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*

Related work

Year	System Name	Goal	Offloading Decision	Partition Level	Parallel	Policy-based Security/ Privacy	Manual Work	No. Cloud spaces
2010	MAUI	Mobile Energy Saving	Dynamic	Method	No	No	Yes	1
2011	CloneCloud	Mobile Energy Saving = Performance Improvement	Static	Method	Pseudo	No	No	1
2012	ThinkAir	Mobile Energy Saving = Performance Improvement	Dynamic	Method	Pseudo	No	Yes	1
2012	Cloud OS (COS)	Load Balancing for Cloud space	Dynamic	Actor	Yes	No	No	Can be Many
2015	IMCM	Mobile Energy Saving, Performance Improvement, Policy-based Distribution	Dynamic	Actor	Yes	Yes	No	Many

IMCM: Illinois Mobile Cloud Manager

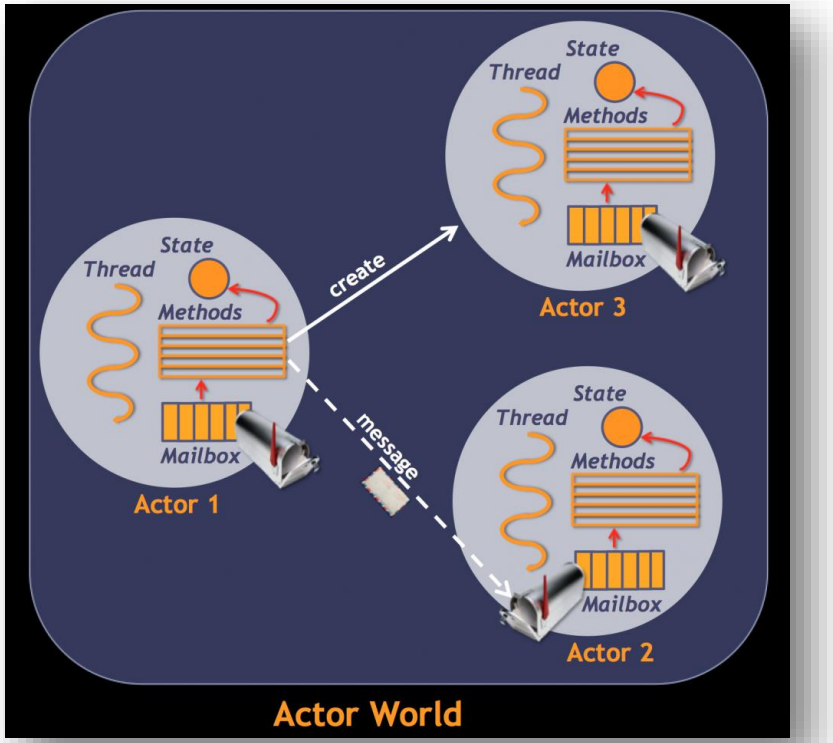
- Code offloading:
  - Automatic
  - Dynamic
  - Fine-grained
  - Parallel
- Supports:
  - Hybrid cloud with multiple cloud spaces
- Provides:
  - Policy-based control by cloud provider, app developer, user

Actor model

- Programming model
  - Natural concurrency
  - Decentralization
  - No shared state
  - Scalability
  - Location transparency

    -> Transparent migration

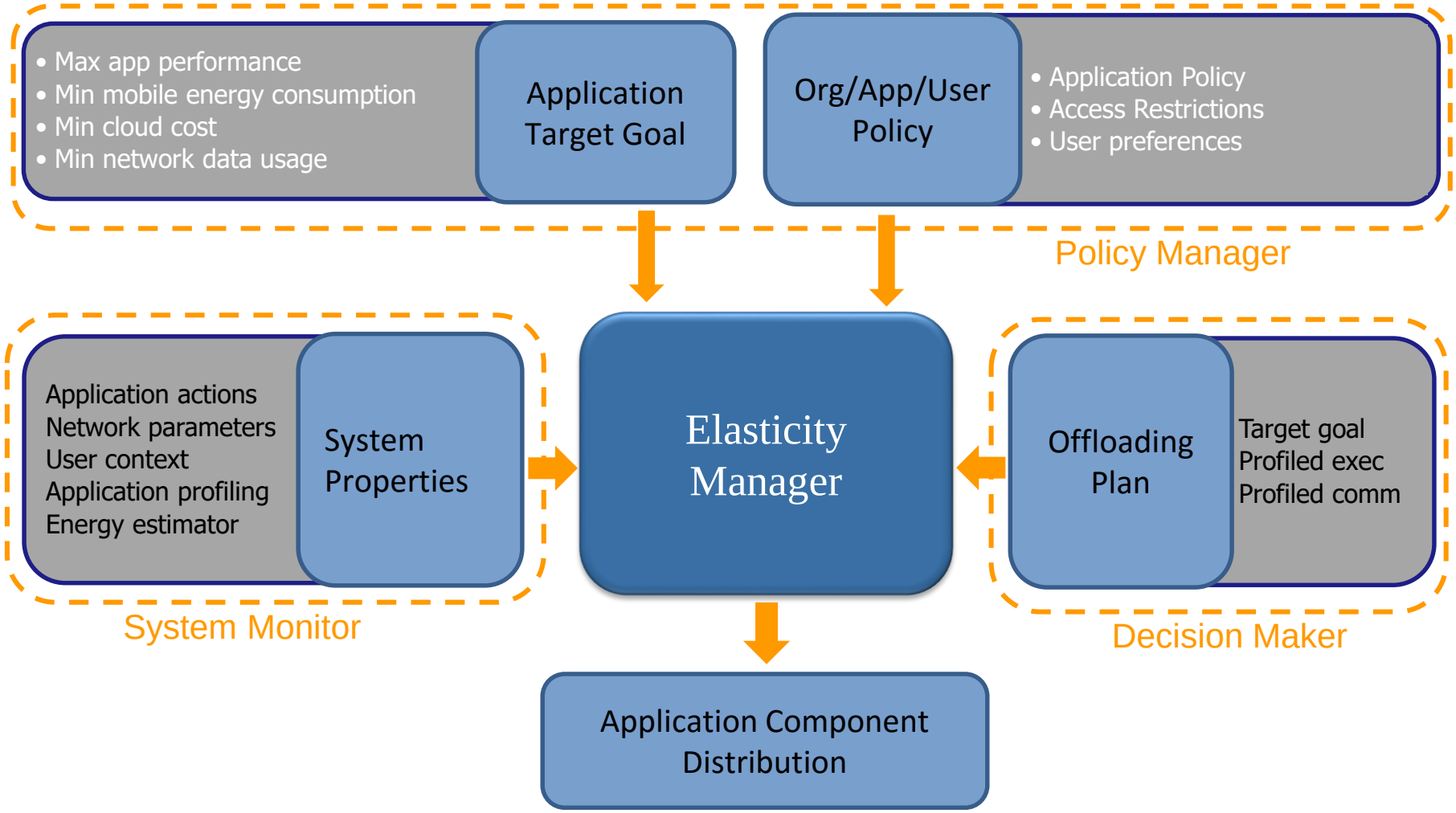
- Implementation: SALSA*
  - Full actor semantics
  - Lightweight actors
  - Migration support
  - Portability (Java-based)



