Macrodebugging: Global Views of Distributed Program Execution

T. Sookoor, T. Hnat, P. Hooimeijer,
W. Weimer and K. Whitehouse

Dept. Computer Science, U. Virginia

The 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009)

Macroprogramming

- A single macroprogram $\rightarrow$ Several microprograms
- No support for debugging
- MacroLab
  - A macroprogramming framework that offers a vector programming abstraction similar to Matlab for Cyber-Physical Systems (CPSs).

MDB

- Like GDB
- “Macrodebugging” on a single machine
- Steps through a macroprogram in a sequential order
- Implemented for MacroLab

Example of MacroLab program

- Lines 1-4
  - Initializes
    - The motes vector of node IDs
    - The magSensors vector of magnetometer sensors
    - The magVals macrovector of magnetometer readings
    - neighborMag, n x n neighbor reflection vector
- Line 7
  - Reads from all magnetometer values
- Line 8
  - Creates an active vector with the IDs of all nodes that have at least 3 neighbors with values > THRESH

```
1  motes = RTS.getMotes('type ', 'tmote ')
2  magSensors =
       SensorVector(motes , 'magnetometer ')
3  magVals = Macrovector(motes)
4  neighborMag = neighborReflection(motes ,
       magVals)
5  THRESH = 500
6  every(1000)
7    magVals = magSensors .sense ()
8    active = find(sum(neighborMag >THRESH , 2)
       > 3)
9    maxNeighbor = max(neighborMag , 2)
10   leaders = find($\ldots$
11       maxNeighbor(active) == magVal(active))
12   focusCameras(leaders);
13 end
```

Example of MacroLab program (cont.)

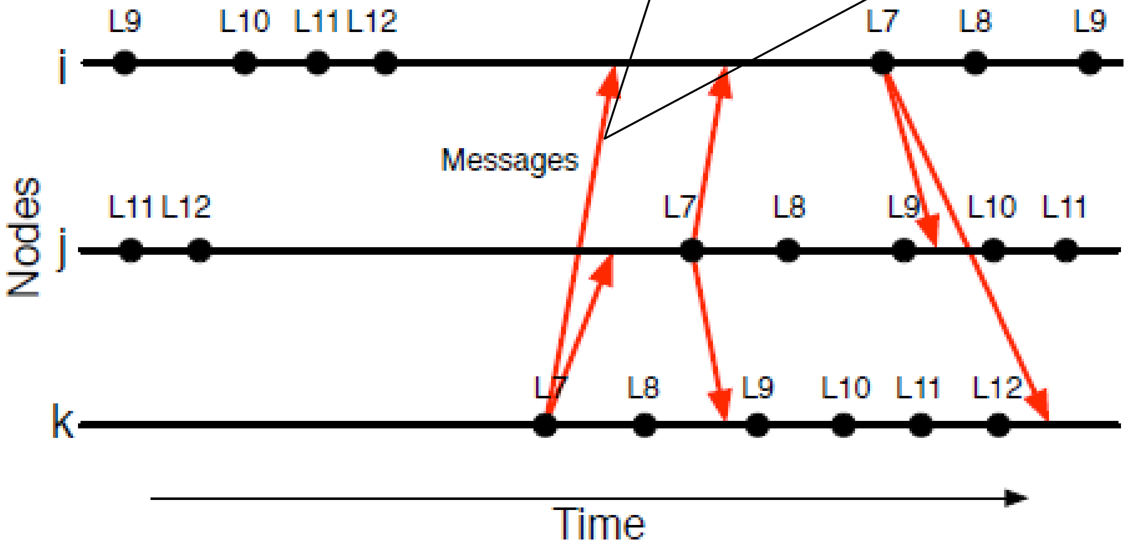
- Line 9
  - Creates a maxNeighbor vector with the highest sensor value in neighborMag
- Line 10-11
  - Creates a leaders vector with the IDs of those nodes having the highest sensor value in an active neighborhood
- Line 12
  - Focus all available cameras on the leader nodes

```
1  motes = RTS.getMotes('type ', 'tmote ')
2  magSensors =
       SensorVector(motes , 'magnetometer ')
3  magVals = Macrovector(motes)
4  neighborMag = neighborReflection(motes ,
       magVals)
5  THRESH = 500
6  every(1000)
7    magVals = magSensors .sense ()
8    active = find(sum(neighborMag >THRESH , 2)
       > 3)
9    maxNeighbor = max(neighborMag , 2)
10   leaders = find($\ldots$
11       maxNeighbor(active) == magVal(active))
12   focusCameras(leaders);
13 end
```

Example of MacroLab program (cont.)

