CloneCloud

PRESENTED BY AYUSH GARG

Motivation/Introduction

- Mobile applications with richer functionalities are becoming ubiquitous
- But mobile devices are limited by their computing resources
- Cloud – the place for abundant resources
- Clouds provide opportunity to do huge computations quickly and accurately

So why not use cloud for mobile computations??

Goals

- Main Goal
  - CloneCloud boosts unmodified mobile applications by seamlessly off-loading part of their execution from the mobile device onto device clones operating in a computational cloud
- Other Aims:
  - Allow fine grained flexibility on what to run where
  - Take programmer out of partitioning business

CloneCloud

- Follows the intuition that "as long as execution on the cloud is significantly faster than execution on the mobile device, paying the cost for sending the relevant data and code from the device to the cloud and back may be worth it."

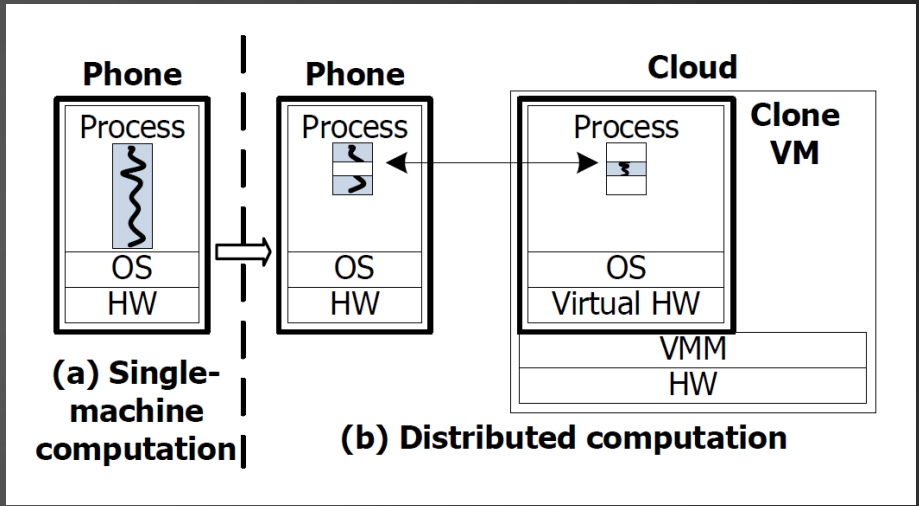
- The partitioning component for finding migration points uses – static analysis to find the constraints and the dynamic profiling for building the cost model for execution and migration
- It finally uses an optimizer that uses the above constraints and cost models to derive the partitions

Overview

- Schema
- Partitioning
  - Static Analyzer
  - Dynamic Profiler
  - Optimization Solver
- Distributed Execution
- Evaluation
- Conclusion

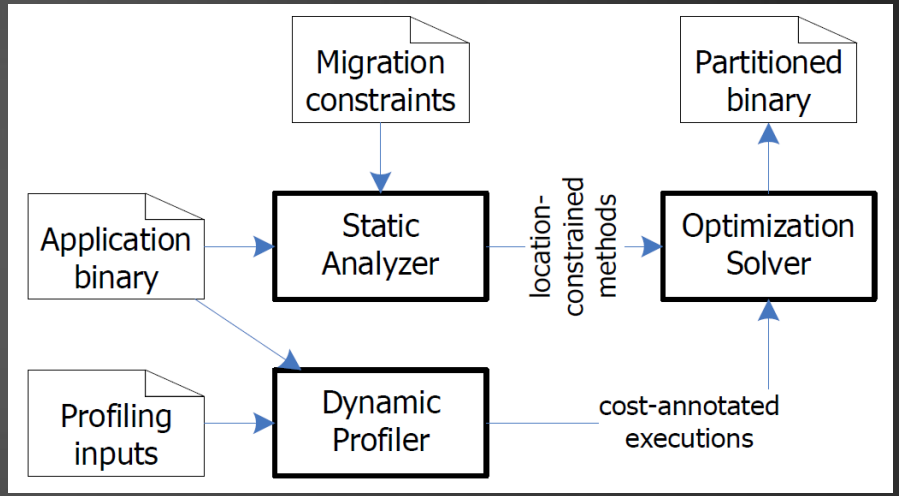
Schema

- Clone an unmodified application executable
- The executable is running at mobile device
- At automatically chosen points individual threads migrate to device clone in cloud along with the application state
- Migrated thread executes on clone accessing native features on hosting platform
- Merge remote and local state back into original process



