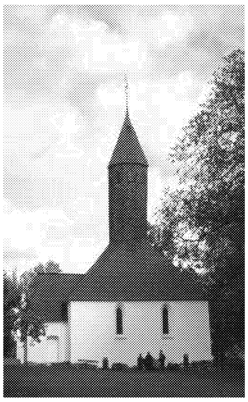
MÄLARDALENS HÖGSKOLA

ESSES 2003

European Summer School on Embedded Systems

Lecture Notes
Part IX

Low Power Systems: Operating System Support for Low-power



European Summer School on
Embedded Systems
Jul 14 - Oct 10 2003
Västerås Sweden

Editors: Ylva Boivie, Hans Hansson, Jane Kim, Sang Lyul Min

Västerås, August 4-8, 2003
ISSN 1404-3041
ISRN MDH-MRTC-106/2003-1-SE

MRTC
MÄLARDALEN REAL-TIME
RESEARCH CENTRE

www.mrtc.mdh.se

Energy-Aware Adaptation for Mobile Computing

Jason Flinn

School of Computer Science
Carnegie Mellon University
jflinn@cs.cmu.edu

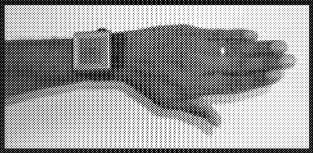
Energy-Aware Adaptation for Mobile Computing

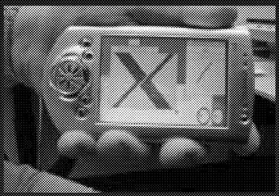
Jason Flinn

(Brief) Motivation

Energy is a vital resource for mobile computing

Can either increase supply or reduce demand





Increasing the supply of energy is difficult:

- Battery constrains size, weight of mobile device
- Battery technology improving slowly
- Processing requirements also increasing

Higher-Level Energy Management

Lots of effort aimed at reducing energy demand:

- Low-power circuit design
- Dynamic voltage scaling (Crusoe, SpeedStep)
- Low-power network design (Bluetooth)

Lower-level efforts by themselves are not enough!

- OS and applications must also be involved
- Poor implementation can waste low-level efforts
- OS and apps have more info about user intent
- Can make better performance/energy tradeoffs

Some Important Questions

- How to best measure OS and app energy consumption?
- What are some possible ways of reducing application energy usage?
- How can the operating system make tradeoffs between performance, quality, and energy conservation?
- What is the best power management interface to present to users?

Roadmap

- Energy Measurement
- Energy-Aware Adaptation
- Operating System Support for Energy Management
- Remote Execution
- Self-Tuning Power Management
- Wrap-Up

Why Measure Energy?

Software developers need to know:

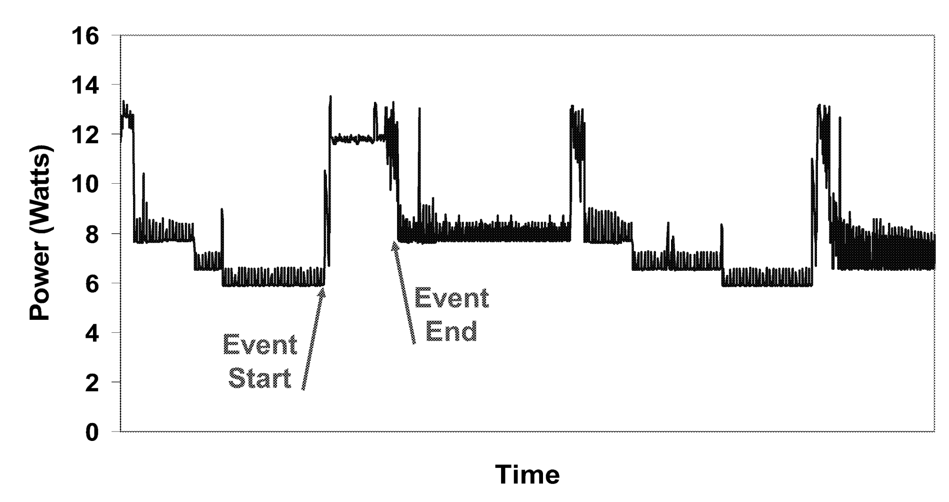
- Which code components are most wasteful of energy?
- What is the energy benefit of an optimization?
- What are the energy tradeoffs of design alternatives?
- Are there any energy-related bugs?

Consider similar problem: optimizing for performance

- Many tools exist: prof, gprof, DCPI
- Even gettimeofday() and printf() very useful
- No easy parallels for energy optimization (yet!)

Need to build the telescope before we can go to the stars.

Measuring Application Energy Usage



Can use DMM or other instrument to measure power

Need to correlate power drawn with application events

Energy Measurement: Design Considerations

Scope:

- Are we measuring a small number of instructions, …
- Or are we measuring a large, multithreaded program?