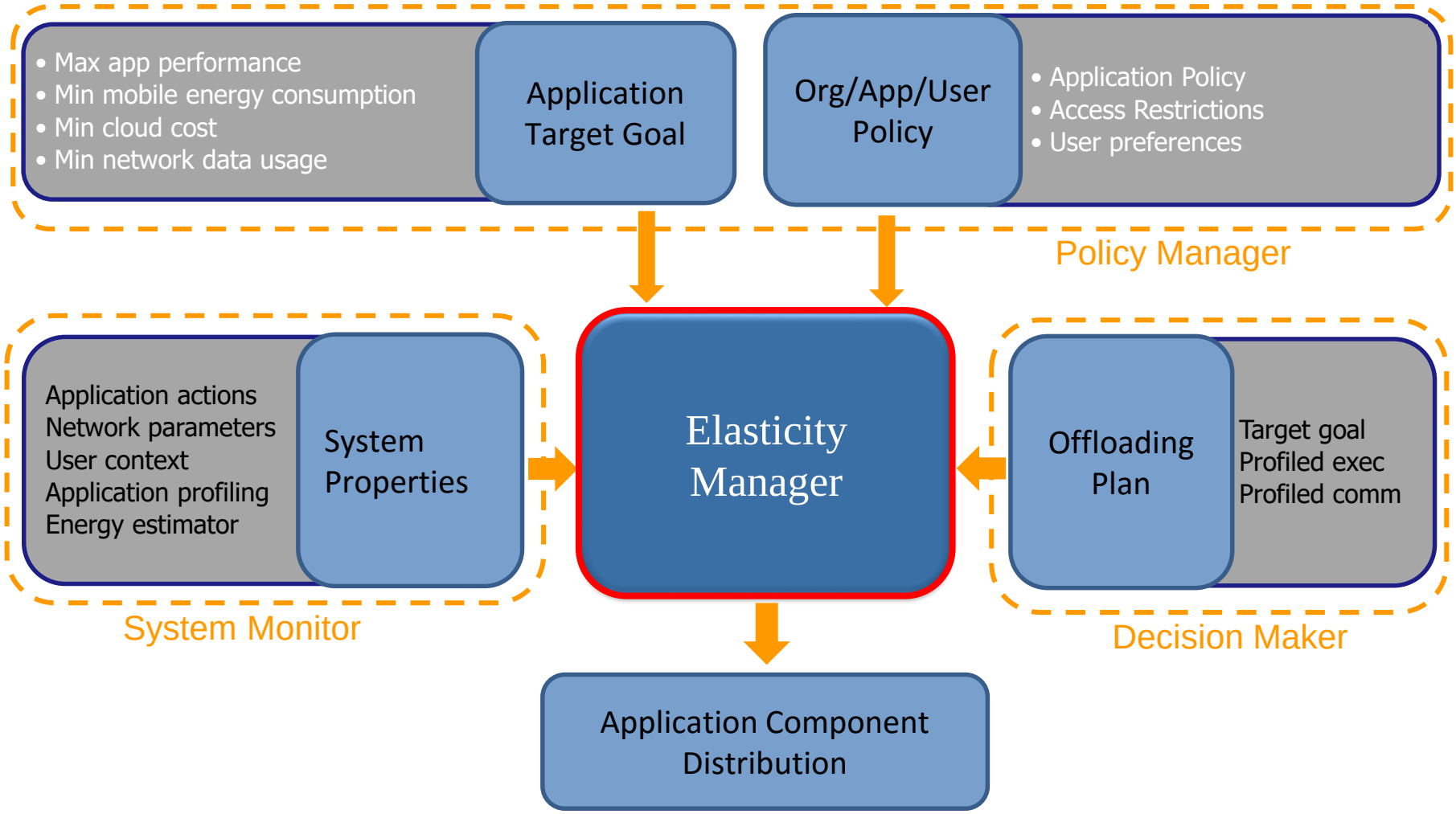
SALSA actors

```
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!" );

   }
}
```

