Application and System Adaptation for Mobility

CS 444N, Spring 2002

Instructor: Mary Baker

Computer Science Department

Stanford University

Examples of adaptation

- Voice traffic
  - Avoid link-level retransmissions: drop packets instead
- Moving within range of a cheaper network
  - Switch to new network as default interface
- Move to network that doesn't take transit traffic
  - Move to using bi-directional tunnel
- Move to slower network
  - Transmit B&W instead of color pictures, except maps
  - Ask TCP to recalculate MTU and RTT
- Batteries running low
  - Route through other nodes in ad hoc network
  - Transmit B&W instead of color pictures, except maps

Types of adaptation

- Application adaptation to environment
  - Application adapts to change in network speed
- System adaptation to environment
  - System brings up new interface as signal strength improves
- Application-driven system adaptation to environment
  - Application asks to use specific interface
  - Application asks to be notified about BW changes
- System adaptation to application
  - System notes an application is using TCP and selects Mobile IP for that flow
  - System notes application's need for power and adapts power scheduling accordingly

Network reconfiguration (Inouye)

- Assume hot swapping technologies
- Avoid per-packet monitoring
- Each network layer adapts as appropriate
- Enable this through cross-layer information passing
- “Desktop equivalence” (no more work for user than a desktop PC requires)

Device availability

- Device is available if it is…
- Present: physically attached, driver exists
- Connected: link-level connectivity
  - E.g. packets to GW
  - May be a continuum
  - May be boolean (PCMCIA cable example)
- Network named: has an IP address
- Powered: enough power to function correctly
- Affordable: use cost model
- Enabled: user can deliberately disable interfaces
  - PPP interface might be disabled by default

Represent interface with state machine

- State changes in response to
  - External events (card removal, signal strength changes, etc.)
  - Internal events
    - A result of external events
    - Guards trigger events
    - Example: Signal strength change causes route table change causes application to choose new interface

Guards and cross-layer notifications

- Guard for each interface characteristic
- Present: depend on hot-swapping support
- Connected: should be in device driver
  - Not all devices provide this information
  - Can monitor/probe
  - Avoid modifying all device drivers by implementing in network layer
- NetNamed:
  - ICMP router advertisements
  - Mobile IP beacons
  - DHCP servers
  - User input
  - Maybe trigger from changes in connectivity?

Implementation

- Device state machines in pmid daemon
  - On start-up pmid uses config files
  - Listens to guards
  - Periodically checks kernel interface info for consistency
- Pseudo device driver for communication between components
  - Passes events from OS to pmid
  - Provides interface through with apps register interest
  - Apps receive notifications via select call

Agility and fidelity (Odyssey)

- *Agility*: speed and accuracy with which system responds to changes in resource availability
- *Fidelity*: the degree to which data presented at a client matches the reference copy at the server
  - Note client/server-centric approach
  - What if if your primary copy is the physical world you are monitoring?

How transparent?

- Users:
  - User may observe changes in application fidelity
  - But user need not direct such changes himself
- Applications:
  - Application-aware adaptation
  - Each app independently decides how to respond to notifications
- System:
  - System monitors resource levels
  - Notifies applications of relevant changes
  - Enforces resource allocation decisions

Transparency trade-offs

- Laissez-faire approach
  - No system support
  - All responsibility placed on apps and user
  - No centralized support means concurrency is hard
- Odyssey
  - System support
  - Applications partner with system
  - Concurrency support
- Application-transparency
  - No apps need be modified
  - Limited support for diversity

Application adaptation model

- System should have no application-specific knowledge
- But too hard to do efficient resource management
- Instead, embody system "type-specific" knowledge in *wardens*
  - Sit on clients
  - Subordinate to type-independent *viceroy*
- Viceroy/warden interaction is data-centric
  - Defines fidelity levels for data types
  - Resource management policies
- Application/Odyssey interaction is action-centric
  - Provides apps with control over selection of fidelity levels supported by wardens

