October 22, 2014

IMCM: A Flexible Fine-Grained Adaptive Framework for Parallel Mobile Hybrid Cloud Applications

Reza Shiftehfar
PhD. Student and Research Assistant in Computer Science
E-mail: ss@illinois.edu

Kirill Mechitov
PhD. Student and Post-Doctoral in Computer Science
E-mail: mechitov@illinois.edu

Gul Agha
Professor in Computer Science
E-mail: agha@illinois.edu

Outline:

- Introduction
- Motivation
- Objective
- Related Works
- Approach
- Conclusion

Introduction: An Adaptive Framework for Mobile-Cloud Apps

- Mobile devices:
  - Ubiquitous and improving very fast
  - still constrained by limited resources:
    1. Weaker Hardware: (CPU, Memory, Storage)
    2. Restricted Network Access: (lower bandwidth, higher latency, cellular data usage charges/cap)
    3. Limited On-board Energy: (Battery-power restrictions)
- Applications
  - are becoming more and more complex & resource-demanding
  - There is a gap between the two that will remain even in future

Introduction: Cloud Computing:

- allows efficient storage and processing of very large datasets
  - Efficient both in terms of time(speed) and cost(price)
  - provides elastic on-demand access to unlimited resources at affordable price
  - Serves as the center of the data storage and processing of many organizations
  - dominant term for new computing paradigm
- Cloud has the potential to help with mobile restrictions:
  1. Remove constraints on mobile device hardware
  2. Reduce mobile energy consumption
  3. How to do this? Use code-offloading
**Introduction:** An Adaptive Prog. Framework for Mobile-Cloud Apps

- **Code-Offloading:**
  - Sending computation to more resource-full machines and bringing back the results
  - How to make code offloading happen:
    1. Ask programmers to rewrite the code
      - **Pros:** Fine-grained
      - **Cons:** Too much additional work, continuous maintenance
    2. Use full process or full VM migration
      - **Pros:** No additional tasks for programmers
      - **Cons:** Very coarse-grained, expensive
    3. Combine the two

**References:** [1]

---

**Motivation:** Mobile-Cloud Computing (MCC) is challenging because of:

1. **Applications:**
   - Have different requirements
   - Behave differently at different times
   - Have different applications goals:
     1. Maximizing the performance
     2. Minimizing network data usage
     3. Minimizing mobile energy consumption
2. **Mobile devices:**
   - Different hardware capabilities
   - Different quality of network access and limitations
   - Different amount of accessible energy to use

---

**Motivation:** Mobile-Cloud Computing (MCC) is challenging because of: (cont.)

3. **Security and Privacy:**
   - **Current Cloud solutions:**
     1. Typically provide very limited security protection
     2. Traditionally Assumed as a trusted environment
   - This can be true when:
     1. Cloud is kept separate from the outside world
     2. Only used by authorized people
   - **Public cloud:**
     - Owned and operated by 3rd party companies
     - Risky for moving sensitive or confidential data or computation
Motivation: An Adaptive Prog. Framework for Mobile-Cloud Apps

- **Motivation**
  - Mobile-Cloud Computing (MCC) is challenging because of: (cont.)
    4. **Users:**
      - have different expectations at different time
      - might have further concerns about the privacy of their data.
  - **Mobile Applications** are used for different target goals in different environmental conditions with dynamic requirements and expectations

Objective: An Adaptive Prog. Framework for Mobile-Cloud Apps

- Bridge the gap between mobile application development, cloud computing and dynamic adaptive code-offloading
- Separate application logic from application component configuration and distribution plan
- Allow dynamic distribution of application components in response to run-time environmental changes while preserving required constraints
- Provide Policy-driven offloading adjustment for Cost, Privacy, Quality

Mobile-cloud applications require restricted fine-grained adaptive and dynamic configuration and distribution of code and data components. Such adaptation can be achieved by online monitoring of user context, resources, communications and solving optimization to update component configuration and distribution

