **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

Credit: cloudcomputingdoc.com
Background: code offloading

- **Full VM emulation**
  - Run mobile device emulator in the cloud
  - Universal solution, but expensive

- **Application partitioning**
  - Run some components on the mobile device and some in the cloud
  - How to partition?
    - Let the app developer do it
    - Let the system do it, *statically*
    - Let the system do it, *dynamically*
<table>
<thead>
<tr>
<th>Year</th>
<th>System Name</th>
<th>Goal</th>
<th>Offloading Decision</th>
<th>Partition Level</th>
<th>Parallel</th>
<th>Policy-based Security/Privacy</th>
<th>Manual Work</th>
<th>No. Cloud spaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>MAUI</td>
<td>Mobile Energy Saving</td>
<td>Dynamic</td>
<td>Method</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>2011</td>
<td>CloneCloud</td>
<td>Mobile Energy Saving = Performance Improvement</td>
<td>Static</td>
<td>Method</td>
<td>Pseudo</td>
<td>No</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>2012</td>
<td>ThinkAir</td>
<td>Mobile Energy Saving = Performance Improvement</td>
<td>Dynamic</td>
<td>Method</td>
<td>Pseudo</td>
<td>No</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>2012</td>
<td>Cloud OS (COS)</td>
<td>Load Balancing for Cloud space</td>
<td>Dynamic</td>
<td>Actor</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Can be Many</td>
</tr>
<tr>
<td>2015</td>
<td>IMCM</td>
<td>Mobile Energy Saving, Performance Improvement, Policy-based Distribution</td>
<td>Dynamic</td>
<td>Actor</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Many</td>
</tr>
</tbody>
</table>
IMCM: Illinois Mobile Cloud Manager

- **Code offloading:**
  - Automatic
  - Dynamic
  - Fine-grained
  - Parallel

- **Supports:**
  - Hybrid cloud with multiple cloud spaces

- **Provides:**
  - Policy-based control by cloud provider, app developer, user
Actor model

- **Programming model**
  - Natural concurrency
  - Decentralization
  - No shared state
  - Scalability
  - Location transparency
    -> Transparent migration

- **Implementation: SALSA**
  - Full actor semantics
  - Lightweight actors
  - Migration support
  - Portability (Java-based)
module demo;

behavior HelloWorld {

    /* The act(...) message handler is invoked automatically and is used to bootstrap salsa programs. */
    void act( String[] argv ) {

        standardOutput<-print( "Hello" ) @

        standardOutput<-print( "World!" );

    }
}
IMCM framework

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

Application Target Goal
- Application Policy
- Access Restrictions
- User preferences

Org/App/User Policy

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

Elasticity Manager

Policy Manager
- Target goal
- Profiled exec
- Profiled comm

Offloading Plan

Decision Maker

Application Component Distribution
Today: Optimization – 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

Decision Maker

Application Component Distribution

Policy Manager
Example: face recognition app

- Components with different computation, bandwidth, energy characteristics
- Different partitioning schemes depending on policy-based restrictions on certain components/data
Example: Tactical Cloudlet (CMU)

- Tactical Cloudlets: Moving Cloud Computing to the Edge

Credit: Software Engineering Institute, CMU
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>I/O</th>
<th>Raw Speedup</th>
<th>Offload Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Comp.</td>
<td>Comm.</td>
<td>read</td>
<td>write</td>
</tr>
<tr>
<td>NQueen</td>
<td>intensive</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Image</td>
<td>intensive</td>
<td>limited</td>
<td>limited</td>
<td>-</td>
</tr>
<tr>
<td>Trap</td>
<td>intensive</td>
<td>limited</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Virus</td>
<td>-</td>
<td>-</td>
<td>intensive</td>
<td>-</td>
</tr>
<tr>
<td>Rotate</td>
<td>-</td>
<td>-</td>
<td>intensive</td>
<td>intensive</td>
</tr>
<tr>
<td>ExSort</td>
<td>intensive</td>
<td>-</td>
<td>intensive</td>
<td>intensive</td>
</tr>
<tr>
<td>Heat1</td>
<td>limited</td>
<td>medium</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Heat2</td>
<td>limited</td>
<td>high</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Face recognition app speedup

