Each Unto Their Abilities...

Low-Power Computing
David Andersen
Carnegie Mellon University

Sources:

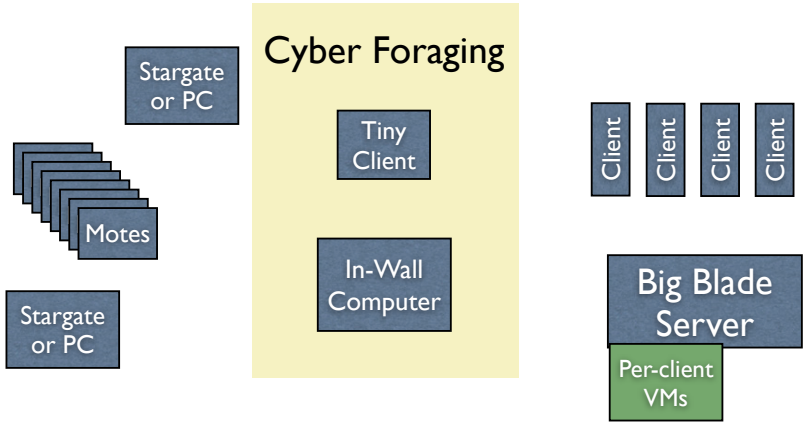
Balancing Performance, Energy, and Quality in Pervasive Computing (Flinn, Park, Satyanarayanan)
A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices (Goyal and Carter)
Simplifying Cyber Foraging for Mobile Devices (Balan, Gergle, Satyanarayanan, Herbsleb)
Transient Customization of Mobile Computing Infrastructure (Wolbach, Harkes, Chellappa, Satyanarayanan)

Strategies for power

- More efficient systems
- Do less work (aggregation; interval sampling)
- Trade speed for power (DVFS, etc)
- Sleep more
  - Consolidate -> deeper sleeping
- Scavenge power from the environment
- *Make someone else do the work for you*
  - Tiered systems; Cyber Foraging

Tiered Sensors

Thin-Client Computing



Why not pure cloud?

- "After all, Google runs in the cloud"
- But some applications require interactivity
  - HCI-ish things (games, graphics, sound & speech, GUIs, etc) - humans start to notice 10s of ms response time
- Something nice: Partition so that "heavy lifting" happens in cloud, "fast response" happens on client

Cyber Foraging

- Ex1: Speech recognition on a handheld (HCI)
- Ex2: <N> recognition on a <wimpy dev>
  - "Smart glasses" (tell you who you're talking to)
- Language Translation (DARPA projects galore)
- Augmented Reality
  - Ex: Boeing uses AR goggles for airplane wiring
- Supercomputing - process (reduce) data near source before sending across (slow) network

Mobile Computing + "The Cloud"

- Computation everywhere - but your devices are battery-limited
- Offload the "heavy lifting"
- Two sets of design constraints

Desirable Outcomes

- Battery Lifetime
- Good application quality (bitrate, latency, etc.)
- High performance

Big Picture

- *Locate* a compute server {advertise capabilities & resources, etc.}
- *Trust* the compute server
  - *Account* for resources consumed
- *Isolate* the client code safely
- *Partition* the application
- *Transfer* the state + code
- *Collect* the results + new state
- And make it easy and transparent in a harsh (lossy, unreliable, etc) environment!

Resources

- What resources? There are many - often platform dependent (disk, CPU, memory, energy, network bw, ...)
- (And even that's an oversimplification - disk seeks? disk bw? etc.)
- Apps may not really know.
  - Spectra: Dynamically estimate
- Discovery: XML, SLP, etc. Similar problems exist, e.g., general cloud computing, planetlab, akamai, etc.
- Not clear whether or not this matters - maybe overprovision, maybe cloud is starved/heterogenous

"Traditional" approaches

- Middleware of various sorts
  - Spectra, Odyssey - provide standard runtime on remote system
- Virtual machines.
  - Some tried with JVMs
  - Most use "real" VMs - Xen, VMware, etc.

