Application Architectures for mobile and IoT applications

1. Background:
   a. Looking for paper on mobile operating systems
   b. Read papers on:
      i. Offloading code from phones to servers to save energy
      ii. Predicting how much time a server has left to respond to a request
      iii. Tracking information flow through Android apps
      iv. Performance monitoring (profiling) mobile applications to understand bottlenecks, delays
      v. Speculation to draw the next frame of a game while the server is still processing what you did, rather than wait to actually see
   c. What this paper offers:
      i. Issues in writing applications that integrate information from multiple places
      ii. Recap of how operating systems people solve problems.

2. Motivating hardware trend:
   a. IoT mentality / devices
      i. Lots of sensors in various places: What are all the ways a computer system can learn about what is happening
         1. Smart sensors in home (thermostat, motion, camera)
         2. Sensors on our computers and phones (microphone, camera, accelerometer)
         3. Sensing of activating by the operating system
            a. What programs are running
            b. How often is someone interacting
      b. Ubiquitous but imperfect connectivity
      c. Cloud computing to run services

3. Problem statement:
   a. How abstract sensors to make it easier to write applications that sense and respond to real world?
   b. Why is it a problem?
      i. Physical devices: how do you know what devices are relevant?
      ii. Mobility: set of physical devices may change over time
      iii. Inferring activity: raw devices produce low level output, may want to use it to infer higher-level property
         1. Example: image recognition from a camera, voice detection from a microphone, activity from an accelerometer
      iv. May want to infer different things from same device
         1. Microphone for voice recognition but also ambient noise levels
      v. May want to infer things from inferred data
1. Combine multiple inferences, such as predicting activity based on usage of multiple devices

4. Operating-systems solution
   a. Number one: build an abstraction
      i. **Inference engines**: compute services that can run inference code
         1. E.g.: home PC, phone, cloud service
         2. Properties: can run inference modules. Implies portable code
      ii. **Inference data units**: abstract data representation to do inference over
          1. E.g.: temperature reading at a time
          2. Properties: has a type, a time, a statement, and an error level
      iii. **Inference modules**: algorithms to actually do inferences: based on inputs, make some inferences that is output
           1. Properties: timeliness: on demand or scheduled
      iv. **Channels**: communication channel to connect elements together to pass IDUs.
         1. Can be local (fast) or remote (slow)
         2. Are asynchronous and unreliable (important to be in abstraction)
      v. **Sensors**: something capable of producing a stream of IDUs
      vi. **Coverage tags**: Label describing what sensor is sensing
          1. Can be a person (if mobile)
          2. Can be a place (e.g., work or home)
          3. Examples: computer can sense who is using it, set tag on anything it produces. Phones can always sense the user, not a location.
             Fixed-location computers or sensors can always sense a location
             a. What else? Subset? Proximity?
      vii. **Inference graph**: set of connected sensors, modules, and applications
   b. Number 2: decide on operations
      i. Selecting a device: knowing what devices are relevant to an application
         1. Solution: application specifies tags, looks for devices with similar tags
         2. Uses discovery mechanism to enumerate available devices in local settings (Home OS)
      ii. Assigning tags:
         1. Tracking services assigns user tags as it senses co-location of a user with a sensor
            a. Can be presence (e.g. GPS)
            b. Can be activity (e.g. logins)
            c. Can use inference itself to make assignment
      iii. Executing a graph
         1. Central problem: where to execute inference modules
         2. QUESTION: what are considerations?
a. Modules input more data than they output (e.g. processing raw events), so want to run closer to producer than consumer

b. May be multiple consumers of an output, so want to avoid transmitting multiple times

3. SOLUTION: use a tool that solves the problem for you: optimization
   a. Set up constraints:
      i. Each engine runs in one place (sum over Pid = 1)
      ii. Set a minimization goal (sum over Cd1,d2 = communication)
   b. Measure current data rates (visible from channel abstraction)
   c. Reoptimize periodically based on actual data rates

4. Optimization:
   a. Combine multiple channels with a new artificial proxy node (“scatter node optimization”)
   c. Niceties – making it work in the real world
      i. Disconnection tolerance
        1. Challenge: decide on what you want
           a. When disconnection detected?
           b. When reconnect?
        2. Basic technique: all communication through coordinator
           a. Coordinator is highly available (e.g., cloud), highly provisioned (e.g., lots of storage)
           b. Buffers some amount of input.
           c. QUESTION: how much?
        3. BEAM solution: drop-tail queues
           a. When queues fill, leave data in queues and drop new data
           b. When reconnect, see lots of old data instead of most recent

4. Question: is this the right policy

5. General thoughts
   a. Tolerating heterogeneity
      i. Years of people implementing “compatibility layers” on top of heterogeneous software to make it look better
      ii. Early approach to standardization – implement a standardization layer
   b. Abstraction
      i. How, in general, design an abstraction?
      ii. Example: IDUs are largely untyped – lots of flexibility, but hard to use
      iii. Channels are unreliable – hard to program, but easier to implement

6. Evaluation: how evaluate a system like this?
   a. What are the right metrics?
      i. Performance: of what?
1. Channel delivery?
   a. Does it matter
2. Optimization
   a. Graphs are small - does it matter that much?
ii. Programmability: what are the metrics?
   1. Lines of code
      a. How count?
      b. How evaluate across languages?
   2. # of components to write?
   3. Time for a novice, median, sophisticated programmer to do something?
   4. User study?
iii. What is the comparison?
   1. Need to implement applications in multiple styles for comparison
iv. Optimization: how evaluate?
   1. Idea 1: look at all possible partitions, show the selected one works best
   2. Idea 2: compare to “oracle” – or, what do humans do
b. Another approach: demonstration
   i. Example: device selection. Cannot evaluate that it works in general very easily
   ii. Instead: create one or more complex scenarios, show that it works
   iii. General approach to evaluation: pick a scenario where naïve system does the wrong thing, show that your system does the right thing
iv. QUESTION: how convincing is this?