Special Topics:
CSci 8980 Edge Computing Outsourcing II

Jon B. Weissman (jon@cs.umn.edu)

Department of Computer Science
University of Minnesota
COMET's Goals

1. Improve mobile computation speed
2. Require no programmer effort
3. Work with existing applications
4. Handle network failures

Multi-threaded Java applications
Fine-grain offloading of threads
Provides a global address space across all available memories
Distributed Shared Memory

- COMET = offloading + DSM
  - Offloading => usual benefits
  - DSM guarantees state is available everywhere

- COMET focus is “how” vs. “what” to offload
  - MAUI/Cuckoo (and others) focused on “what”
Java Memory Model

- Dictates which writes a read can observe
- Specifies 'happens-before' partial order
  - Access in single thread totally ordered
  - Lazy Release Consistency locking for performance

- Fundamental memory unit is the field
  - Known alignment, known width

- DSM: track and transmit changes
DSM => VM-synchronization

- Used to establish 'happens-before' relation
- Directed operation between mobile and remote
- Tracks “dirty” fields
- Synchronizes
  - Java thread stacks
  - Java heap
Figure 2: At the beginning, baz() and bar() are running in separate threads on ep1, while qux() is running as a different thread on ep2. ep2 holds the ownership of mutex at the beginning, but no thread lock on it.
Native Methods

- Written in C with bindings for Java
- Native methods exist to
  - Access device resources (file system, display, etc)
  - For performance reasons
  - To work with existing libraries
- Not generally safe to outsource due to hidden state
  - Manually white list safe native methods
Failure Recovery

- VM-synchronization is recovery safe
- Always leave enough information on client
- If server is lost resume threads running locally!
What to migrate?

- General case: any thread that has not accessed native methods for a while
  - Why?
- Lock case: thread wants a lock on an object held at the other end-point
  - Migrate thread or transfer lock and state
Tau-Scheduler

\[ T = 2 \times \text{VM-synchronization time} \]
Results

- 8 applications from Google Play
  - Average speed-up of 2.88X on WiFi / 1.28X on 3G
  - Average energy saving of 1.51X on WiFi / 0.84X on 3G

- 2 computation benchmark applications
  - 10.4X speed-up w/ WiFi on Linpack
  - 500+X speed-up w/ multi-threaded factoring
Results

- Java JavaScript Interpreter
  - Ran with SunSpider JavaScript benchmark
Discussion

- Scheduling seems naïve
- COMET may decide to send over data that is not needed for computation thus wasting bandwidth
CloneCloud - Elastic Execution between Mobile Device and Cloud
Motivation

- Mobile applications with richer functionalities are becoming ubiquitous
- Mobile devices have limited compute and power resources
- Cloud: abundant resources
  - Greater performance
  - Greater accuracy
  - Reduced energy
Goals

- Allow *fine grained flexibility* on where to run parts of the application
- Take programmer out of the business of application partitioning
  - make it more automatic and seamless
- What results is a dynamic split of client/server roles
High-level Details

- Application-level VMs
  - Remote clone of process/VM
  - Local clone
  - Multi-threaded applications

- Static/Dynamic Analysis
  - Determines chosen thread
  - migration/reintegration points
    - (method call)/(method return/migrate back)

- Threads run concurrently in both places
  - All threads can exploit native features
(a) Single-machine computation

(b) Distributed computation
Partitioning

- Partitioning mechanism yields the partitions in the application that are optimal at execution time.
- Partitions are based on offline profiling.
- At run time, the execution picks a partition from the DB (based on current state?) and then modifies the executables before invocation.
Partitioning
Static Analyzer

- The static analyzer identifies the legal partitions of the application executable according to the set of constraints
- Migration is restricted to the method entry and exit points
- Two more restrictions for simplicity
  - Migration is allowed only at the boundaries of application methods but not core system library methods
  - Migration is not allowed at native method boundaries
Static Analyzer: Example

Orthogonal to which code a thread owns

class C {
    void a () {
        if () {b(); c();}
    }
    void b() {
        } // lightweight
    void c() {
        } // expensive
    }
    void main () {
    C c; c.a();
    }

(a) program     (b) static control-flow graph     (c) partitioned graph

Figure 4. An example of a program, its corresponding static control-flow graph, and a partition.
Static Analyzer - constraints