- All nodes will execute the program asynchronously and independently

Magnetometer values automatically get reflected to neighboring nodes and populate the caches of neighborMag vector

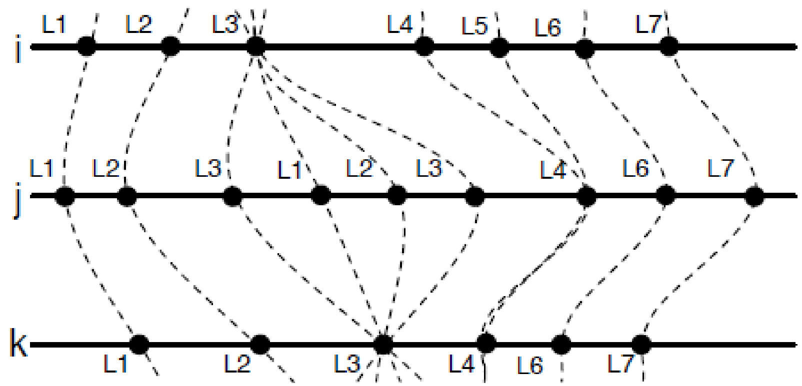


Three types of macroprogramming bugs

- Logical Error
  - active = **find**(**sum**(magVals>THRESH , 2) > 3)
- Configuration Error
  - Line 6, 1000ms might be too big
- Synchronization Error
  - Message loss, data races, asynchronous execution

MDB UI – Logically synchronous views

- Commands
  - lbreak(l)
    - Place a breakpoint at line l
  - lcont
    - Move forward to the next breakpoint
  - lstep[(l)]
    - Increment to the next line (or step l lines)
- Limitation
  - Views of distributed state may include values that correspond to different points in time



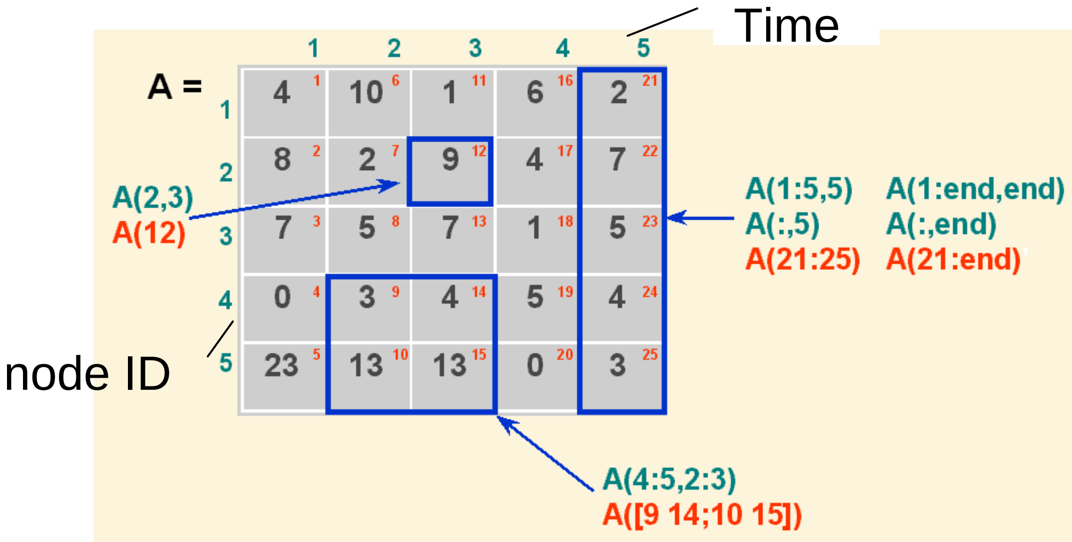

```
while( magVals < 10 )
  magVals = magSensor.sense()
end
if magVals > 3000
  magSensor.calibrate()
end
Display(magVals)
```