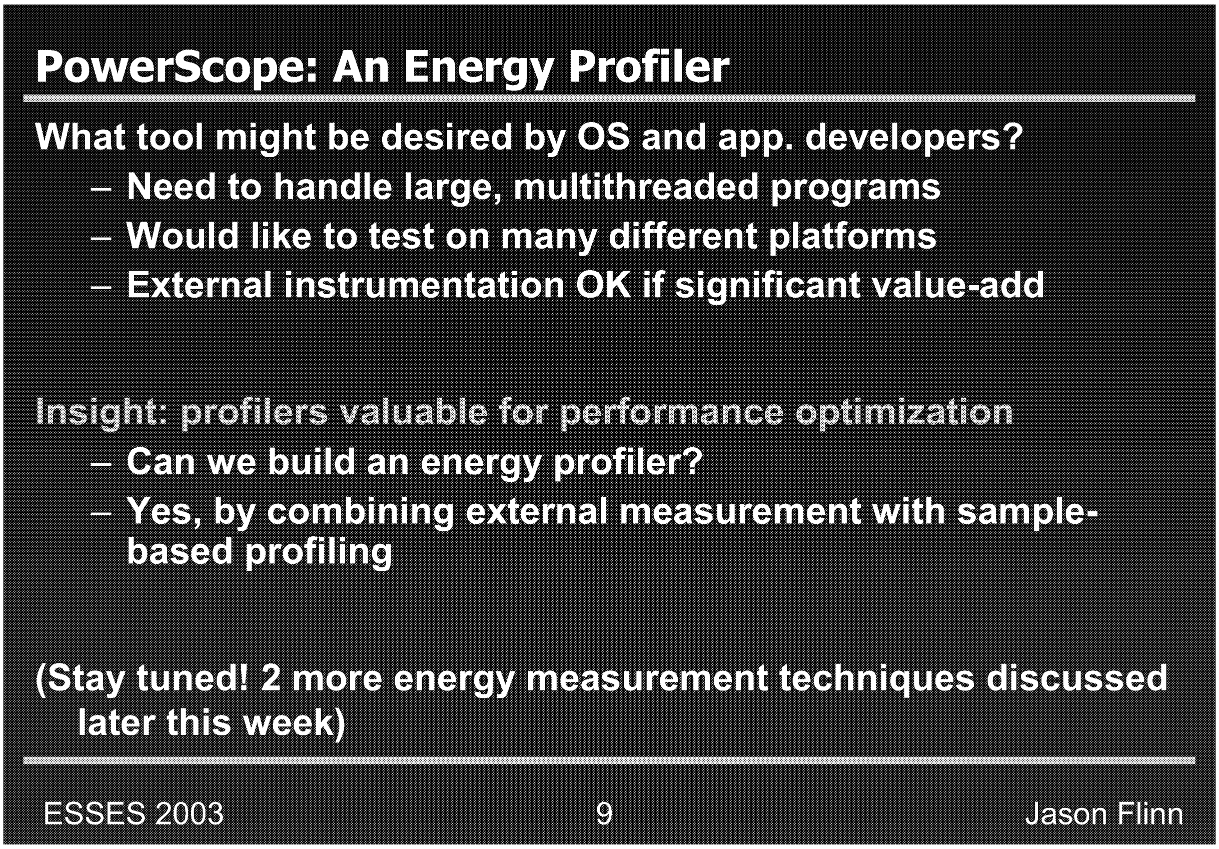
Instrumentation:

- Can we use external infrastructure for measurement,
- Or do we just use what is on the mobile computer?

Heterogeneity:

- Are we measuring a single computer system,
- Or are we measuring many different types?

Very diverse criteria: maybe no single solution is possible!



PowerScope: An Energy Profiler

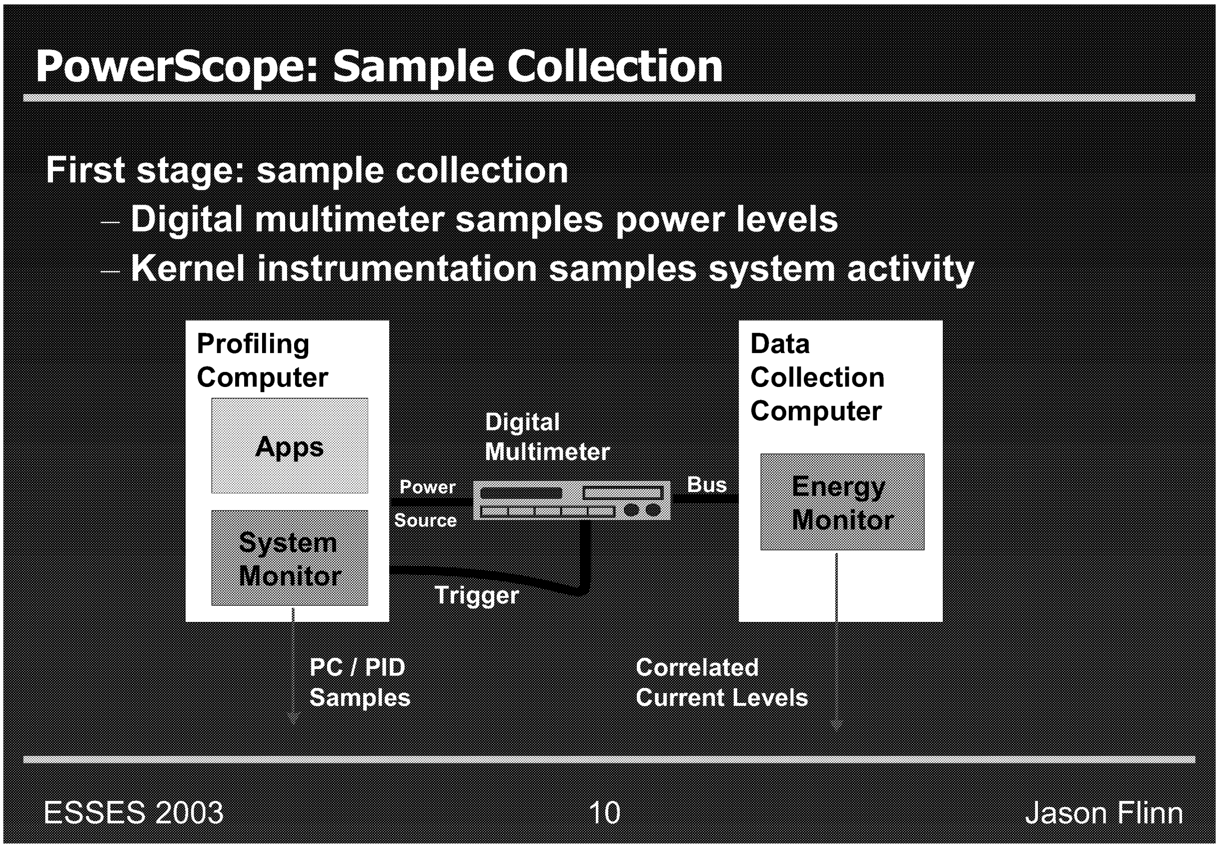
What tool might be desired by OS and app. developers?