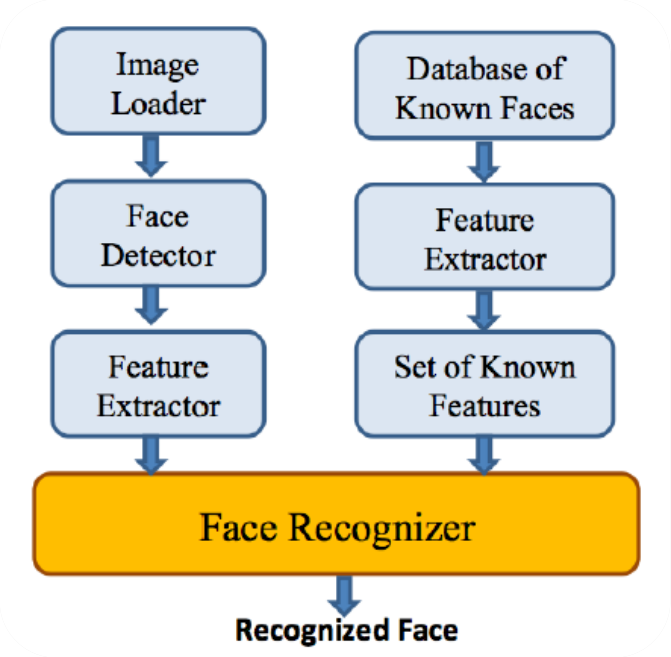
IMCM framework

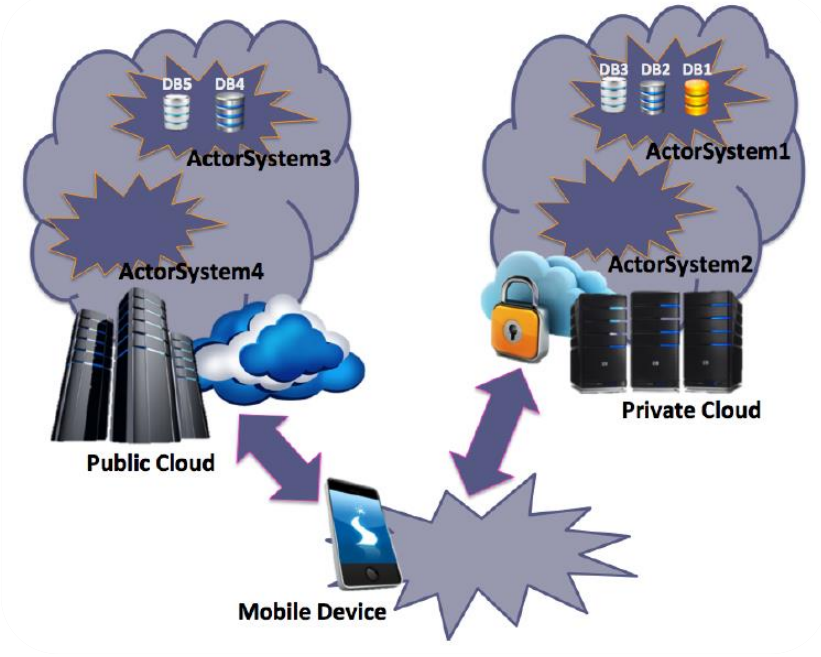


Today: Optimization – Elasticity Manager



Example: face recognition app

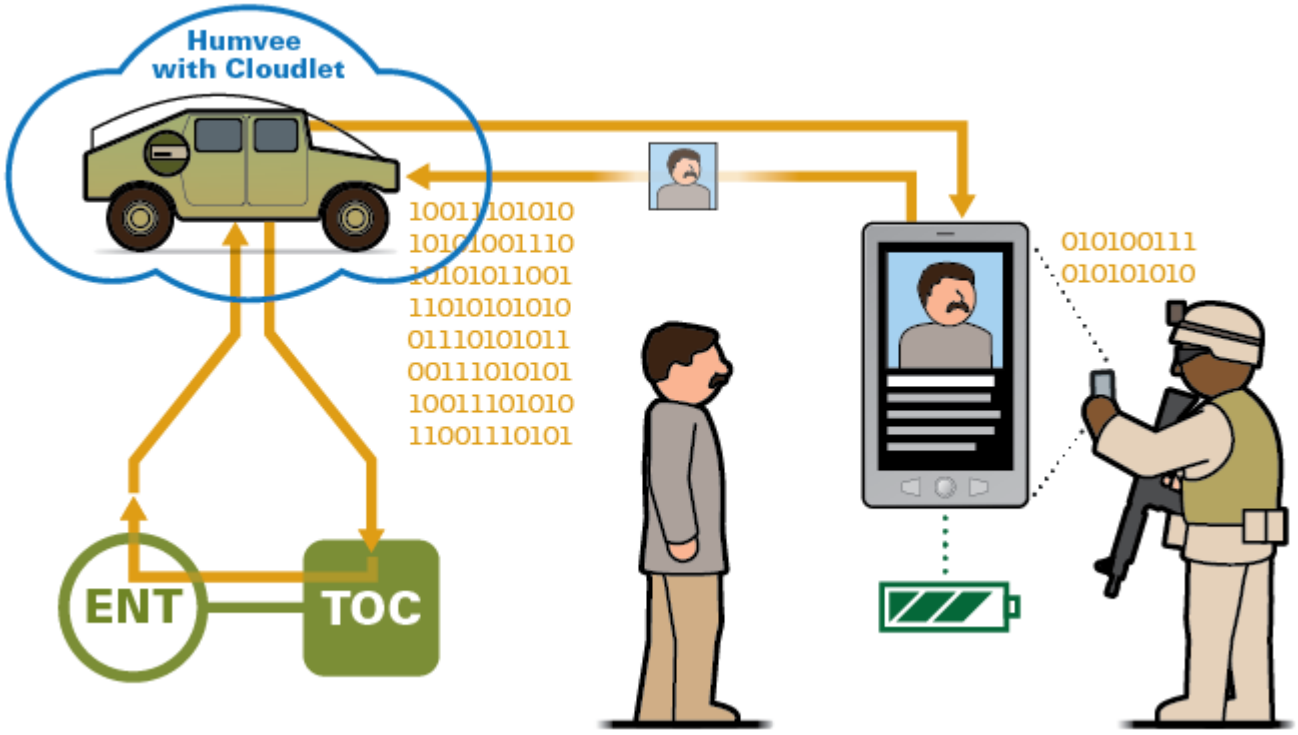




- Components with different computation, bandwidth, energy characteristics
- Different partitioning schemes depending on policy-based restrictions on certain components/data

Example: Tactical Cloudlet (CMU)