MDB UI – Temporally synchronous views

- Commands
  - tjump(t)
    - Change the state of the system to time t μs
  - tstep[(t)]
    - Change the state of the system to next logged time (or current time + t)
- Limitation
  - The user must be able to specify the exact times of interest

MDB UI – Historical Search

- Access the history of a macrovector by adding a new dimension to it
  - magVals(5, 1000)
- Standard Matlab operators can be applied to macrovectors
  - find(numel(leaders(:, :)) > 1)

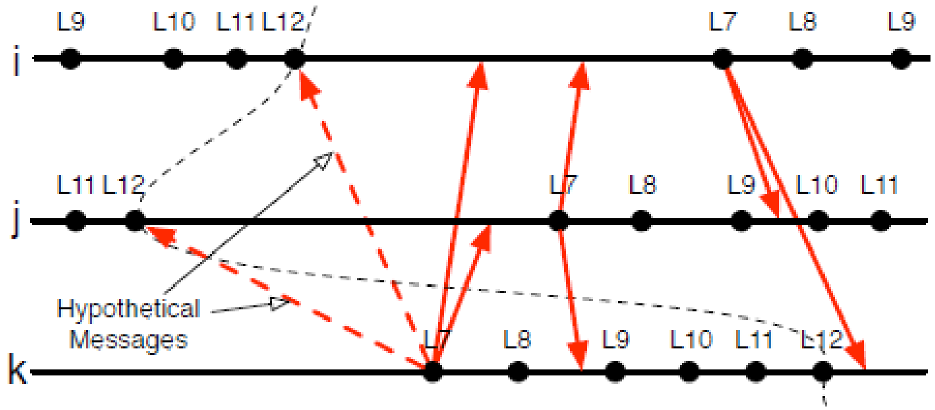


MDB UI – Hypothetical Changes

- Base timeline (bt)
  - The original execution trace that was collected during program execution
- Alternative timeline (at)
  - Generated by applying hypothetical changes to the base timeline
  - at = alt('magVals = 0', 7, bt)
- Four hypothetical changes

MDB UI – Hypothetical Barrier

- Barrier
  - A point in the source code that all nodes must reach before any node can proceed
- hb = alt('barrier()', 12, bt)
- message re-ordering

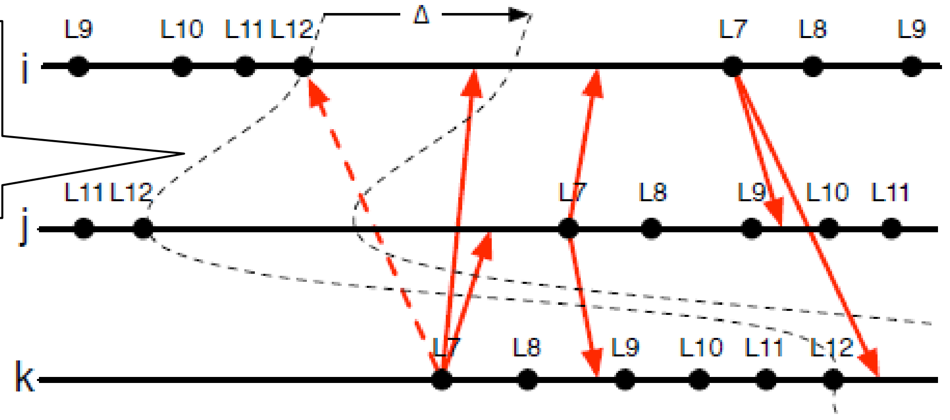


(a) Hypothetical Barrier

MDB UI – Hypothetical Time Delay

- Produces the distributed state that would have been produced if a time delay of Δt were inserted at a particular point in the macrocode
- dt = alt('deltat(10)',12, bt)
- message re-ordering