- Need to handle large, multithreaded programs
- Would like to test on many different platforms
- External instrumentation OK if significant value-add

Insight: profilers valuable for performance optimization

- Can we build an energy profiler?
- Yes, by combining external measurement with sample-based profiling

(Stay tuned! 2 more energy measurement techniques discussed later this week)



PowerScope: Sample Collection

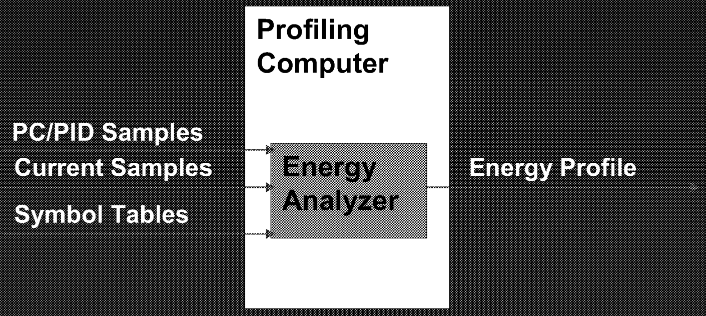
First stage: sample collection

- Digital multimeter samples power levels
- Kernel instrumentation samples system activity

PowerScope: Profile Generation

Stage 2: Generate profile off-line

- Minimize overhead during profile collection
- Attributes samples to specific processes, procedures



Sample Profile: Summary

Process	Elapsed Time (s)	Total Energy (J)	Average Power (W)
/usr/odyssey/bin/xanim	66.57	643.17	9.66
/usr/X11R6/bin/X	35.72	331.58	9.28
/netbsd (kernel)	50.89	328.71	6.46
Interrupts-WaveLAN	18.62	165.88	8.91
/usr/bin/odyssey	12.19	123.40	10.12
Total	183.99	1592.75	8.66

Sample Profile: Process Detail

Procedure	Elapsed Time (s)	Total Energy (J)	Average Power (W)
_Dispatcher	0.25	2.53	10.11
_IOMGR_CheckDiscript	0.17	1.74	10.23
_sftp_DataArrived	0.16	1.68	10.48
_rpc2_RecvPacket	0.16	1.67	10.41
_ExaminePacket	0.16	1.66	10.35
. . .			
Total	12.19	123.40	10.12

Evaluating an Energy Measurement Tool

What metrics should be used for evaluation?

Overhead: performance and power impact of tool

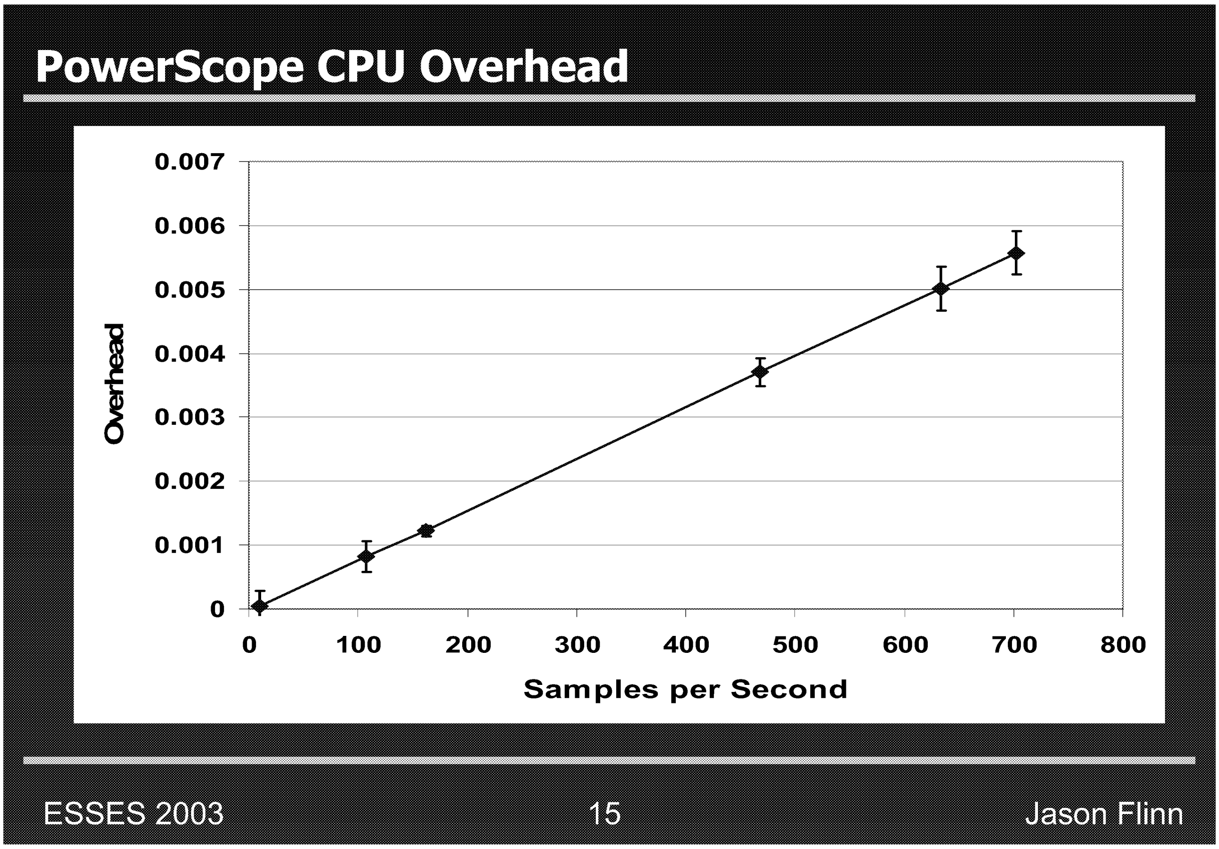
- Overhead perturbs results, so try to minimize it