- Tactical Cloudlets: Moving Cloud Computing to the Edge



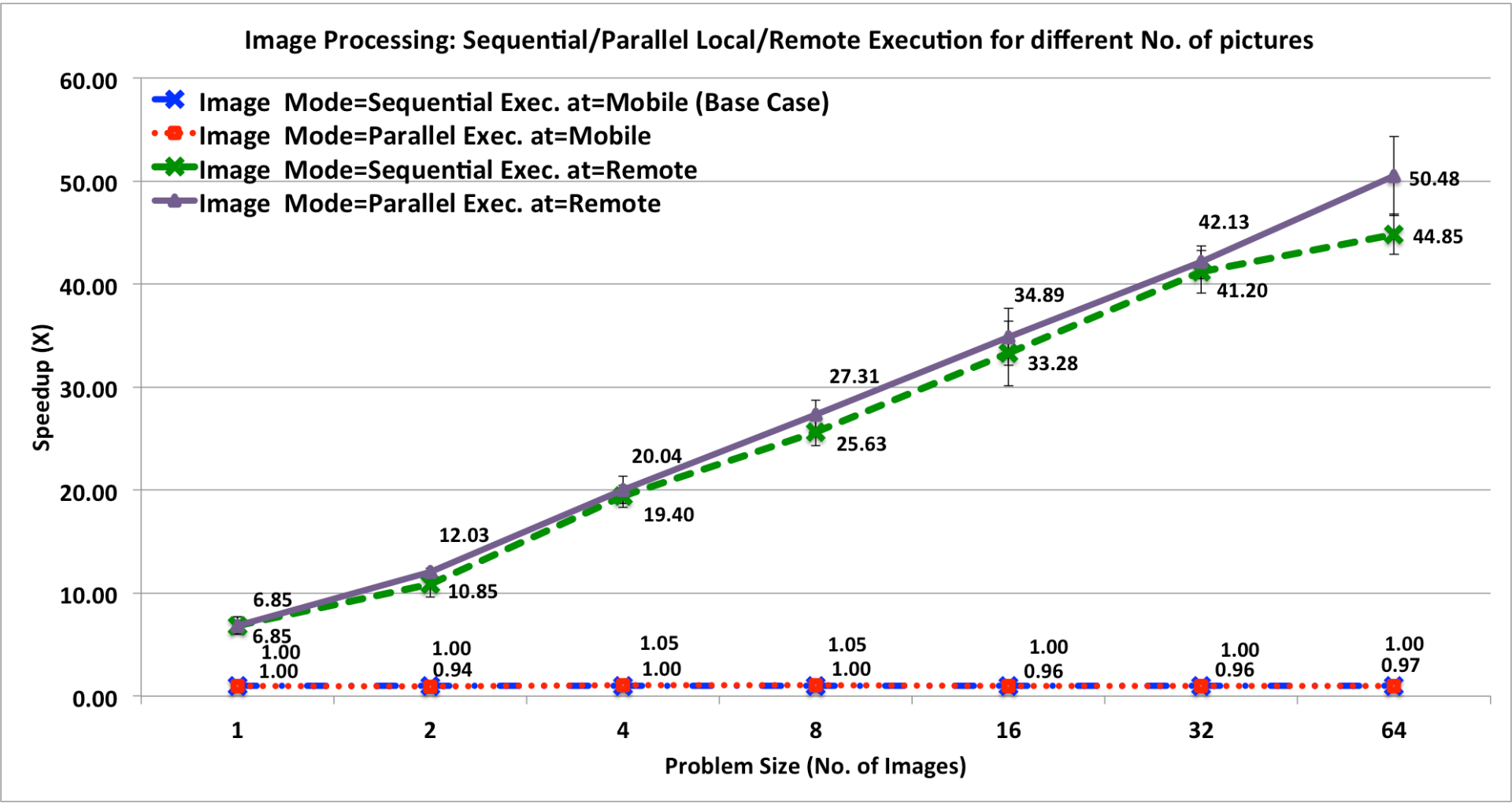
Credit: Software Engineering Institute, CMU

Benchmark results

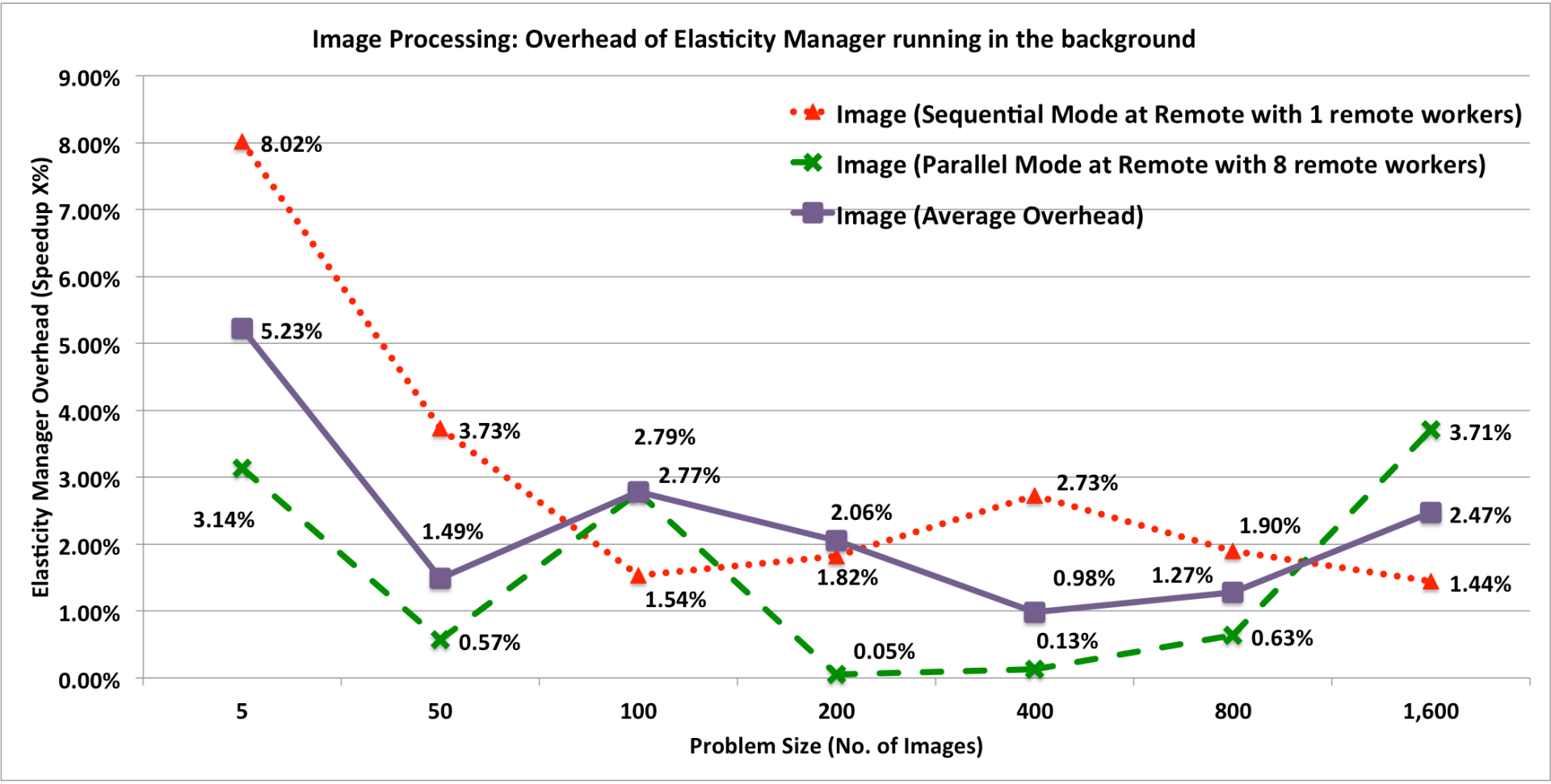
- Running the same code on more powerful HW
- Running some components in parallel
- Keeping in mind the cost of offloading

Experiment	Application Characteristic				Raw Speedup	Offload Speedup
	Comp.	Comm.	I/O			
			read	write		
NQueen	intensive	-	-	-	73	56
Image	intensive	limited	limited	-	91	44
Trap	intensive	limited	-	-	30	21
Virus	-	-	intensive	-	28	21
Rotate	-	-	intensive	intensive	28	9
ExSort	intensive	-	intensive	intensive	46	36
Heat1	limited	medium	-	-	31	29
Heat2	limited	high	-	-	14	14