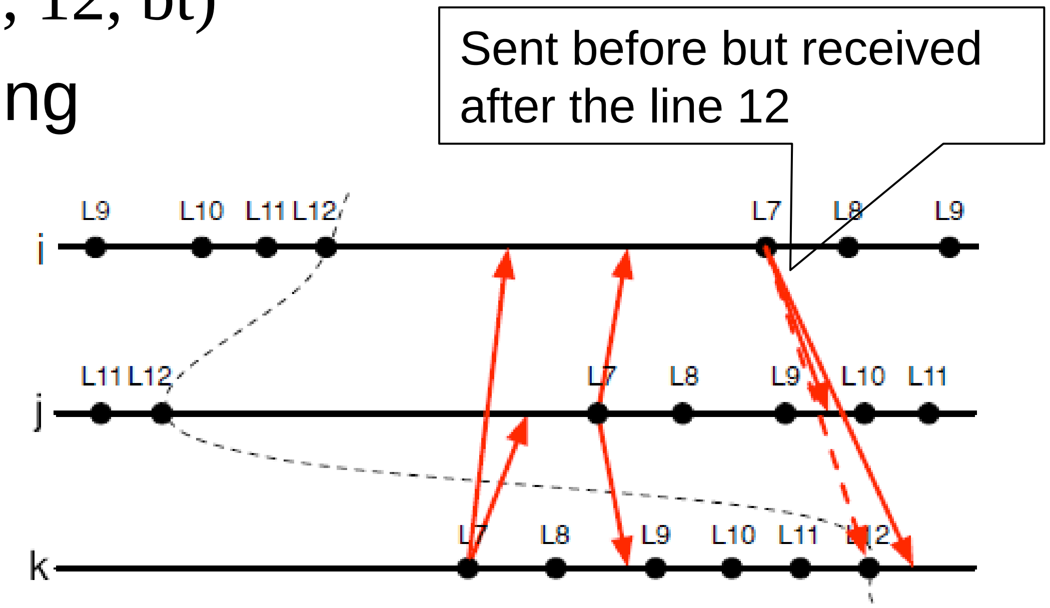
MDB first creates a logically synchronous view on line12



(b) Hypothetical Time Delay

MDB UI – Hypothetical Cache Coherence

- Shows the hypothetical distributed state if all caches were coherent at a give time
- cv = alt('coherent()', 12, bt)
- message re-ordering



(c) Hypothetical Cache Coherence

MDB UI – Hypothetical Cache Expiration

- Shows the hypothetical state that would result if cache expiration were used at a given line of code, without a particular expiration time
- ev = alt('expire(100000)', 12, bt)
  - ev stores the last values written to each element of the macrovector in the time interval
  - ev = NaN if an element had no value written to it within the time interval

MDB Execution Traces

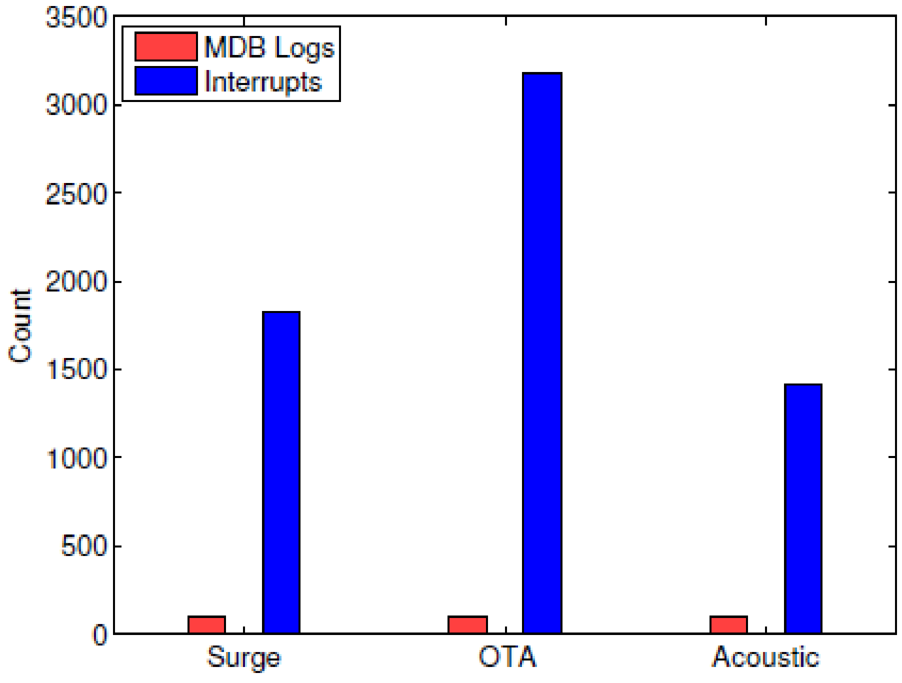
- Post-mortem debugger
  - Allow the user to inspect program execution after the logs are retrieved
- Data traces
  - Log entry
    - program counter, variable location, etc.
- RAM $\rightarrow$ external flash
  - Advantage
    - Reduces contention for the CPU
  - Disadvantage
    - Log entries in the RAM buffer may be lost if the node crashes