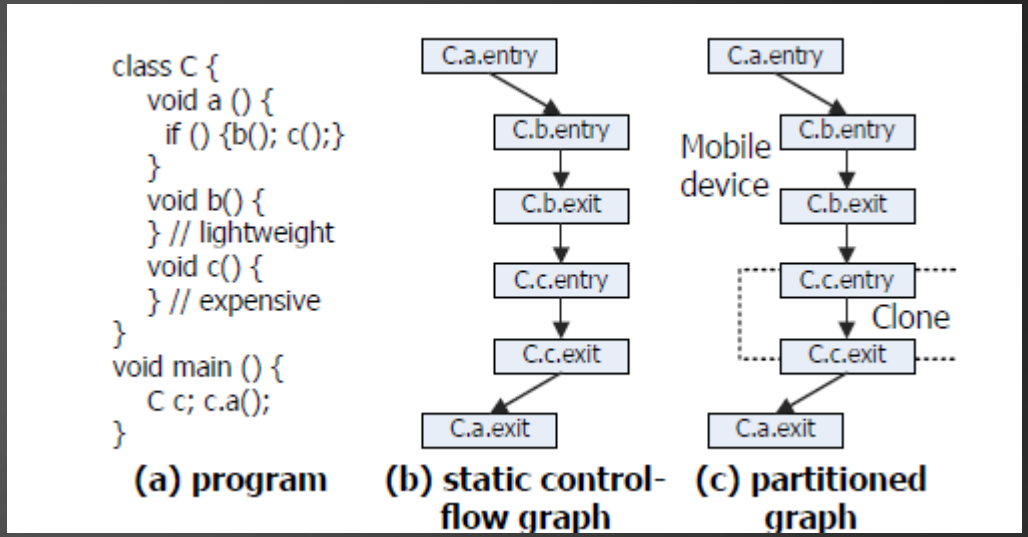
Partitioning

- The partitioning mechanism yields the partitions in the application
- Run multiple time under different conditions and objective functions – stores all partitions in a database
- At run time, the execution picks a partition among these and modifies the executable before invocation
- It has three components –
  - **static analyzer**
  - **dynamic profiler**
  - **optimization solver**



Static Analyzer

- Identifies the legal partitions of the application executable according to a 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 allowed at the VM-layer method boundaries but not native method boundaries