Face recognition app speedup



IMCM framework overhead



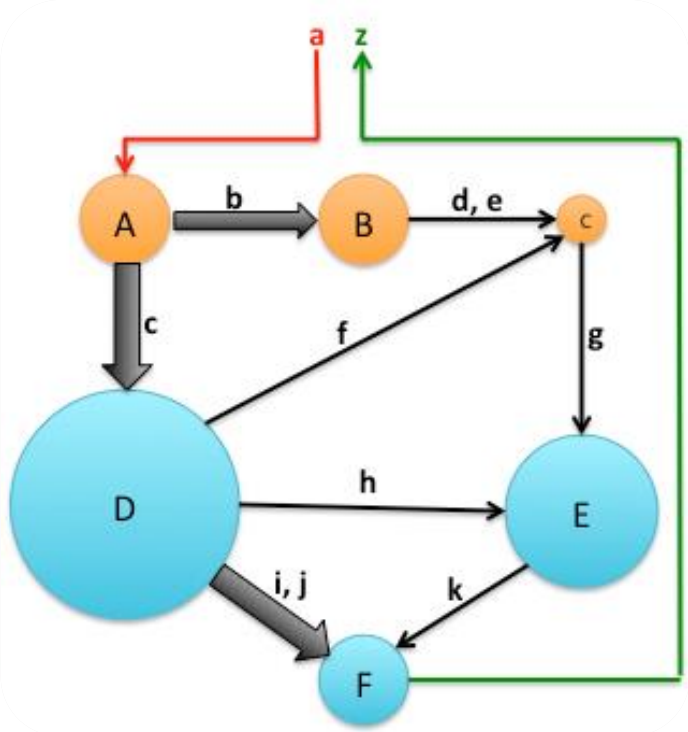
Offloading decision

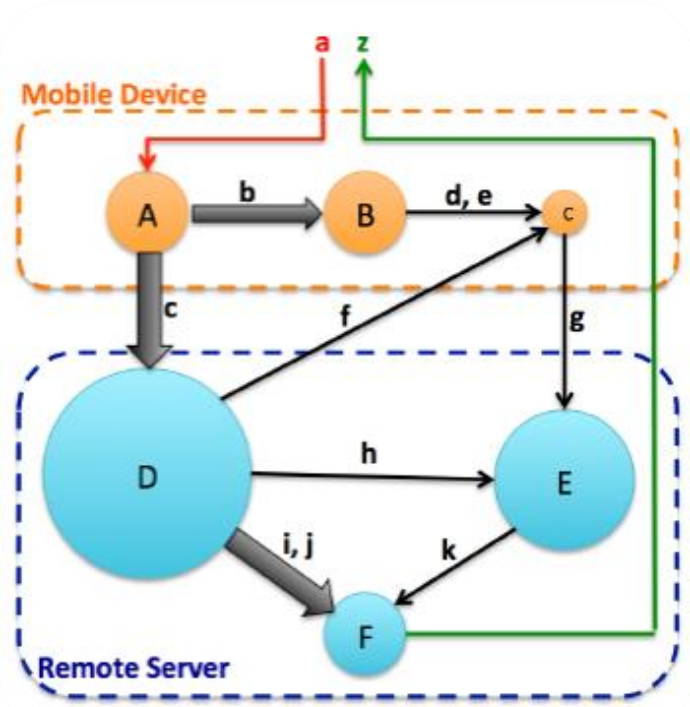
- Which components to offload, and where?

