CSci 5105

Introduction to Distributed Systems

Edge Computing
Taxonomy

• Far edge: mobile, sensor/IoT, human
  – very limited networking
• The “edge”
  – local compute, storage
  – 1 hop to far edge, Internet connected
• Local cloud
  – Collection of edge nodes
• Centralized cloud
The Case for VM-based Cloudlets in Mobile Computing

Just-in-Time Provisioning for Cyber Foraging

Satya CMU
Cloud Offloading

Rich, interactive applications are emerging in mobile context

- Apple’s Siri, AR apps
- Wearable devices

Cloud offloading

- These applications are too expensive to run on clients alone!
- Offload computation to a back-end server at cloud

Today’s remote cloud is a suboptimal place; high latency and limited bandwidth
Solution: Cloudlet

• Resource poverty of a mobile device can be addressed by using a nearby resource-rich cloudlet
• Provide virtualized service to the mobile
• The end-to-end response time of applications executing in the cloudlet needs to be fast (few milliseconds) and predictable
Benefits

• Centralization => Dispersion
• Four reasons
  – Proximity/latency: highly responsive cloud services/applications (e.g. AR, VR)
    • Low latency, high b/w, low jitter
  – Scalability via edge analytics
    • Local processing of high b/w sensors (e.g. cameras)
  – Privacy enforcement
    • First point of contact between far edge and system
  – Masking cloud outages
  – Sheer volume of edge resources
Cloudlet as a Nearby Offload Site

Cloudlet: an nearby offloading site dispersed at the edges of the Internet

→ Let’s bring the cloud closer!

How to launch a custom back-end server at an arbitrary edge?
How Cloudlets Can Help

- A cloudlet only contains soft state such as cache copies of data or code that is available elsewhere.

<table>
<thead>
<tr>
<th></th>
<th>Cloudlet</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State</strong></td>
<td>Only soft state</td>
<td>Hard and soft state</td>
</tr>
<tr>
<td><strong>Management</strong></td>
<td>Self-managed; little to no professional attention</td>
<td>Professionally administered, 24x7 operator</td>
</tr>
<tr>
<td><strong>Environment</strong></td>
<td>“Datacenter in a box” at business premises</td>
<td>Machine room with power conditioning and cooling</td>
</tr>
<tr>
<td><strong>Ownership</strong></td>
<td>Decentralized ownership by local business</td>
<td>Centralized ownership by Amazon, Yahoo!, etc.</td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td>LAN latency/bandwidth</td>
<td>Internet latency/bandwidth</td>
</tr>
<tr>
<td><strong>Sharing</strong></td>
<td>Few users at a time</td>
<td>100s-1000s of users at a time</td>
</tr>
</tbody>
</table>

(b) Key Differences: Cloudlet vs. Cloud
Challenge

• To make this viable and scalable, we need an edge infrastructure (maybe 3rd party)
  – Wide-area: think mobiles and travel
  – Shared: multiple apps running on the edge
  – Enable any apps in any language in any OS + software libraries, etc.
  – Robust and Secure

• Encapsulate apps in VMs
Options

• Static provisioning
  – Store all possible VMs on the edge nodes
  – Feasible?
  – Advantages?

• Dynamic provisioning
  – Issues?
Just-in-Time Provisioning

GOAL: Just-in-time provisioning of a custom VM for offloading
1. Support **widest range of user customization** including OS, language, and library
2. Strong **isolation** between untrusted computations

- VM (virtual machine) cleanly encapsulates this complexity, but delays provisioning too expensive to send/boot a complete VM!

A traveler wants to use natural language translation with **speaker-trained voice recognition**
VM Synthesis

**VM Synthesis**: dividing a custom VM into two pieces

1) **Base VM**: Vanilla OS that contains kernel and basic libraries
2) **VM overlay**: A binary patch that contains customized parts
VM Synthesis

Steps for VM synthesis

1. User offloads operations to a Cloudlet.
2. Cloudlet with pre-populated base VM.
3. VM overlay.
4. Synthesize VM:
   - Decompress
   - Apply delta.
5. Resume launch VM
6. Backend Server in VM
7. Ready.
8. Offload operations.
Optimizations

1. Minimize VM overlay size
2. Accelerate VM synthesis

Creating VM overlay (offline)

- Launch VM
  - Deduplication
  - Reducing Semantic Gaps
  - VM overlay

VM synthesis (runtime)

- VM overlay
  - Pipelining
  - Early Start
  - Launch VM

Transfer

new site
MAUI: Making Smartphones Last Longer With Code Offload

Microsoft Research
Battery is a scarce resource

- CPU performance during same period: \(246X\)
- A solution to the battery problem seems unlikely
Mobile apps can’t reach their full potential