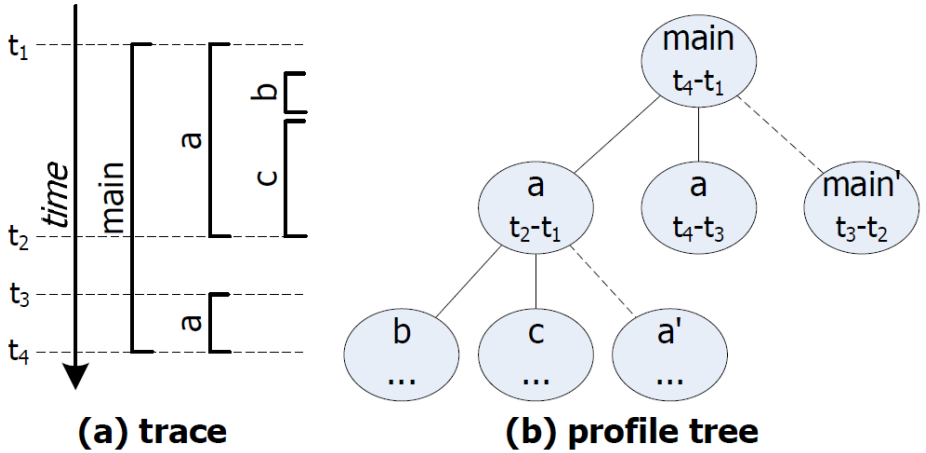
Static Analyzer - constraints

- Methods that access specific features of a machine must be pinned to the machine [V_M]
  - Static analysis marks the declaration of such methods with a special annotation M
  - Done once for each platform, not repeated for each application
- Methods that share native state must be collocated at the same machine [V_{NatC}]
  - When an image processing class has initialize, detect and fetchresult methods that access native state, they need to be collocated
- Prevent nested migration
  - Static analysis of the control flow graph to identify the set of methods called directly by a method (DC) and transitively (TC)

Dynamic Profiler

- Profiler collects data used to construct cost model
- Uses randomly chosen set of inputs
- Each execution is run once on mobile device and once on the clone in the cloud
- Profiler outputs set of executions S and a “profile tree”, for both mobile device and the clone
- Example…

Dynamic Profiler - example



Dynamic Profiler - 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 by it
- Each edge is annotated with the state size at the time of invocation of the child node, plus the state size at the end of that invocation
  - Amount of data that the migrator needs to capture and transmit in both directions if the edge were to be a migration point
- Computation cost 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

Optimization Solver

- Aim is to pick application methods to migrate to the clone to minimize the expected cost of the partitioned application
- Decision variable R(m) m= method in the application.
- R(m)=1 -> partitioner places a migration point at the entry point of the method.
- R(m) = 0 -> method m is unmodified in the application binary
- But not all partitioning choices for R(.) are legal

Optimization problem

$$
\begin{aligned}
L(m_1) \neq L(m_2), \quad & \forall m_1, m_2 : DC(m_1, m_2) = 1 \\
& \wedge R(m_2) = 1 \quad (1) \\
L(m) = 0, \quad & \forall m \in V_M \quad (2) \\
L(m_1) = L(m_2), \quad & \forall m_1, m_2, C : m_1, m_2 \in V_{Nat_C} \quad (3) \\
R(m_2) = 0, \quad & \forall m_1, m_2 : TC(m_1, m_2) = 1 \\
& \wedge R(m_1) = 1 \quad (4)
\end{aligned}
$$

- Using the decision variables R(.), the auxiliary decision variables L(.), the method sets V_M and V_{NatC} for all classes C defined during static analysis, and the relations I, DC and TC

- *DC(m1,m2 -> method m1 Directly Calls method m2*
- *TC(m1,m2) -> method m1 Transitively Calls method m2*

Cost of a partition

- The cost of a legal partition R(.) of execution E is given by

$$
\begin{aligned}
C(E) &= Comp(E) + Migr(E) \\
Comp(E) &= \sum_{i \in E, m} [(1 - L(m))I(i,m)C_c(i,0) \\
&\quad + L(m)I(i,m)C_c(i,1)] \\
Migr(E) &= \sum_{i \in E, m} R(m)I(i,m)C_s(i)
\end{aligned}
$$

- Optimization objective is to choose R() to minimize $\sum_{E \varepsilon S} C(E)$

Distributed Execution

At Mobile Device:

- User attempts to launch partitioned application
- Execution conditions looked up in db
- Return binary of modified application
- New process launched
- Migration point->
  - Executing thread is suspended
  - State (*stack frames, relevant data objects in heap and register contents*) packaged and shipped to synchronized clone