Offloading History:

<table>
<thead>
<tr>
<th>Research Focus</th>
<th>Feasibility of offloading</th>
<th>Offloading Decisions</th>
<th>Modern Offloading Era</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless standard</td>
<td>Making working prototypes Alg. for offloading decision</td>
<td>Cloud Computing Mobile-Cloud Computing</td>
<td></td>
</tr>
<tr>
<td>Virtualization</td>
<td>Virtual PC</td>
<td>VMware</td>
<td>Xen</td>
</tr>
<tr>
<td>Enterprise infrastructure</td>
<td>Application Servers XML</td>
<td>Web Services SOAP</td>
<td>Software as a Service</td>
</tr>
<tr>
<td></td>
<td>Software in Cloud</td>
<td>Amazon AWS, EC2 Cloud</td>
<td></td>
</tr>
</tbody>
</table>

Publications with Offloading in Topic:

- 1996 - 2000
- 2000 - 2007
- 2007-Present
- References: [1, 2, 3, 9, 10]
**Cloud Model**
- Traditionally, Cloud infrastructure is provided by third-party large organizations and is known as Public Cloud.
- Problems when stored by 3rd part:
  1. Potential lack of control and transparency
  2. Legal implications of data and applications
- This encouraged companies to build their own private Clouds
- Problems:
  - Not as efficient/scalable/elastic as public Cloud
- Recently, Hybrid Cloud:
  - Benefits from all advantages of public Cloud
  - Keeps confidential or sensitive data or algorithms in-house

**Cloud Application Model**
- Mobile-Cloud Application:
  - Should have independent components that can be distributed over public Cloud, private Cloud, and the end-user device.
  - These components can
    - store data or
    - perform computation
  - To maximize parallelism:
    - avoid global or shared state
    - Limit interactions to communicating using messages
- This aligns well with the Actor Model of Computation

**Cloud Application Model: Actor Model of Computation**
- Advantages:
  1. Natural Concurrency
  2. No shared state (No data race)
  3. Elasticity
  4. Decentralization
  5. Ease of scaling-up or -out
  6. Location Transparency
  7. Transparent Migration

**Cloud Application Model: Actor Model of Computation**
- We used SALSA:
  - Loyal to standard actor semantics
  - Depends on Java -> can be made to work on Android DalvikVM
  - Provides lightweight actors
  - Lightweight actors -> migration between devices is fast

---

**Overhead of SALSA actor creation**

**Overhead of SALSA actor migration**
**Approach:** An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

**Overall System Design:**
- Max. App. Performance
- Min. Mobile Energy Consumption
- Min. Cloud cost
- Min. Network data usage

**App./User Policy**
- Application Policy
- Access Restrictions
- Offloading Budget
- User preferences
- Quality

**System Properties**
- Battery Level
- Network Parameters
- Device profiling
- Application profiling

**Elasticity Manager**

**Application Component Distribution**

**System Monitor**

**Policy Manager**

**First Step toward Application Offloading:**
- **Ignoring Offloading process overhead:**
  - Running the same code on a faster machine provides linear speedup:
    \[ \text{Speedup} = \frac{S_s}{S_m} \]
    
    - \( S_s, S_m \): the speed of server and mobile device
    - \( F_{\text{server}}, F_{\text{mobile}} \): Freq. of the server and mobile processor
    - \( C_{\text{server}}, C_{\text{mobile}} \): No. of cores available on the server and mobile
    - \( X_{\text{server}} \): Additional speedup resulting from availability of more caches and more memory in addition to more aggressive pipelining