Distributed Timekeeping

- Causal consistency
  - Any event E that cause event E’ must have smaller timestamp than E
- Lamport algorithm
  - Any events on the sender have an earlier timestamp than events they might cause on the recevier
  - Used off-line after the logs are collected
- One Message every two minutes

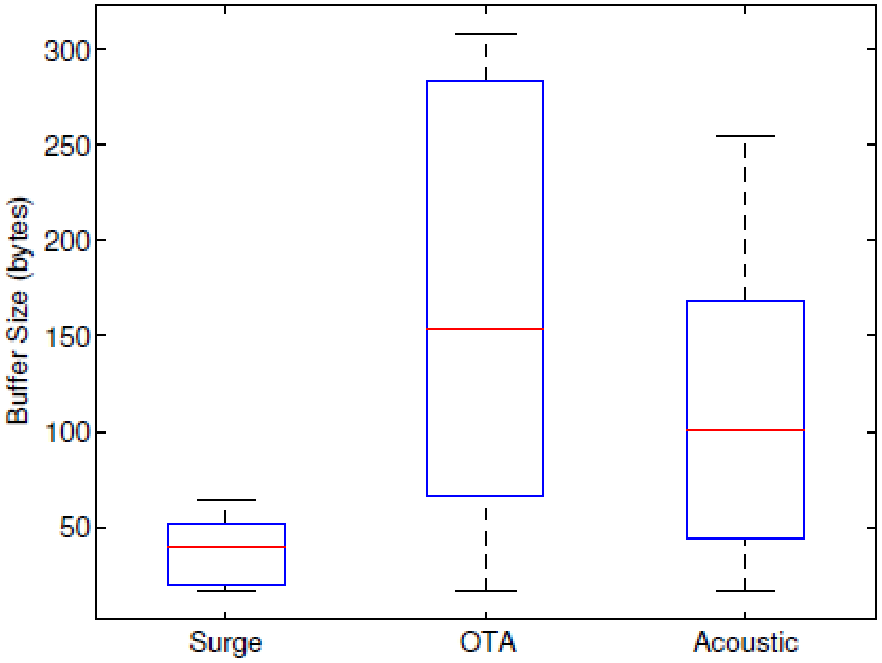
Data vs. Event Logging

- Data logging
  - Number of logging statement
- Event logging
  - Number of interrupts
- Testbed
  - 21 Tmote Sky nodes with photoresistor sensors



RAM overhead

- MDB has modest RAM requirements
- MDB needs to store a maximum of 304 bytes of data