Resolution: minimum event size that can be measured

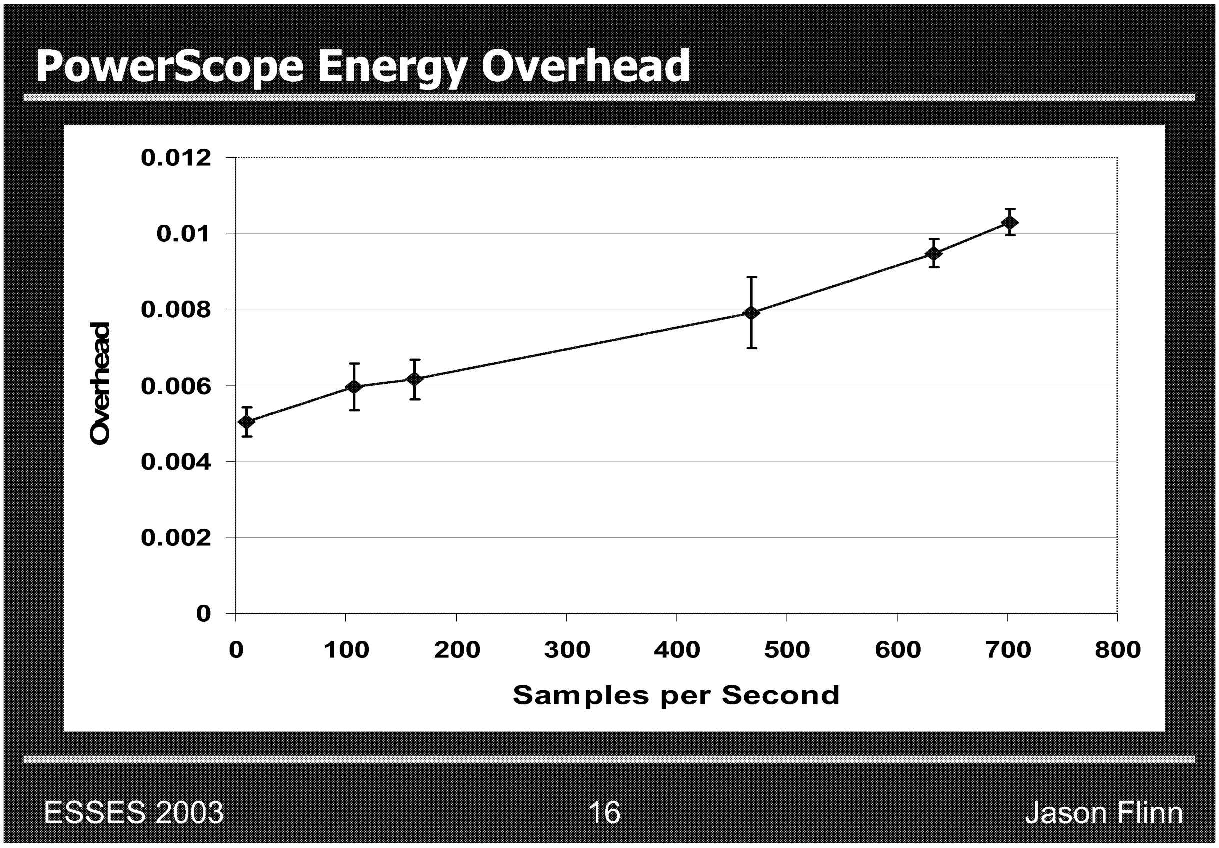
- Smaller event size enables fine-grained optimization
- Greater resolution may imply greater overhead!