**System Resources and Application Performance:**

**Experimental results:**

<table>
<thead>
<tr>
<th>System</th>
<th>Remote Server</th>
<th>Mobile Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Mac OS X 10.9.4</td>
<td>Android 4.1.2</td>
</tr>
<tr>
<td>VM</td>
<td>JVM (JRE 1.6)</td>
<td>DalvikVM</td>
</tr>
<tr>
<td>Processor</td>
<td>Intel Core i7</td>
<td>ARM Cortex-A8</td>
</tr>
<tr>
<td>Processor speed</td>
<td>2.3 GHz</td>
<td>1 GHz</td>
</tr>
<tr>
<td>Number of cores</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>L2 Cache (per core)</td>
<td>256 KB</td>
<td>256 KB</td>
</tr>
<tr>
<td>L3 Cache</td>
<td>6 MB</td>
<td>-</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB</td>
<td>512 MB</td>
</tr>
</tbody>
</table>

\[ \text{Speedup} = \frac{S_s}{S_m} = \frac{F_{\text{server}} \times C_{\text{server}} \times X_{\text{server}}}{F_{\text{mobile}} \times C_{\text{mobile}}} = \frac{2.3 \times 4 \times 7}{1.0 \times 1} = 64 \]

**Detailed experimental results:**

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Description</th>
<th>Application Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>NQueen</td>
<td>Places N Queens on N*N board</td>
<td>Intensive (Comp)</td>
</tr>
<tr>
<td>Image</td>
<td>Detects &amp; recognizes all faces in a photo</td>
<td>Intensive (Comp)</td>
</tr>
<tr>
<td>Trap</td>
<td>Uses trapezoidal rule to calculate definite integral</td>
<td>Intensive (Comp)</td>
</tr>
<tr>
<td>Virus</td>
<td>Scans a file stream for a specific virus signature</td>
<td>-</td>
</tr>
<tr>
<td>Rotate</td>
<td>Rotates a disk image</td>
<td>-</td>
</tr>
<tr>
<td>EnSort</td>
<td>External Sort of the content of a file</td>
<td>Intensive (Comp)</td>
</tr>
<tr>
<td>Heat1</td>
<td>Simulates heat exchange on a board</td>
<td>Limited (Comp)</td>
</tr>
<tr>
<td>Heat2</td>
<td>Simulates heat exchange on a board</td>
<td>Limited (Comp)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Raw Speedup</th>
<th>Offload Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>NQueen</td>
<td>73</td>
<td>56</td>
</tr>
<tr>
<td>Image</td>
<td>91</td>
<td>44</td>
</tr>
<tr>
<td>Trap</td>
<td>30</td>
<td>21</td>
</tr>
<tr>
<td>Virus</td>
<td>28</td>
<td>21</td>
</tr>
<tr>
<td>Rotate</td>
<td>28</td>
<td>9</td>
</tr>
<tr>
<td>EnSort</td>
<td>46</td>
<td>36</td>
</tr>
<tr>
<td>Heat1</td>
<td>31</td>
<td>29</td>
</tr>
<tr>
<td>Heat2</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>
Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- Second Step toward Application Offloading:
  - Including Offloading process overhead:
  - Code offloading for Maximizing the Application Performance:
    - \( \frac{w}{S_m} + \frac{d_i}{S_s} + \frac{w}{B} > 0 \)
    - \( w \): amount of computation to offload
    - \( S_m, S_s \): Speed of mobile and remote server processor
    - \( B \): the network bandwidth
    - \( d_i \): size of data to be transferred
    - Offloading is beneficial, when left hand side > right hand side

References: [1]

Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- Second Step toward Application Offloading: (ctd.)
  - It is true for:
    - Large \( w \) -> program requires heavy computation
    - Large \( S_s \) -> server is fast enough
    - Small \( d_i \) -> small amount of data to be transferred
    - Large \( B \) -> available bandwidth is large
  - If this is not true, offloading is not economical even with \( S_s = \infty \)
  - So, only tasks with following conditions worth offloading:
    - Heavy computation (large \( w \))
    - Small data exchange (small \( d_i \))

References: [1]

Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- Code offloading for Maximizing Application Performance:
  - N-Queen problem:
    - Input: Integer value \( N \) -> \( d_i \) is very small
    - Problem is computationally expensive -> \( w \) is very large
    - \( B_{\min} = 1040 \times 1/16 = 65 \text{ bit/sec} \)
    - So, it is always efficient to offload N-Queen problem

References: [1]
Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- Code offloading for Maximizing Application Performance:
  - Image Processing Problem:
    - Input: Reasonable size of picture (\(w\) medium)
    - Assuming that remote server is super fast (\(S_s = \infty\))
    - Offloading depends on \(d_i\) and \(B\)
    - Case 1: Face detection part is offloaded
      - Whole image has to be sent \(-\to\) \(d_i\) is large
      - Then, Offload only if \(B\) is large
    - Case 2: Face detection is local
      - only extracted features needs to be sent over \(-\to\) \(d_i\) is small
      - Then, Offload even for smaller \(B\)

References: []

Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- Parameters affecting Offloading Decision:
  1. Problem Size:
     - N-Queen Problem:

References: []

Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- Parameters affecting Offloading Decision:
  2. Parallelism Degree of the problem
     - N-Queen Problem:

References: []

Figure: Speedup summary for different amount of work

Figure: Speedup summary with different degree of parallelism
Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

1. Code offloading for Maximizing Application Performance:
   - Parallelism Degree of the problem
     - Image Processing
     - Parallelism is great but ONLY if enough resources are available

Reference: [1]

Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

2. Code offloading: Min. Mobile Energy Consumption
   - Max performance case:
     \[ P_{\text{calc-mobile}} \times \frac{w}{S_m} > P_{\text{comm-mobile}} \times \frac{d}{S_y} \]

   - \( P_{\text{calc-mobile}} \): Mobile power consumption when performing comp.
   - \( P_{\text{comm-mobile}} \): Mobile power consumption when communicating data over network
   - \( P_{\text{idle-mobile}} \): Mobile power consumption when idle
   - Left hand side: energy consumed to perform comp. locally
   - Right hand side: energy consumed on the mobile to offload and then wait for the result to come back
   - Focus: Minimizing MOBILE energy consumed and not total energy

Reference: [1, 3]

Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

3. Code offloading in Large Applications:
   - Previous analysis is for deciding on offloading one piece of code with w amount of computation
   - It is beneficial to offload multiple parts (methods) at a time.
   - This requires considering communication between components

Reference: [1]
**Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps**

- **Code offloading in Large Applications:**
  - A possible result for partitioning the program graph:

- **How Parallelism can affect Large Application Offloading?**
  - **Full-Parallelism mean:**
    - Multiple Concurrent Components
    - Multiple Cloud resources
    - Simultaneous Local + Remote Execution

- **Image Processing: Speedup from Local + Remote execution with different problem size**

**References:** [9,10]
Approach: An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- Code offloading and Parallelism:
  - Maximizing Application Performance:
    
    \begin{align*}
    \text{Max} (\text{Application performance}) &= \text{Min} (\text{Total application execution time}) \\
    &= \text{Min} (\text{Max} (\text{execution time at different locations}) + \\
    & \quad \text{Max} (\text{communication time between different locations})) \\
    &= \text{Min} (\text{Max}_{l \in L} (\text{execution time at Location L}) + \\
    & \quad \text{Max}_{l \in L} (\text{communication time from components on L to others}))
    \end{align*}

    \begin{align*}
    \text{Max}_{l \in L} (\text{execution time at Location L}) &= \max \frac{1}{\text{Cores}(L)} \sum_{i=1}^{N} \text{LocEQ}(L, \text{Loc}(i, t_2)) \times \text{Exec}(i, \text{Loc}(i, t_2)) \times \text{JobNo}(i) \\
    \text{Max}_{l \in L} (\text{communication time from components on L to others}) &= \max \frac{1}{\text{Cores}(L)} \sum_{i=1}^{N} \sum_{j=1}^{N} \text{LocEQ}(L, \text{Loc}(i, t_2)) \times (1 - \text{LocEQ}(L, \text{Loc}(j, t_2))) \times \text{profiledConn}(i, j)
    \end{align*}

