

Initial investigations into macroprogramming

Programming across massive aggregates

Matt Welsh

Intel Research and Harvard University

mdw@eecs.harvard.edu

Eric Brewer

UC Berkeley

brewer@cs.berkeley.edu

Macroprogramming: The basic idea

Can we design a high-level language to express **aggregate programs** across a large, volatile field of motes?

- Examples: contour finding, object tracking

Current programming models are **node centric**

- Focus entirely on individuals, rather than behavior of the aggregate
- Want to program the “whole system”

Current programming models are **too low-level**

- Scientists don't want to think about gronky details of radios, timers, battery life, etc.
- Like writing Linux by toggling switches on a PDP-11
- Evidence: Huge engineering effort for each demo

Key goals

Yield high-level, expressive programming environment

- Express **sets** and **aggregate operators** across them (reductions, restrictions, etc.)
- Expose enough structure to permit locality

Deal with highly volatile nature of sensor networks

- Nodes coming and going, frequent failures, lossy radios

Optimize for communication and energy efficiency

- **In-network processing** to reduce comm. overhead
- Data/computation layout to maximize battery lifetime

TinyDB/TAG is one approach to high-level sensor queries

- SQL-like language for spatial/temporal queries
- We'd like to generalize on this approach
- Incorporate actuation, adaptivity, and control
- Compile down to fast machine code (rather than heavyweight query interpreter)

Example: Data-parallel set language

```
// Average reading of all sensors over some threshold
```

```
// ©2003 Sam Madden, Wei Hong, and Joe Hellerstein, all rights reserved
```

```
function avg_reading_over_thresh(THRESHOLD) =  
  let rs = { r : r in Readings | r >= THRESHOLD };  
  in (sum(rs) / #rs);
```

```
// Compute contour of readings on either side of threshold
```

```
function compute_contour(THRESHOLD) =  
  let aboveset = { i : i in index(#Readings) | Readings[i] >= THRESHOLD };  
  belowset = { i : i in index(#Readings) | Readings[i] < THRESHOLD };  
  boundary = { { (i,j) : j in intersection(belowset, nset) } :  
              i in aboveset;  
              nset in Neighbors->aboveset };  
  contourpoints = { (Locations[i], Locations[j]) : (i,j) in boundary };  
  in contourpoints;
```

Built-in variables, implemented by runtime:

- **Readings** - Set of current mote sensor readings
- **Locations** - (x, y) location for each mote
- **Neighbors** - (Delaunay) triangulation of one-hop radio neighborhood graph

Much to do...

Interpolation of sensor values

- Define overlay set in space (e.g., virtual grid)

Temporal operations and aggregation

- Triggers/event-driven operations a la TinyDB
- Sample over time

Express fidelity and uncertainty of data

- Specify timeouts, periodic execution, etc.
- Aggregate operators take a timeout, report **yield**
- Permit **lossy programming**
- Errors/uncertainty (i.e., interpolation) exposed to programmer

More information

Just e-mail me! `mdw@eecs.harvard.edu`