Evaluation of centralized resource management

- Modified viceroy to support laissez-faire resource management
  - Examine user-level RPC trace logs individually
  - Mimics what individual apps would discover
  - Information less accurate, but similar discovery times
- Blind optimism:
  - Notify apps when switching between network technologies (upcall to viceroy, viceroy to apps)
  - Less accurate: does not take into account other apps
  - Fastest discovery time
- Odyssey: central management best with BW competition

Energy adaptation

- Energy is a vital resource for mobile computers
- Traditional techniques aren't buying us enough
  - Advances in battery technology
  - Low-power circuit design
- Claim: higher levels of the system must help
  - Operating system
  - Applications
- Answer questions:
  - Will lower data fidelity save energy?
  - Can we combine technique with hardware power management?

Energy reduction techniques

- Applications dynamically modify their behavior
  - High fidelity when energy is plentiful
  - Reduction in quality when energy is scarce
- Combine with hardware power management
- Operating system control over adaptation
- Powerscope tool for profiling energy usage

Powerscope

- Like profiling CPU usage
- Find fraction of total energy consumed over time by specific processes
- Find energy consumption of particular procedures

Powerscope Technique

- First pass: statistical sampling of power consumption, process ID and program counter
- Digital multimeter samples current drawn from power source (voltage is constant)
- Second pass: use symbol table info to correlate samples with procedures

Video Example

- Requires support of proxy or server
- Lossy compression helps
- Energy proportional to display size
- With hardware and all techniques: ~35%
- Unfortunate news
  - Most of energy in idle loop
  - Rest in display

Speech Recognition Example

- Reduce fidelity through reduced vocabulary and less complex acoustic model
  - Saves CPU
  - Smaller memory requirements
- Accuracy still okay, since easier to choose from smaller vocabulary
- Local versus remote recognition, and hybrid
- Hardware only gives 33%: they turn off display!

Speech Recognition, continued

- Remote better than local – saves CPU
- Why does lowering fidelity in remote case help?
  - Speeds recognition time
  - Lowers time waiting in idle loop for reply
- Hybrid better than remote
  - More CPU, but bigger savings in network
- Overall 70-80% savings
- Savings structure is so application-specific!

Map Viewer Example

- Incorporates notion of “think time” in results
  - Display consumes energy while user views results
  - Energy consumption linear with respect to time
- Fidelity reduced through:
  - Filtering (less detail)
  - Cropping (smaller section of map)

Map Viewer, continued

- Hardware only: about 10-20%
  - Network idle during think time
  - Disk off throughout
- Combined techniques give up to 70% savings
- Little room for further software optimization: most energy spent in idle loop

Web Browser Example

- Again incorporates think time
- Fidelity reduction through lossy JPEG compression
- Results disappointing: up to 34% reduction

Concurrent Applications

- Is the energy greater or less than the sum of both applications?
- Could be greater due to resource fighting
  - Paging, for example
- Could be less if both applications use the same resource in non-interfering manner
  - Display already on for second app., for example
- Idle state could be used for second app., so energy spent there is useful

Concurrent Apps., continued

- Composite application (speech, web, map)
  - Does it really model anything as claimed?
- Compare results with adding second application (video)
- 64% more energy with hardware-only
  - Reduces chances to power down hardware
- 53% more otherwise
- Only 18% more energy with low fidelity
  - Makes use of otherwise idle power usage

Overall Results

- Very sensitive to data and applications
- Sometimes combining low fidelity with hardware management buys more than expected
  - Provides more opportunities for further hardware savings
  - Could save up to 30% by backlighting only needed portion of display

Goal-directed Energy Saving

- Battery needs to last a certain amount of time
- Use Odyssey to manage energy consumption to last long enough
- Base on observed past and present usage
- If predicted usage exceeds remaining supply, direct apps to lower fidelity
- More practical than asking apps to guess

Goal-directed Savings, continued

- Aim for best possible user experience
  - Highest fidelity possible, given predictions
  - Avoid frequent adaptations
- Prototype uses on-line Powerscope
  - Could be built into BIOS or use PCMCIA multimeter
- Smoothing function between past and present: a is weight of past versus present

  new = (1-a)(this sample) +(a)(old)