Let’s look at these metrics for PowerScope:

- Platform 75MHz 486 laptop.
- Vary PowerScope sample frequency

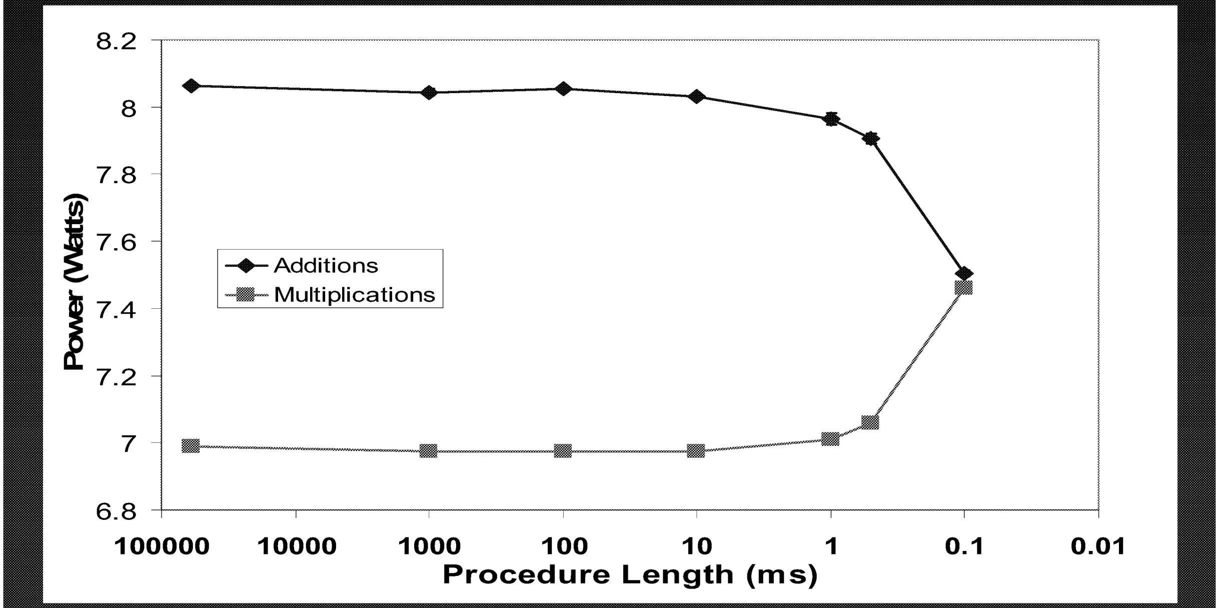


PowerScope CPU Overhead

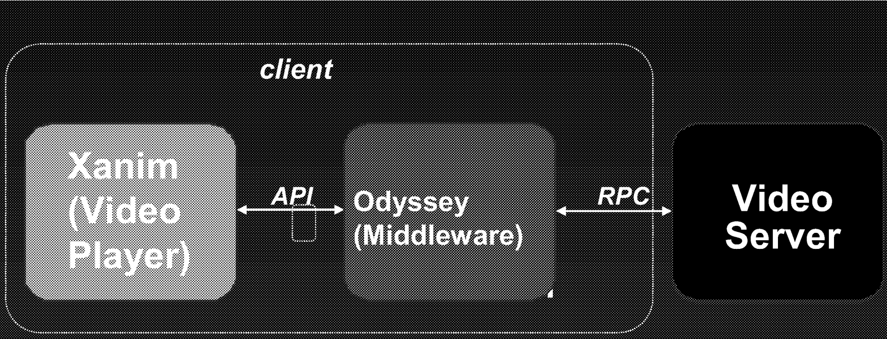


PowerScope Energy Overhead

PowerScope Accuracy (633 Hz.)



Case Study: Adaptive Video

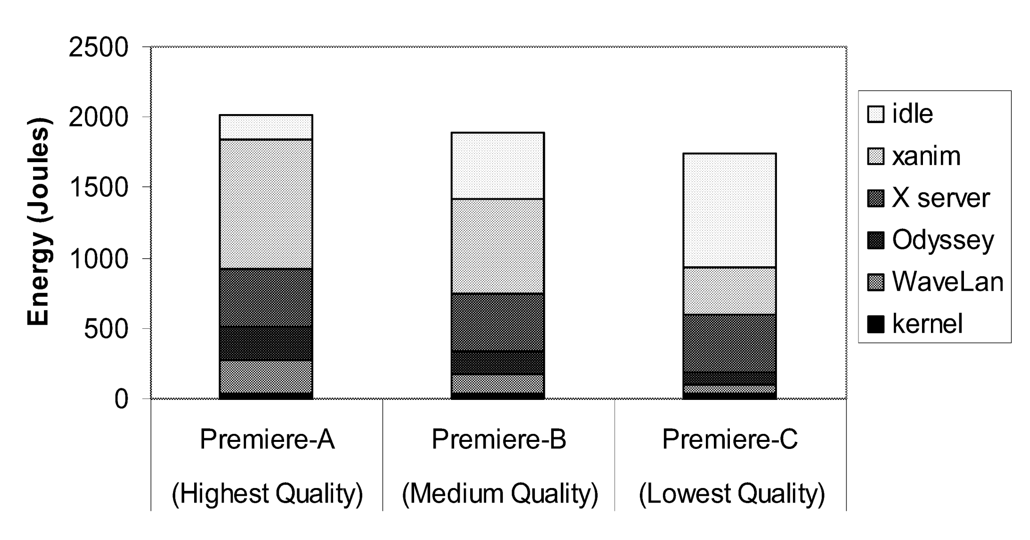


Goal: reduce power usage to meet design constraints

Experimental Setup:

Client: 75MHz 486 laptop
Server: 200 MHz Pentium Pro
Network: 900 MHz 802.11

Case Study: Benefit of Lossy Compression



Benefit of lossy compression: 13% energy reduction

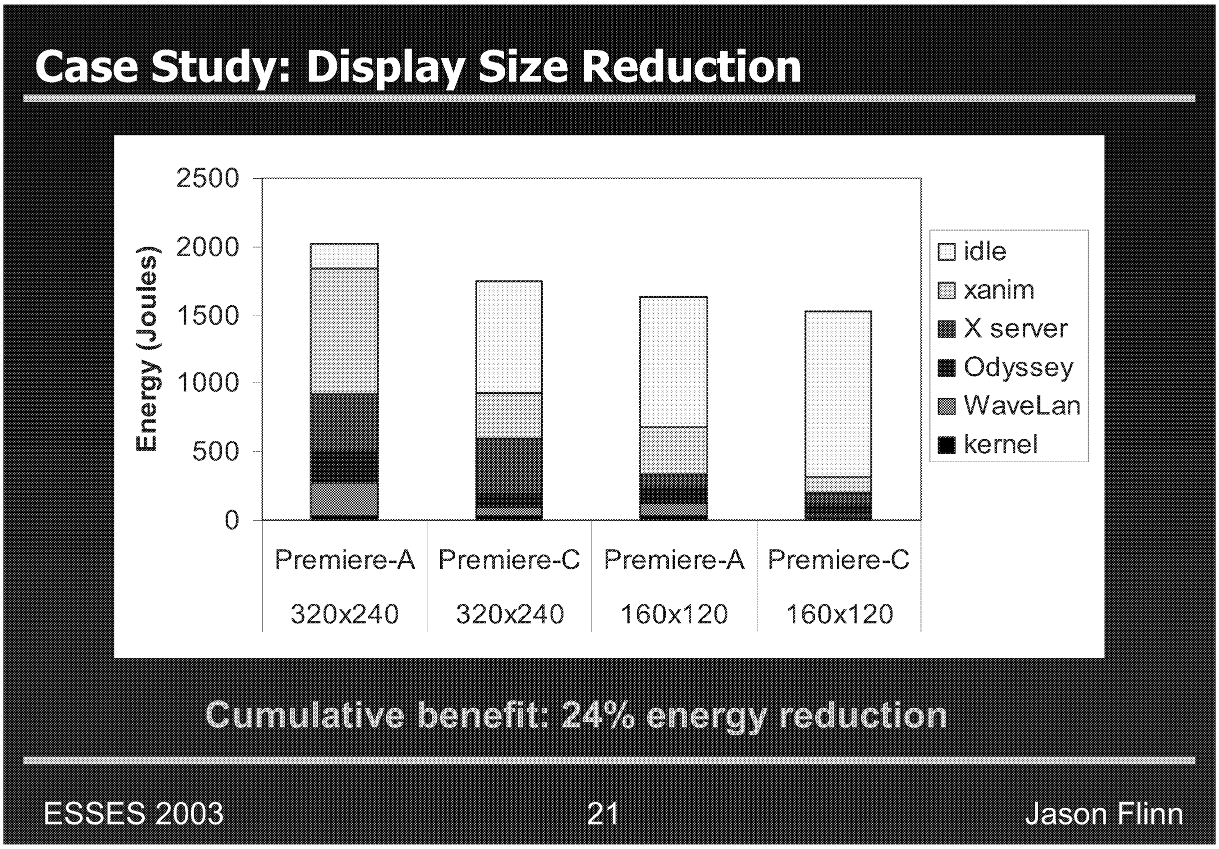
Experiment: Display Size Reduction

Compression does not reduce X server energy usage.

Will reducing the display size help?

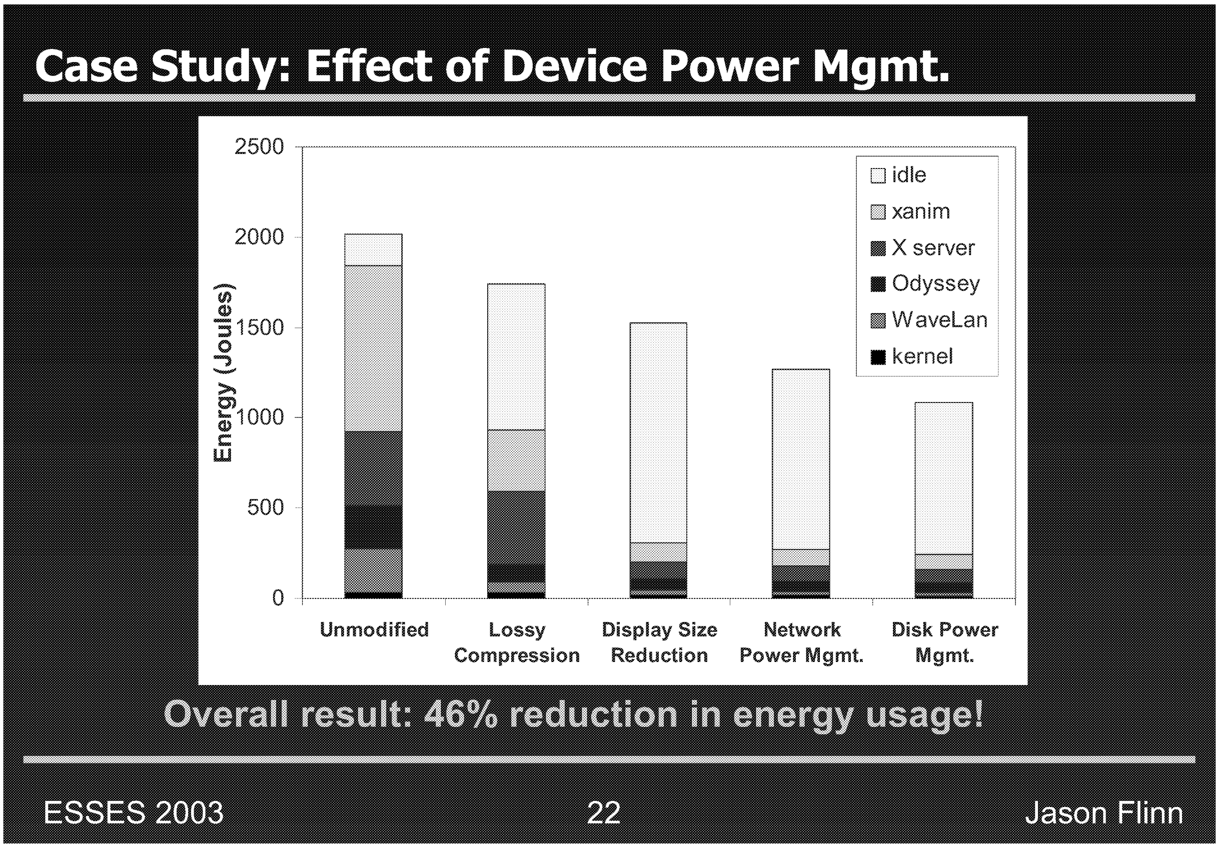
Repeat experiment with two additional tracks:

- One each at highest and lowest compression.
- Halve the height and width of the video display.



Case Study: Display Size Reduction

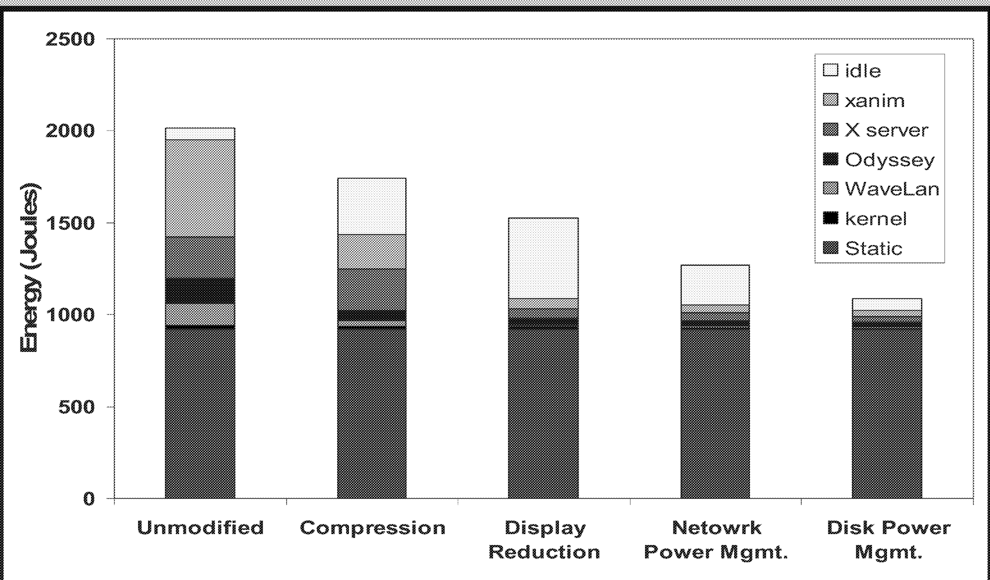
Cumulative benefit: 24% energy reduction



Case Study: Effect of Device Power Mgmt.

Overall result: 46% reduction in energy usage!

Case Study: Static vs. Dynamic Power



Now most power usage is static: no more knobs to turn!

PowerScope: Discussion

Advantages:

- Works well for complex applications
- Relatively easy to use for different mobile platforms
- Does not require per-device models

Disadvantages:

- Requires external measurement equipment
- Cannot differentiate asynchronous I/O
- Measurement granularity limited to large procedures

Roadmap

- Energy Measurement
- Energy-Aware Adaptation
- Operating System Support for Energy Management
- Remote Execution
- Self-Tuning Power Management
- Wrap-Up

Energy-Aware Adaptation

How to write applications that use less energy?

- Make them more energy-efficient (hard)
- Reduce application-specific quality (i.e., case study)
- Remote execution
- Better integration with device power management
- Other possibilities?

Energy-Aware Adaptation: Dynamic balancing of quality and energy conservation

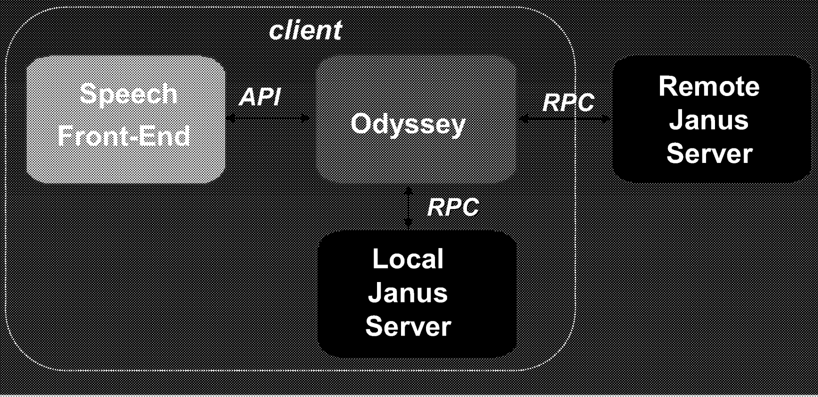
Reduce fidelity: an application-specific metric of quality