- Three properties of any legal partition
  - Methods that access specific features of a machine must be pinned to the machine
  - Methods that share native state must be collocated at the same machine
  - Prevent nested migration
    - Do not partition at \( a() \) AND \( b() \) or \( c() \)

```java
class C {
    void a () {
        if () {b(); c();}
    }
    void b() {
        // lightweight
    }
    void c() {
        // expensive
    }
    void main () {
        C c; c.a();
    }
}
```
Dynamic Profiler

- Profiler collects data that will be used to construct the cost model
- Currently uses randomly inputs
- Each execution is run once on mobile device and once on the clone in the cloud
- Profiler outputs a set $S$ of executions and a “profile tree”, for both mobile device ($T$) and the clone ($T'$) for each complete execution
Dynamic Profiler - Example

(a) trace

(b) profile tree

residual
Profile Tree

- One node for each method invocation
- Every non-leaf node also has a leaf child called its residual node
- Residual node holds residual cost which represents the cost of running the body of code excluding the costs of the methods called within it
- Computation cost for invocation $i$ $Cc(i, l); l=0$ on mobile device and filled from $T$, $l=1$ on the clone and filled from $T'$
- Migration cost $Cs(i)$; sum of a suspend/resume cost and the transfer cost
Distributed Execution

- Thread granularity migration
  - Migration operates at the granularity of a thread

- Native-Everywhere:
  - Enables migrated threads to use native non-virtualized hardware (GPUs, Cryptographic accelerators, etc).
Migration Overview

Mobile Phone

<table>
<thead>
<tr>
<th>Application</th>
<th>Migrator</th>
<th>App-VM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Migrator</th>
<th>App-VM</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clone

<table>
<thead>
<tr>
<th>Clone</th>
<th>Migrator</th>
<th>App-VM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Migrator</th>
<th>App-VM</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram notes:
1. (1) Connection from Mobile Phone to Node Manager
2. (2) Connection from Clone to Node Manager
3. (3) Connection from Clone to Node Manager
4. (4) Connection from Mobile Phone to Node Manager

Partitions

Evaluation

SUMMARY
Suspend/Resume

- **Suspend and capture**
  - Thread migrator suspends thread
  - Captures its state, passes it to node manager
  - Node manager transfers the capture to clone

- **Resume and Merge**
  - When mobile thread needs state of remote thread ..
  - Clone's thread migrator captures and packages the thread state
  - Node manager transfers the capture back to the mobile device
  - Migrator in the original process is given the capture for resumption
Object Mapping

(1) Mobile Phone

<table>
<thead>
<tr>
<th>Reference</th>
<th>MID</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>1</td>
<td>null</td>
</tr>
<tr>
<td>0x02</td>
<td>2</td>
<td>null</td>
</tr>
<tr>
<td>0x03</td>
<td>3</td>
<td>null</td>
</tr>
</tbody>
</table>

(2) Clone

<table>
<thead>
<tr>
<th>Reference</th>
<th>MID</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x21</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>0x22</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>0x23</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>0x24</td>
<td>null</td>
<td>14</td>
</tr>
<tr>
<td>0x22</td>
<td>null</td>
<td>15</td>
</tr>
</tbody>
</table>

(3) Mobile Phone

<table>
<thead>
<tr>
<th>Reference</th>
<th>MID</th>
<th>CID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>0x02</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>0x03</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>0x04</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>0x05</td>
<td>5</td>
<td>15</td>
</tr>
</tbody>
</table>

migration

reintegration

GC’ed

New objects

Figure 8. Mean execution times of virus scanning (VS), image search (IS), and behavior profiling (BP) applications with standard deviation error bars, three input sizes for each. For each application and input size, the data shown include execution time at the phone alone, that of CloneCloud with WiFi (CC-WiFi), and that of CloneCloud with 3G (CC-3G). The partition choice is annotated with M for “monolithic” and O for “off-loaded,” also indicating the relative improvement from the phone-alone execution.
Figure 9. Mean phone energy consumption of virus scanning (VS), image search (IS), and behavior profiling (BP) applications with standard deviation error bars, three input sizes for each. For each application and input size, the data shown include execution time at the phone alone, that of CloneCloud with WiFi (CC-WiFi), and that of CloneCloud with 3G (CC-3G). The partition choice is annotated with M for “monolithic” and O for “off-loaded,” also indicating relative improvement over phone-only execution.
Discussion

How does static partitioner handle dependencies?

What about synchronization of shared state accessed by different threads?
On Tuesday

Outsourcing III

• Parametric
• ThinkAir