- 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

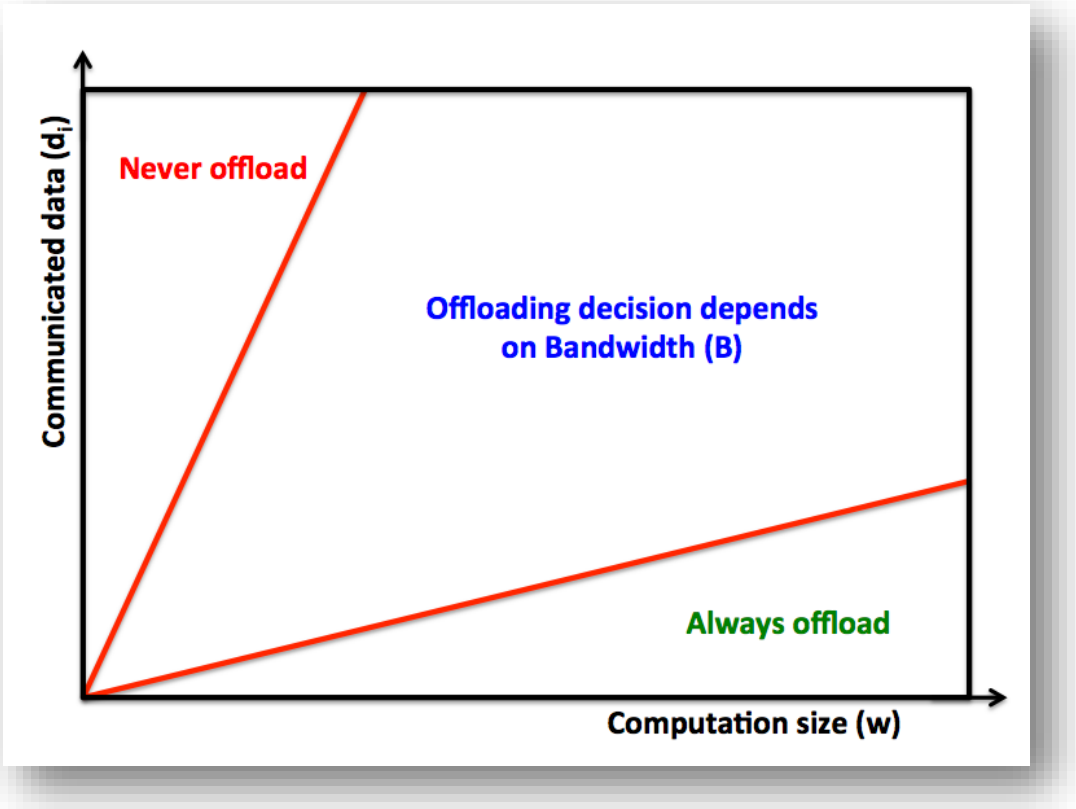
Offloading decision: partitioning





Offloading decision: bandwidth

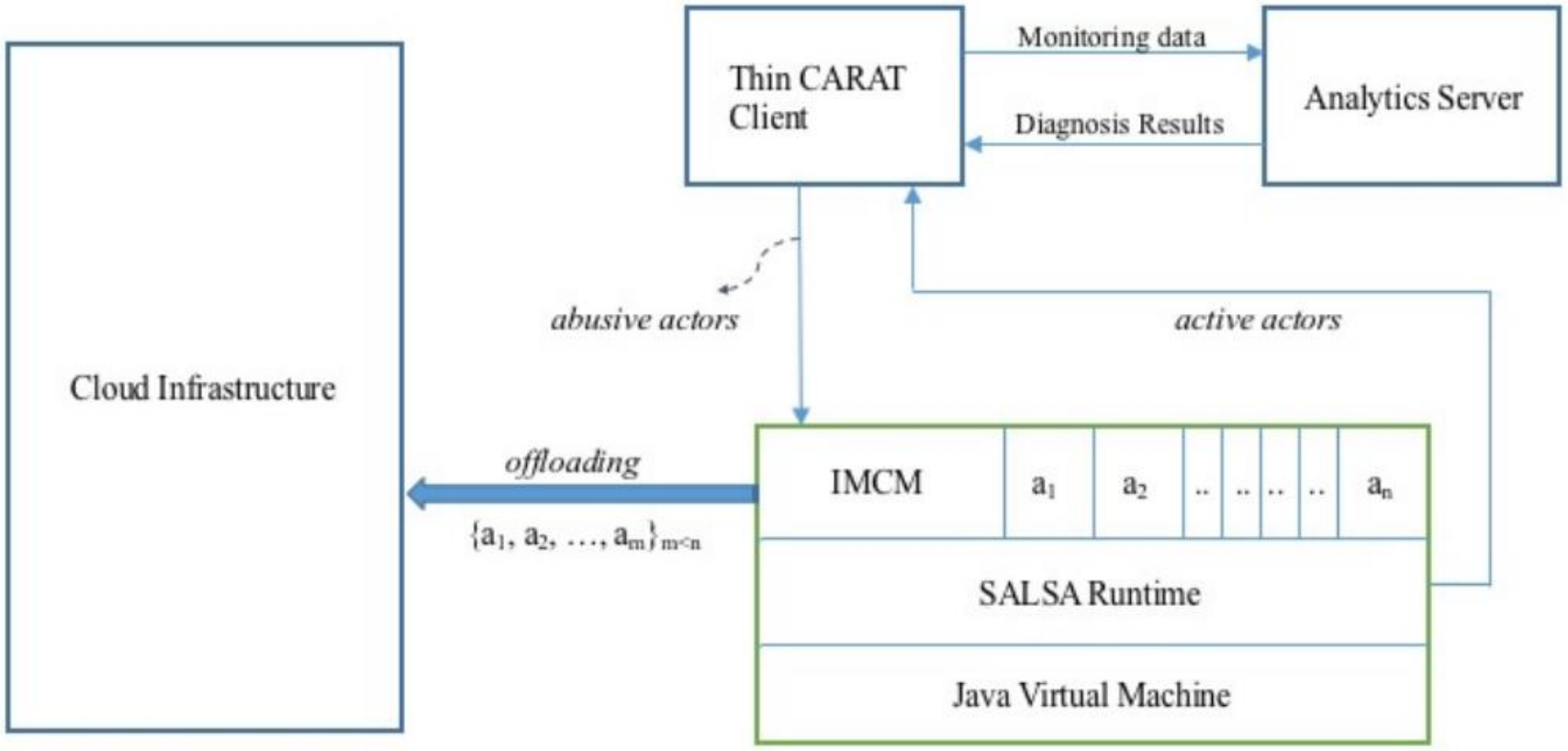
$$\frac{w}{S_m} > \frac{d_i}{B} + \frac{w}{S_s} \quad \Rightarrow \quad w * (\frac{1}{S_m} - \frac{1}{S_s}) > \frac{d_i}{B}$$



Offloading decision: energy use

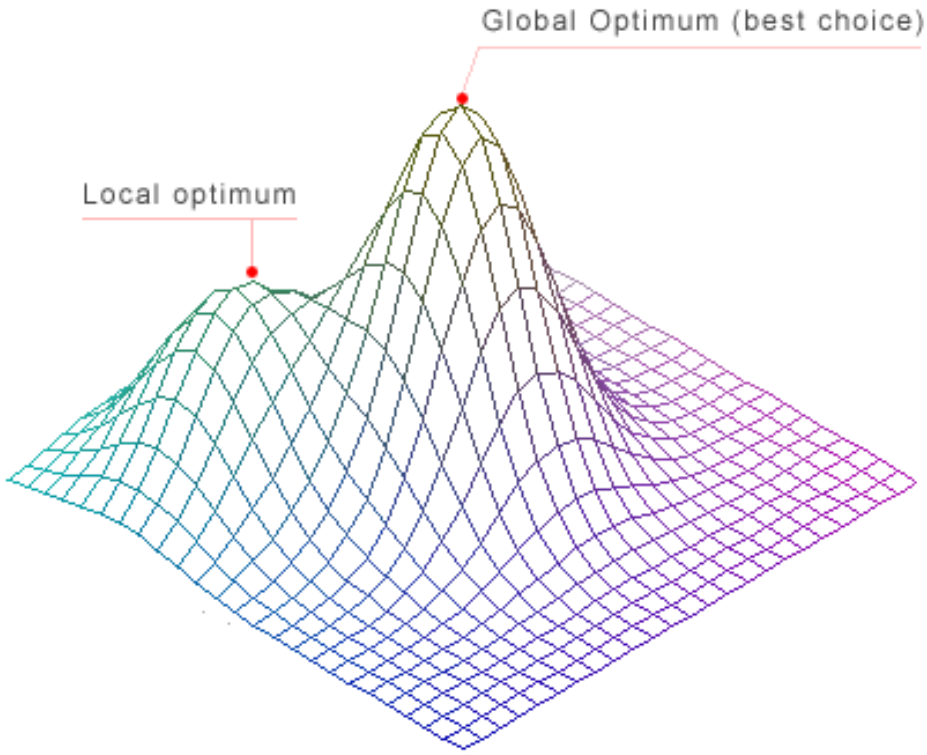
- Extend mobile profiling framework CARAT to actor-level monitoring
- Track events (actor message executions)
- Attribute overall energy use to particular components
- Heuristic: ignore low energy use actors