- Minimizing Mobile Device Energy Consumption:

\begin{align*}
\text{Min} (\text{Application Energy Consumption}) &= \text{Max} (\text{Energy saving on Mobile Device}) \\
&= \text{Max} (\text{Total Mobile Energy Saving by running components remotely}) \\
&\quad - \text{Energy Loss due to prev. local comm., becoming remote comm.} \\
&\quad + \text{Energy Saving due to prev. remote comm., becoming local comm.}
\end{align*}

\begin{align*}
\text{Total Mobile Energy Saving by running components remotely} &= \sum_{i=1}^{N} \text{LocEQ}(0, \text{Loc}(i, t_1)) \times (1 - \text{LocEQ}(0, \text{Loc}(i, t_2))) \times \text{Energy}(i) \\
\text{Energy Loss due to prev. local comm., becoming remote comm.} &= \sum_{i=1}^{N} \text{LocEQ}(0, \text{Loc}(i, t_2)) \times (1 - \text{LocEQ}(0, \text{Loc}(j, t_2))) \times \text{profiledConn}(i, j) \times P_{\text{comm, mobile}} \\
\text{Energy Saving due to prev. remote comm., becoming local comm.} &= \sum_{i=1}^{N} \text{LocEQ}(0, \text{Loc}(i, t_1)) \times (1 - \text{LocEQ}(0, \text{Loc}(j, t_1))) \times \text{profiledConn}(i, j) \times P_{\text{comm, mobile}}
\end{align*}

References: [1]
**Approach:** An Adaptive Prog. Framework for Mobile-Cloud Cloud Apps

- **Code offloading and Parallelism:** Max. App. Perf. In Parallel
- **Evaluation of Elasticity Manager:**

  ![Graph](image1.png)

  Image proc.: Local exec (base case) vs. Remote exec (Ideal Case) vs. Automatic Management (All components starting locally and then distributed)

- **Policy-based offloading adjustment:**
  - **Offloading with unlimited offloading budget:**

  ![Graph](image2.png)

  Image proc: Automatic mobile code offloading to a single remote server without any offloading budget restriction

- **Offloading with a fixed total offloading budget:**

  ![Graph](image3.png)

  Image proc: Automatic mobile code offloading to a single remote server with a fixed total offloading budget restriction

**References:** [1]
**Approach:** An Adaptive Prog. Framework for Mobile-Cloud Apps

- Policy-based offloading adjustment:
- Offloading with a limited offloading budget rate:

![Image Processing: Processing 500 images with limited offloading budget rate](image.png)

*Image proc: Automatic mobile code offloading to a single remote server with a limited offloading budget rate per time interval*

**Conclusion:** An Adaptive Prog. Framework for Mobile-Cloud Apps

- Conclusion:
  - A Flexible Fine-Grained Adaptive Framework for Mobile-Hybrid-Cloud Applications
  - Modeling of the Application Performance Maximization problem for Parallel Privacy-Preserving Hybrid Cloud offloading
  - Modeling of the Application Mobile Energy Consumption Minimization problem for Parallel Privacy-Preserving Hybrid Cloud offloading
  - An On-Line Monitor to profile dynamic system properties and to predict resource usage + Energy-model for fine-grained energy profiling
  - Policy-based offloading adjustments for Cost, Quality, Privacy
  - Results show great potential for improving mobile-cloud application experiences for both developers and users

**The End:** An Adaptive Prog. Framework for Mobile-Cloud Apps

*A Flexible Fine-Grained Adaptive Framework for Parallel Mobile Hybrid Cloud Applications*

Thanks for attending

Questions?