- **Slow, Limited or Inaccurate**
- **Too CPU intensive**
- **Power Intensive**
- **Not on par with desktop counterparts**
- **Limited**

Speech Recognition

Augmented Reality

Interactive Games
One Solution: Remote Execution

• Remote execution can reduce energy consumption

• Challenges:
  – What should be offloaded?
  – How to dynamically decide when to offload?
  – How to minimize the required programmer effort?
Energy Results

What do these energy results tell us?
MAUI: Mobile Assistance Using Infrastructure

MAUI Contributions

– Makes dynamic offload decisions
– Optimize for energy reduction
– Profile: device, network, application
MAUI Architecture
How Does a Programmer Use MAUI?

• Goal: make it simple to MAUI-ify apps
  – Build C# app as a standalone phone app
  – Add .NET attributes to indicate “remoteable”
  – Follow a simple set of rules

```csharp
[Remoteable]
ArrayList GetValidMoves(Square s)
{
    if (s.IsEmpty())
    {
        return new ArrayList();
    }
    if (s.Piece.IsEnemyOf(active))
    {
        //this piece does not belong to the active side, no moves possible
        return new ArrayList();
    }
    //forward the call to the Rule-class
    return rules.getMoves(s);
}
```
Design Choice

• Programmer selects ‘remoteable’
  – Granularity is the method
  – This method is eligible for remote execution
  – What is eligible or ineligible?

• Alternative is marking methods ‘local’
Programmer Effort?

• Must be smart enough to tag methods

• Must be willing to optimize
  – Missile example
  – Call a method X times with each $pos_i$
  – Call a method once with $\{pos_1, pos_2, \ldots\}$
  – Why is this important to do?
Contrast

• Two versions of method for phone and server
  – either phone ships code
  – or server obtains code from the “cloud”

• Is it enough just to ship method code?
  – Must also transmit dependent state (e.g. accessed member vars, globals, …)
  – Interesting delta optimization: Maui can remember what it has previously shipped
Example

//original interface
class IEnemy {
    [Remoteable] bool SelectEnemy(int x, int y);
    [Remoteable] void ShowHistory();
    void UpdateGUI();
}

//remote service interface
class IEnemyService {
    MAUIMessage<AppState, bool> SelectEnemy(AppState state, int x, int y);
    MAUIMessage<AppState, MauiVoid> ShowHistory(AppState state);
}

What issue does this raise?
Failure?

• If remote method fails
  – E.g. network failure

• Maui re-executes it
  – Issues?
  
global int i;
method => i++;
Decision Process

- Continuous monitoring when executing to build a program graph
  - Data size
  - Network cost
  - Execution time

- Server solves linear program to estimate where to run method(s)
Maui Graph

Energy model based on benchmarks
CPU cycles => Joules

Past execution is a good predictor of future?
MAUI Solver

A sample callgraph

A

1000mJ

B
900 mJ

C
5000 mJ

D
15000 mJ

Energy for execution
How much can MAUI reduce energy consumption?

Face Recognizer

<table>
<thead>
<tr>
<th>Energy (Joules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

- Smartphone only
- MAUI (Wi-Fi, 10ms RTT)
- MAUI (Wi-Fi, 25ms RTT)
- MAUI (Wi-Fi, 50ms RTT)
- MAUI (Wi-Fi, 100ms RTT)
- MAUI* (3G, 220ms RTT)

An order of magnitude improvement on Wi-Fi

Big savings even on 3G
How much can MAUI improve performance?

### Face Recognizer

- **Execution Duration (ms)**
- **Improvement of around an order of magnitude**

- **Graph**:
  - Smartphone only
  - MAUI (Wi-Fi, 10ms RTT)
  - MAUI (Wi-Fi, 25ms RTT)
  - MAUI (Wi-Fi, 50ms RTT)
  - MAUI (Wi-Fi, 100ms RTT)
  - MAUI* (3G, 220ms RTT)
Latency to server impacts the opportunities for fine-grained offload.

- Solver would decide not to offload

**Arcade Game**

- Up to 40% energy savings on Wi-Fi

<table>
<thead>
<tr>
<th>Energy (Joules)</th>
<th>Smartphone only</th>
<th>MAUI (Wi-Fi, 10ms RTT)</th>
<th>MAUI (Wi-Fi, 25ms RTT)</th>
<th>MAUI (Wi-Fi, 50ms RTT)</th>
<th>MAUI (WiFi, 100ms RTT)</th>
<th>MAUI* (3G, 220ms RTT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per frame communication modest computation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Next Time

• A few Cool Techniques
• Final
• Evaluations