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

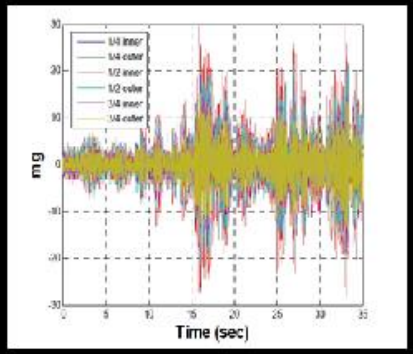
- Joint work with Ehsan Khamespanah (U. of Tehran) and Marjan Sirjani (Reykjavik University)

Siebel Center staircase demo

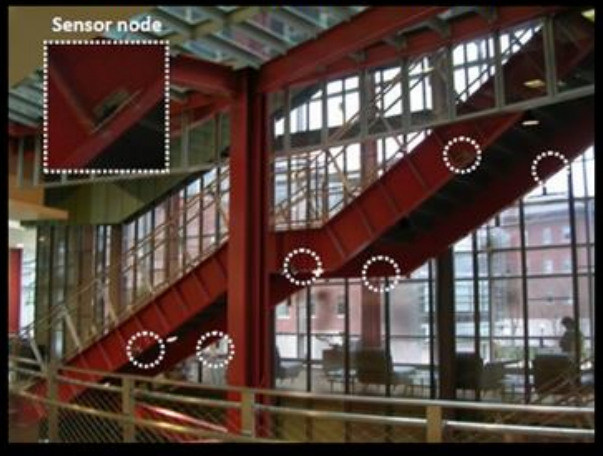
ILLINOIS STRUCTURAL HEALTH MONITORING PROJECT
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
OPEN SYSTEMS LABORATORY & SMART STRUCTURES TECHNOLOGY LABORATORY

Autonomous Structural Health Monitoring System

Camera image is intentionally blurred to protect privacy.



Live data stream from vibration monitoring sensors.



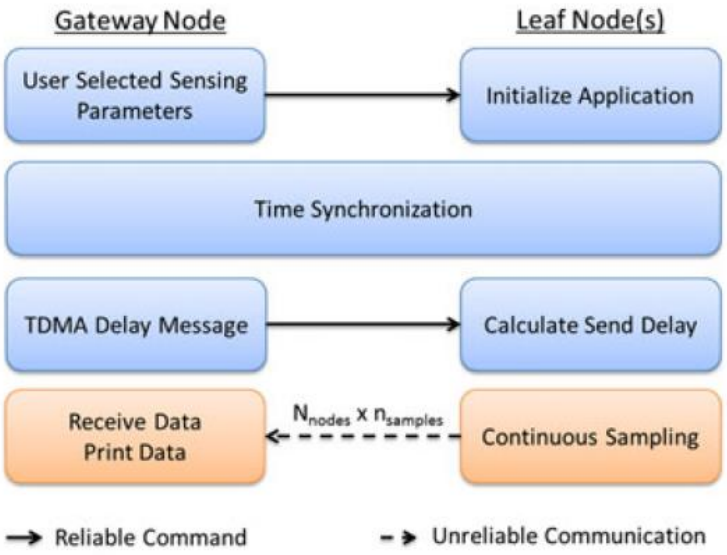
The **Illinois Structural Health Monitoring Project** pioneers the use of densely deployed smart wireless sensors for long-term continuous monitoring of civil infrastructure.

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.

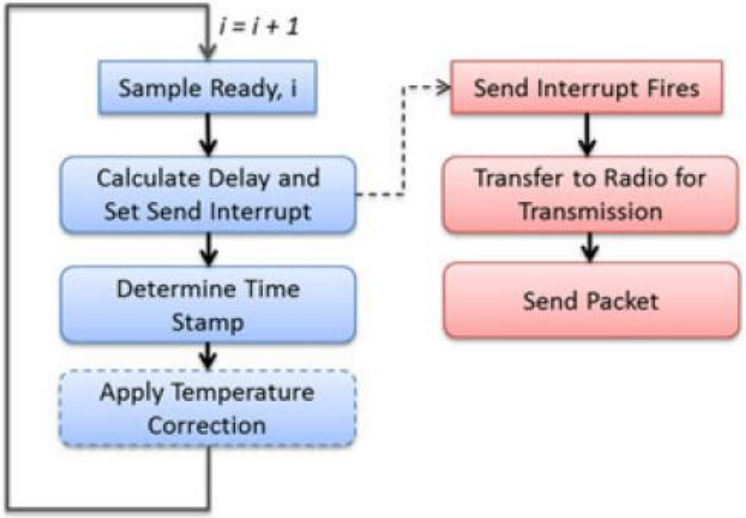
Continuous real-time sensing app

- 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



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

- 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

Problem	Size	Using BFTTS		Using timed automata		Using McErlang	
		#States	Time	#States	Time	#States	Time
Ticket Service	1 customer	8	<1 s	801	<1 s	150	<1 s
	2 customers	51	<1 s	19M	5 hours	4.5k	3 s
	3 customers	280	<1 s	–	>24 hours†	190K	5.1 min
	4 customers	1.63K	<1 s	–	>24 hours†	>4M‡	–
	5 customers	11K	<1 s	–	>24 hours†	>4M‡	–
	6 customers	83K	2 s	–	>24 hours†	>4M‡	–
	7 customers	709K	3 min	–	>24 hours†	>4M‡	–
	8 customers	6.8M	9.7 hours	–	>24 hours†	>4M‡	–
Sensor network	1 sensor	183	<1 s	–	>24 hours†	>6.5M‡	–
	2 sensors	2.4K	<1 s	–	>24 hours†	>6M‡	–
	3 sensors	33.6K	1 s	–	>24 hours†	>6M‡	–
	4 sensors	588K	13 s	–	>24 hours†	>6M‡	–
Slotted ALOHA protocol	1 interface	68	<1 s	–	>24 hours†	153K	1.8 s
	2 interfaces	750	<1 s	–	>24 hours†	>2.8M‡	–
	3 interfaces	7.84K	1 s	–	>24 hours†	>2.8M‡	–
	4 interfaces	45.7K	6 s	–	>24 hours†	>2.8M‡	–
	5 interfaces	331K	64 s	–	>24 hours†	>2.8M‡	–