Image Processing: Sequential/Parallel Local/Remote Execution for different No. of pictures

- Blue line: Image Mode=Sequential Exec. at=Mobile (Base Case)
- Red line: Image Mode=Parallel Exec. at=Mobile
- Green line: Image Mode=Sequential Exec. at=Remote
- Purple line: Image Mode=Parallel Exec. at=Remote

Speedup (X)

Problem Size (No. of Images)

1. 6.85
2. 12.03
4. 20.04
8. 27.31
16. 34.89
32. 42.13
64. 50.48

1.00
0.94
1.00
0.96
1.00
0.96
1.00
0.97

1.00
1.05
1.05
1.00
0.96
0.96
1.00
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)

Elasticity Manager Overhead (Speedup X%) vs. Problem Size (No. of Images)
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: partitioning
Offloading decision: bandwidth

\[
\frac{w}{S_m} > \frac{d_i}{B} + \frac{w}{S_s}
\]

\[
w \cdot \left( \frac{1}{S_m} - \frac{1}{S_s} \right) > \frac{d_i}{B}
\]

Offloading decision depends on Bandwidth (B)

- Never offload
- Always offload
Offloading decision: energy use

- Extend mobile profiling framework CARAT to actor-level monitoring
- Track events (actor message executions)
- Attribute overall energy use to particular components
- Heuristic: ignore low energy use actors
CARAT-based monitoring architecture
Potential problem

- Similar adaptive system optimization problems sometimes tend to get stuck at local optima:
Potential problem

- Particularly troublesome when:
  - Want to find best option from a given current configuration
  - Transition costs can be large

- Even if we know the global optimum, it may be too expensive to get to it from the current configuration

- I.e., initial starting configuration matters quite a bit
  - How to start in a good config?
Two cases

1. “Big Data” scenario

2. “Cold Start” scenario
Big Data

- Lots of profiling data of different configurations on different hardware available

- Good coverage of possible configuration space
  - With some random perturbation

- Can start at/near global optimum
  - At least most of the time
Cold Start

- No data
  - New application
  - New hardware
  - Unique or unusual setting/environment

- How to get profiling data for a good sampling of possible configurations?
Some related work [SPIN 2016]

A Model Checking Tool for Schedulability Analysis of Distributed Real-Time Sensor Network Applications

- Joint work with Ehsan Khamespanah (U. of Tehran) and Marjan Sirjani (Reykjavik University)
The Illinois Structural Health Monitoring Project pioneers the use of densely deployed smart wireless sensors for long-term continuous monitoring of civil infrastructure.

Six wireless sensors deployed on the Siebel Center central staircase use ambient vibration — such as that caused by people walking up and down the stairs — to measure changes in characteristic vibration frequencies, which can be used to detect and pinpoint structural damage (none yet!).

A larger-scale version of this system, with more than 100 sensor nodes, has been deployed for monitoring of long-span bridges, and the software developed by the project is being used by 75 institutions in 15 countries.
Continuous real-time sensing app

- Continuously collect *synchronized* sensor data from multiple sensors
- Send to gateway node with low latency
The problem

- Sensing (left) and radio transmission (right) have their own deadlines, and a dependency relationship.

![Diagram showing the process of sensing and radio transmission with deadlines and dependency relationship.](image)
The main idea

- Use model checking tool to check schedulability
- Explore configuration parameter space on the boundary of schedulability
- For a given set of initial parameters, find optimal configuration(s)
  - I.e., find bounding surface in the parameter space
What is model checking?

- Automated ("push-button") verification technique

- Given:
  - System model
  - Logical formula/specification to be verified

- Do:
  - Exhaustive search of the state space of the model

- Result:
  - Formula holds, or
  - A counter-example
Why model-checking?