Adaptation Method & Parameters

- Value of a changes as energy drains
- Upcalls to tell app to decrease fidelity as predicted demand exceeds energy
- Upcalls to app to increase fidelity when remaining energy exceeds predicted demand
- Cap fidelity changes at 1 per 15 seconds
- Degrades lower priority apps first

Results

- Overhead (measurement & prediction computation):
  - probably 0.25% of background consumption
- Sensitivity to half-life (a):
  - 1% too low (unstable)
    - Largest residue and too many adaptations
  - Large means more stable
  - 15% too high (not agile enough)
    - Failed to meet goal
- Longer experiments: even bursty workload is okay due to hysteresis

Reducing clock speed for energy savings

- Battery lifetime important for mobile devices
- Display, disk and CPU major energy consumers
- Now-popular idea for reducing CPU energy consumption
- MIPJ metric: work you can get done given some amount of energy consumed
- Energy savings possible from reducing clock speed

MIPJ and clock speed

- MIPJ itself unaffected by reduced clock speed
  - Linear reduction in energy consumption
  - But also a linear reduction in work done
- Work done and energy consumed cancel each other out
  - In idle loop can reduce clock to zero
  - But no work being done then

Voltage reduction

- Reducing clock rate creates better opportunity: quadratic energy savings
  - E/clock proportional to V^2
- With lower voltage, must reduce clock speed
  - Settling time for a gate proportional to voltage
  - Any voltage reduction from running slower will help

Advantage of running slowly

- Run fast for ½ the time?
  - Spend X amount of energy
- Run half as fast for the whole time?
  - Spend ¼ the energy per cycle
- Spend only ¼ the energy, since same # of cycles
- Idle time is wasted energy, even if clock is stopped!

Unusual scheduling philosophy

- Usually we ask how much work we can get done before various deadlines
- Here we ask how slow we can go!

Simulations

- Evaluation through simulation of trace data
- Assume can stretch runtime into “soft” idle time
  - Soft idle time is waiting for user input or response to request
  - Hard idle time is something like a disk wait
- Also assume
  - No reordering
  - Can switch speeds instantly

Traces

- Several hours of UNIX scheduler info
- A few short specific traces of interactive work
- Overhead of tracing determined from traces (1.5-7%)
- Trace points:
  - Context switch away from a process
  - Enter idle loop
  - Exit idle loop to run a process
  - Fork
  - Exec
  - Exit
  - Sleep (wait on event)
  - Wakeup of sleeping process

Scheduling algorithms

- OPT
  - Unbounded delay, perfect future knowledge
  - Stretches runs to fill all idle times, except when machine is “off”
  - Requires knowledge of future and assumes reordering okay
  - Undesirable: large delays of jobs will affect real-time events
  - Limited by minimum speed – achieved maximum savings
- FUTURE
  - Bounded delay, limited future knowledge
  - Limited window into future
  - Jobs only delayed up to end of window
  - Size of window affects results: 1ms no savings, 400ms = OPT

More realistic algorithm

- PAST
  - Bounded delay, limited knowledge of the past
  - Practical version of FUTURE
  - Fixed window into past - assume future is like past
  - Look at percent of interval used, if idle, slow down
  - If excess work or too much “hard” idle time, speed up

Results

- As interval lengthens, FUTURE and PAST approach OPT
- As interval lengthens, PAST has more excess cycles (jobs that “missed” deadline)
- PAST actually better than FUTURE
  - Delaying excess cycles to next window effectively lengthens window
- 1.0V not always better than 2.2V
  - Too slow: excess cycles cause speed ups
- Overall savings good: from 5 to 75%, usually 25-65%

Conclusions

- Better to maintain average speed than slow down and speed up
- Trade-off between excess cycles (user experiences delay) and energy saved
- A short enough window also allows disk to spin down or display to go blank
- Overall results look good