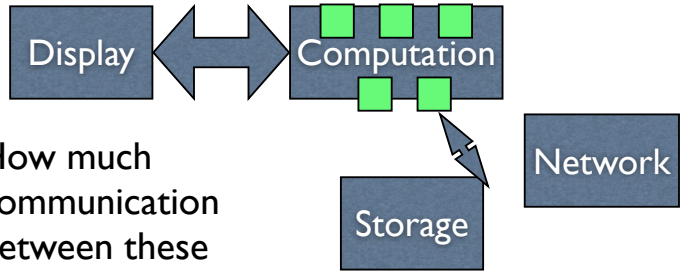
Sensor Decomposition

- Typically explicit
  - TENET -- Click-like configuration
  - Constrained functionality on motes (record, briefly summarize, report)
  - Arbitrary functionality on higher tier
  - could do same with TinyAgg, etc.

Middleware

- Modify the source code
  - Like RPC decomposition for most apps
  - Programmer picks functional decomposition
- + runtime (RPCs again) - but runtime decides which components to run where

Partitioning



- How much communication between these components?
- How sensitive to b/w and *latency*?

Granularity of Remote Execution

more opportunities for remote execution

amortizes overhead over larger execution unit

Spectra considers many factors to place objects

- **decision overhead not negligible**
- **targets coarse-grained remote execution**
- **remote operations typically >100 ms.**

Source Code Modification?

Can transparently partition applications:

- **use externally visible objects (e.g. Coign)**
- **If your program is written using nicely encapsulated objects**
  (Coign uses COM objects)

Problems:

- **can't support legacy applications**
- **best partitioning may not be visible**
- **lots of objects = expensive computation**
- **Spectra uses app hints to optimize computation (what's likely to matter)**

No source code changes?

- Can partition *display*
  - e.g., remote X11, VNC, etc.
- Move *entire* computation
- Thin client example does exactly this.
  - Display latency can be higher (where is the surrogate? What's your b/w to it?)
- But must be able to execute code...

Execution

- Java VMs?
- Real VMs! (Kimberley, WASCo, etc).
- Encapsulate entire computation and state needed for it
- Send to surrogate
- Run
  - And get the data back, please!

Transfer

- State (filesystem, memory, etc.)
  - Spectra: Use Coda (or other DFS)
  - Kimberley: Transfer entire VM
- Key: For fast response, binaries, system state, etc., *must* be pre-place-able at surrogate
  - VM caching, coda hoarding, etc.
- ISR & Kimberley approaches: Chunk-based or delta-encoding. ISR maintains full VM state - deltas can be large (100s of MB). Kimberley ditches state.
  - But, er, I modified data?

Kimberley Persistent State

- Question: Do you snapshot *all* state, or only *explicitly saved* state?
  - ISR did the former
- We do have abstractions for explicitly saved state: files on disk.
- "Export" a disk to the surrogate. Let it lock it, write it, and then copy just the disk back.
- Avoids transferring large amounts of transient/ unimportant state changes.

Trust 1: Evil Clients

- Don't let clients goof up the surrogate
- Don't let clients goof each other up
- Don't build the perfect botnet[1]
  - bw/computation/anonymity for attacks

[1] Sachin Goyal and John Carter, "Safely Harnessing Wide Area Surrogate Computing -or- How to Avoid Building the Perfect Platform for Network Attacks"

Solving Trust 1

- WASCo idea: Tunnel through the client if surrogate wants to access arbitrary nodes
  - Assumes client has connectivity...
  - Allow access to authorized nodes (other surrogates used by same client, etc.)
  - Allow external nodes to say "OK to contact me!"
- Plus: Traffic shaping & Rate Limiting (WASCo, PlanetLab, etc).
- Plus: "Don't talk to me!" lists
- Plus: "No contacting internal nodes" for some sites (CoDeeN policy for academic Planetlab sites)
- Plus: Logging, IDS, etc. Can probably afford to be aggressive about cutting off access - surrogate is "just" an optimization

Trust 2: The Infrastructure

- Kimberly example: viewing medical images on untrusted infrastructure.
  - *1) please don't show my spleen in a restaurant.* :)
  - *2) What about access credentials, private data, etc?*

Solving Trust 2

- Null hypothesis: "Trust us!"
- Proxy data access through client
  - Still leaves untrusted data on surrogate, potentially; uses more b/w
- Use trusted computing hardware to ensure that surrogates execute only client code
  - This is kind of easier said than done. :)