- Returned package is merged int state of original process

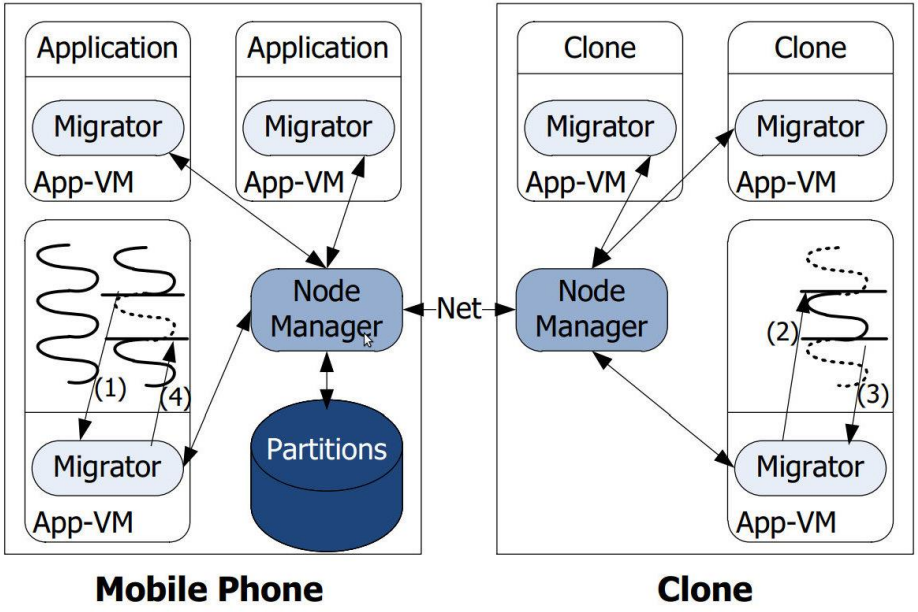
At Clone:

- Thread state is initiated with migrated stack and heap objects
- Thread resumed
- Integration point->
  - Suspended, state packaged and shipped

Distributed Execution - Components

- Migrator:
  - per process
  - manipulates internal state of application-layer virtual machine
- Node Manager:
  - per node
  - enables application-unspecific node maintenance, including file-system synchronization
  - amortizes the cost of communicating with the cloud over a single, possibly authenticated and encrypted transport channel
- Partition Database

Migration Overview



Migration Overview

Suspend and Capture

- Migrator suspends migrant thread
- Captures its state, passes it to node manager
- Node manager transfers the capture to clone

Resume and Merge

Resume:

- Node manager transfers capture to migrator
- Migrator overlays the thread context over the clean process address space
- Captured classes and object instances are allocated in the virtual machine's heap
- New thread is created with the state, heap and registers
- At integration point, 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

Merge:

- the context updates the original thread state to match the changes effected at the clone

Evaluation

Application	Input Size	Phone Exec. (sec)	Clone Exec. (sec)	Max Speedup	CloneCloud 3G			CloneCloud WiFi		
					Exec. (sec)	Part.	Speedup	Exec. (sec)	Part.	Speedup
Virus scanning	100KB	5.70	0.30	19.00	5.70	Local	1.00	5.70	Local	1.00
	1MB	59.70	2.95	20.24	59.70	Local	1.00	20.30	Offload	2.94
	10MB	640.90	30.90	20.74	114.52	Offload	5.60	45.60	Offload	14.05
Image search	1 image	22.20	0.97	22.89	22.20	Local	1.00	15.90	Offload	1.40
	10 images	212.20	8.40	25.26	98.40	Offload	2.16	23.60	Offload	8.99
	100 images	2096.70	83.20	26.20	193.10	Offload	10.86	98.90	Offload	21.20
Behavior profiling	depth 3	3.60	0.20	18.00	3.60	Local	1.00	3.60	Local	1.00
	depth 4	46.80	2.00	23.40	46.80	Local	1.00	14.50	Offload	3.23
	depth 5	315.80	12.00	26.32	77.50	Offload	4.07	25.40	Offload	12.43

Conclusion

- A design that achieves basic augmented execution of mobile applications on the cloud
- Prototype delivers up to 20x speed up
- Programmer involvement is not required
- Opens up a path for a rich research agenda in hybrid mobile-cloud systems

Two unique features of CloneCloud

- 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.)

Thank You!