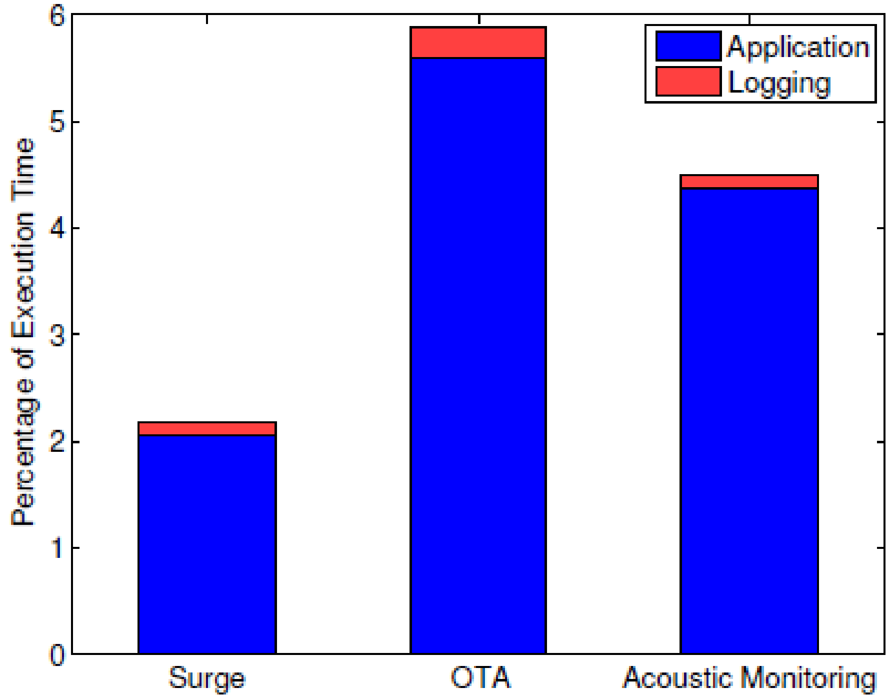
Flash overhead

- The applications store less than 300 Bps to the flash
- Tmote Sky has 1MB of external flash
- Approximately,
  - 9 hour logs for Surge

Application	Flash (Bps)	Wraparound (hr)
Surge	31	9
Accoustic	187	2
OTA	288	1

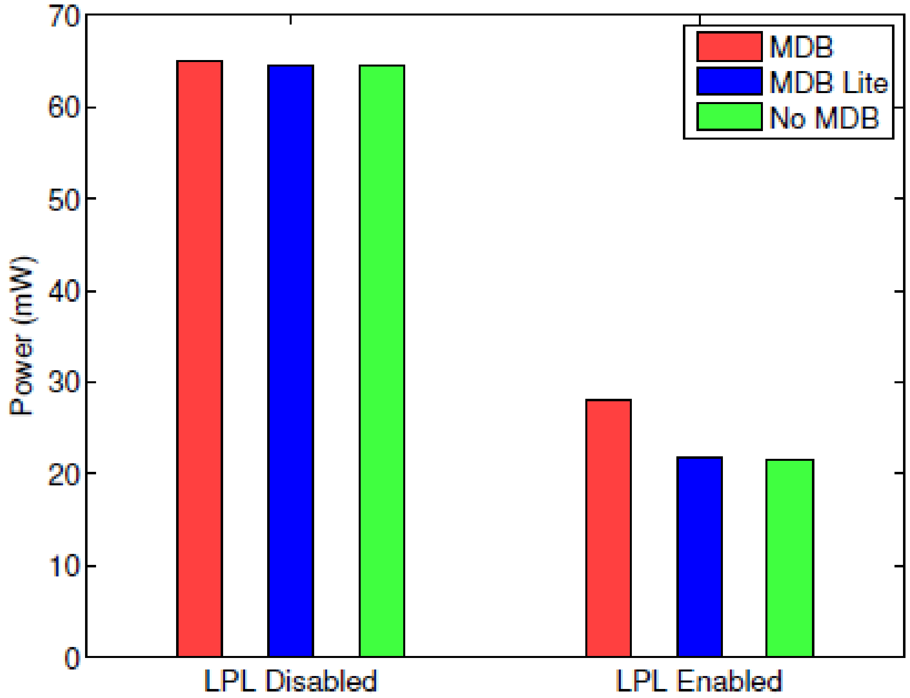
CPU overhead

- Count the logging instructions executed during a particular run
- Run the applications on Cooja simulator
  - 1 node for Surge
  - 5 nodes for Accoustic
  - 10 nodes for OTA
- Logging code executes for less than 0.5% of the total execution time



Energy Consumption

- Test application
  - OTA
- When Low-power listening (LPL) is enabled
  - With MDB
    - Consumes 30% more energy comparing to no-MDB
  - With MDB-lite
    - Consumes 0.9% more energy comparing to no-MDB



END

- Q&A