Pros:
- No/little formal methods knowledge needed
- Simple “push-button” testing
- Can provide counter-examples for debugging

Cons:
- Model may not be faithful to implementation
- Prone to state explosion problem
Timed Rebeca is an actor-based modeling language with bounded floating time transition system (BFTTS) semantics

- Can reduce size of state space and dramatically increase model checking performance for timed models

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size</th>
<th>Using BFTTS</th>
<th>Using timed automata</th>
<th>Using McErlang</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#States</td>
<td>Time</td>
<td>#States</td>
</tr>
<tr>
<td>Ticket Service</td>
<td>1 customer</td>
<td>8</td>
<td>&lt;1 s</td>
<td>801</td>
</tr>
<tr>
<td></td>
<td>2 customers</td>
<td>51</td>
<td>&lt;1 s</td>
<td>19M</td>
</tr>
<tr>
<td></td>
<td>3 customers</td>
<td>280</td>
<td>&lt;1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>4 customers</td>
<td>1.63K</td>
<td>&lt;1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>5 customers</td>
<td>11K</td>
<td>&lt;1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>6 customers</td>
<td>83K</td>
<td>2 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>7 customers</td>
<td>709K</td>
<td>3 min</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>8 customers</td>
<td>6.8M</td>
<td>9.7 hours</td>
<td>–</td>
</tr>
<tr>
<td>Sensor network</td>
<td>1 sensor</td>
<td>183</td>
<td>&lt;1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>2 sensors</td>
<td>2.4K</td>
<td>&lt;1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>3 sensors</td>
<td>33.6K</td>
<td>1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>4 sensors</td>
<td>588K</td>
<td>13 s</td>
<td>–</td>
</tr>
<tr>
<td>Slotted ALOHA protocol</td>
<td>1 interface</td>
<td>68</td>
<td>&lt;1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>2 interfaces</td>
<td>750</td>
<td>&lt;1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>3 interfaces</td>
<td>7.84K</td>
<td>1 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>4 interfaces</td>
<td>45.7K</td>
<td>6 s</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>5 interfaces</td>
<td>331K</td>
<td>64 s</td>
<td>–</td>
</tr>
</tbody>
</table>
Example: simple TR model

Ticket service

```java
1 reactiveclass TicketService {
2    knownrebecs {
3        Agent a;
4    }
5    statevars {
6        int issueDelay;
7    }
8    msgsrv initial(int myDelay) {
9        issueDelay = myDelay;
10    }
11    msgsrv requestTicket() {
12        delay(issueDelay);
13        a.ticketIssued(1);
14    }
15 }
16
17 reactiveclass Agent {
18    knownrebecs {
19        TicketService ts;
20        Customer c;
21    }
22    msgsrv requestTicket() {
23        ts.requestTicket()
24        deadline(5);
25    }
26    msgsrv ticketIssued(byte id) {
27        c.ticketIssued(id);
28    }
29 }
30
31 reactiveclass Customer {
32    knownrebecs {
33        Agent a;
34    }
35    msgsrv initial() {
36        self.try();
37    }
38    msgsrv try() {
39        a.ticketIssued(byte id) {
40            self.try() after(30);
41    }
42    }
43 }
44
45 main {
46    Agent a(ts, c):();
47    TicketService ts(a):(3);
48    Customer c(a):();
49 }
50 ```
Example: sensor network optimization

- Explore parameter space to find optimal configurations
Example: protocol comparison/evaluation

(a) TDMA, Sensor task delay is 5ms
(b) B-MAC, Sensor task delay is 5ms
(c) TDMA, Sensor task delay is 10ms
(d) B-MAC, Sensor task delay is 10ms
Application to MCC

- Instead of schedulability, check satisfaction of performance and energy use properties

- E.g.:
  - Optimal parameter configuration under iso-performance or iso-energy
  - Component and parameter selection to meet SLA guarantees

- Use app model for rapid prototyping & estimating energy/performance of actors
  - Start in better initial configuration that is likely to be at or close to global optimum
Work in progress

- **Model-checking:**
  - Basic mobile hybrid cloud and application model
  - Define properties of interest in FTTS
  - Create tool to automatically generate IMCM-compatible configuration schemas

- **Monitoring:**
  - Fully integrate CARAT energy monitoring framework into IMCM
The end

- Questions?