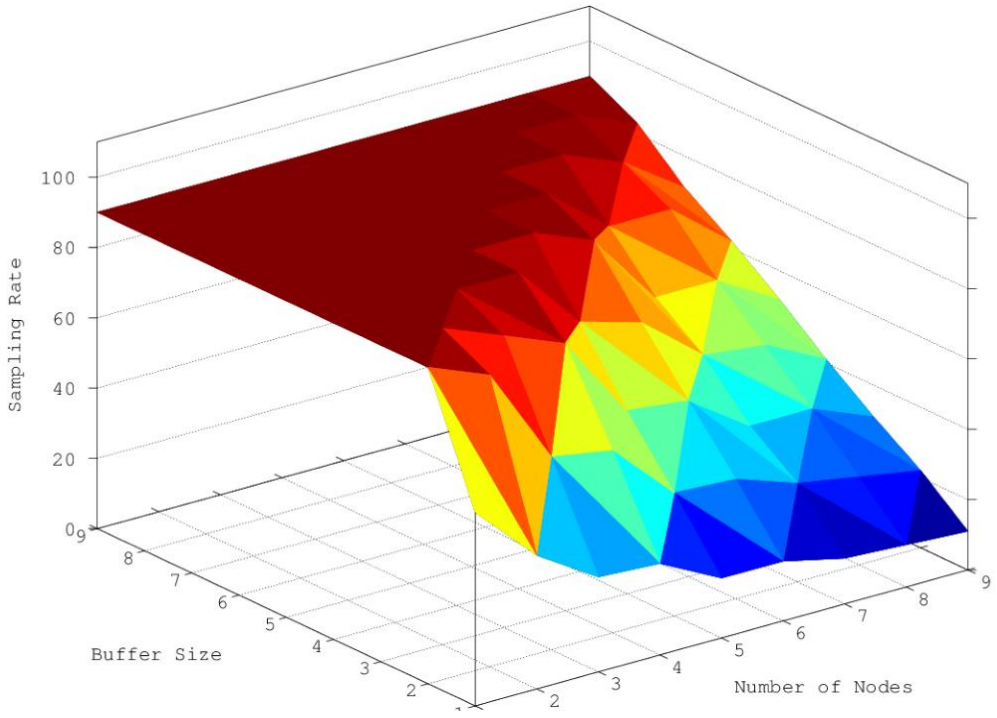
Example: simple TR model

- Ticket service

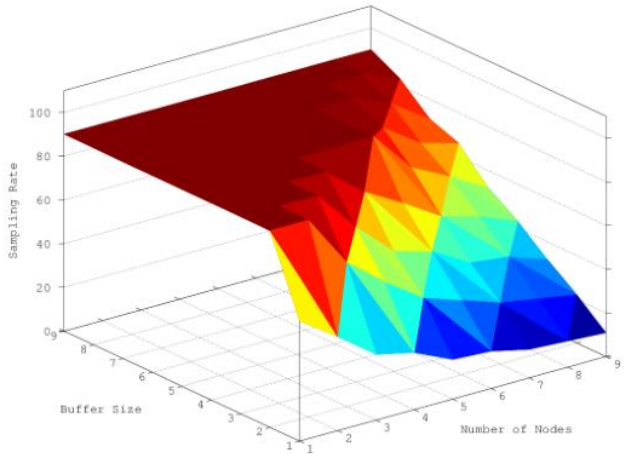
```
 1 reactiveclass TicketService {
 2   knownrebecs {
 3     Agent a;
 4   }
 5   statevars {
 6     int issueDelay;
 7   }
 8   msgsrv initial(int myDelay) {
 9     issueDelay = myDelay;
10   }
11   msgsrv requestTicket() {
12     delay(issueDelay);
13     a.ticketIssued(1);
14   }
15 }
16
17 reactiveclass 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.requestTicket();
40   }
41   msgsrv ticketIssued(byte id) {
42     self.try() after(30);
43   }
44 }
45
46 main {
47   Agent a(ts, c):();
48   TicketService ts(a):(3);
49   Customer c(a):();
50 }
```

Example: sensor network optimization

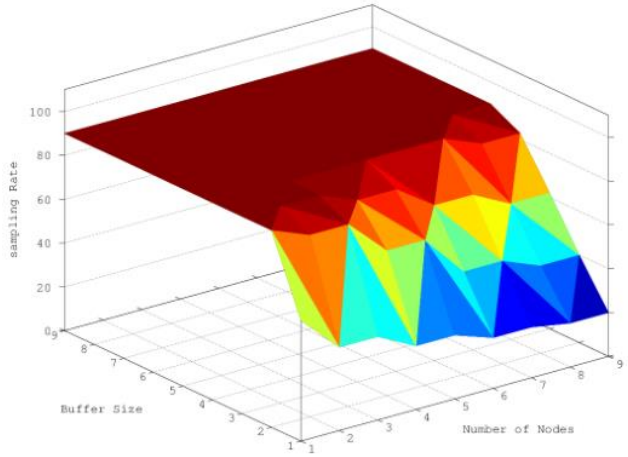
- Explore parameter space to find optimal configurations



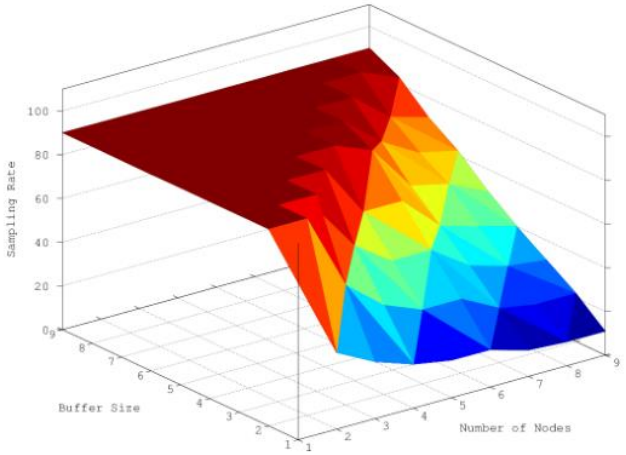
Example: protocol comparison/evaluation



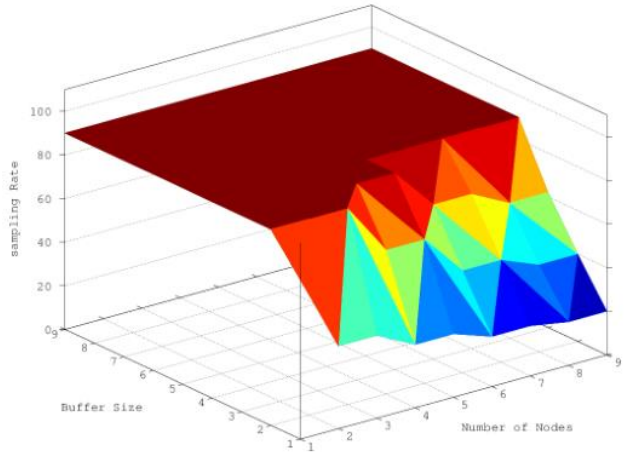
(a) TDMA, Sensor task delay is 5ms



(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?