Speech Recognition

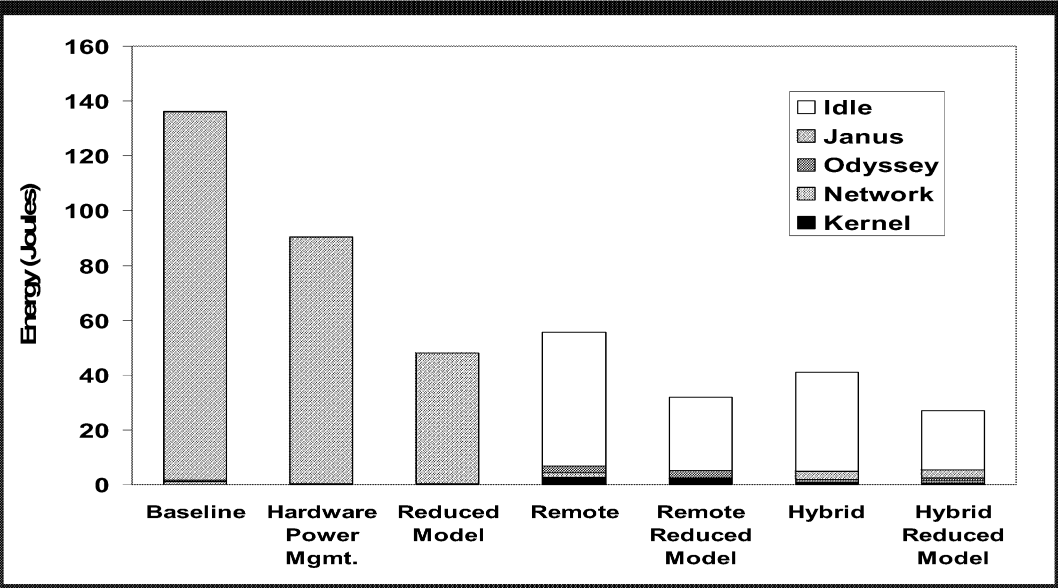
Janus: Speech-to-text translation of spoken utterances

Fidelity: full or reduced speech model

Execution location: local, remote, or hybrid

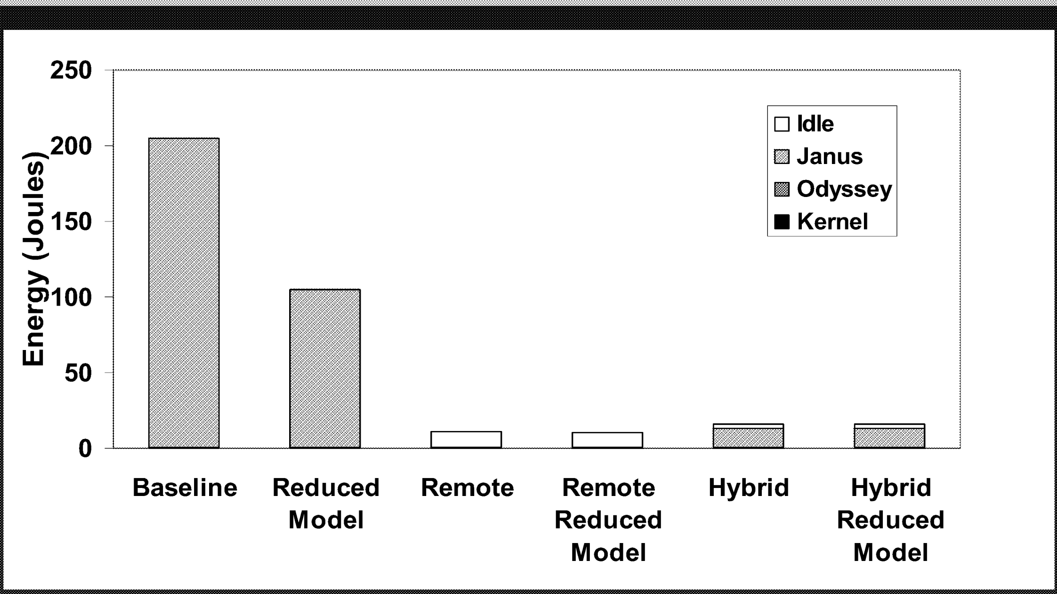


Impact of Fidelity on Speech Recognition



System energy usage reduced 76%!

Speech Results on Itsy Handheld



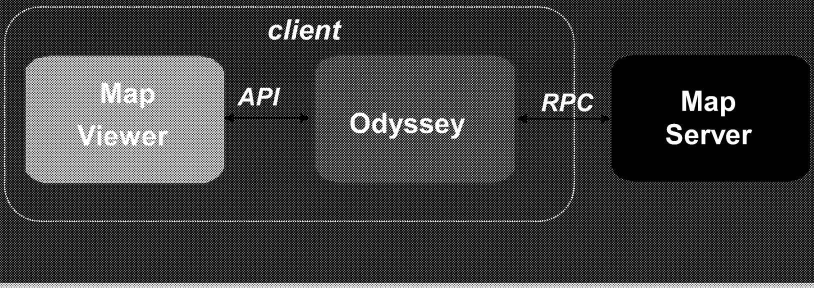
Energy-aware adaptation more effective on Itsy

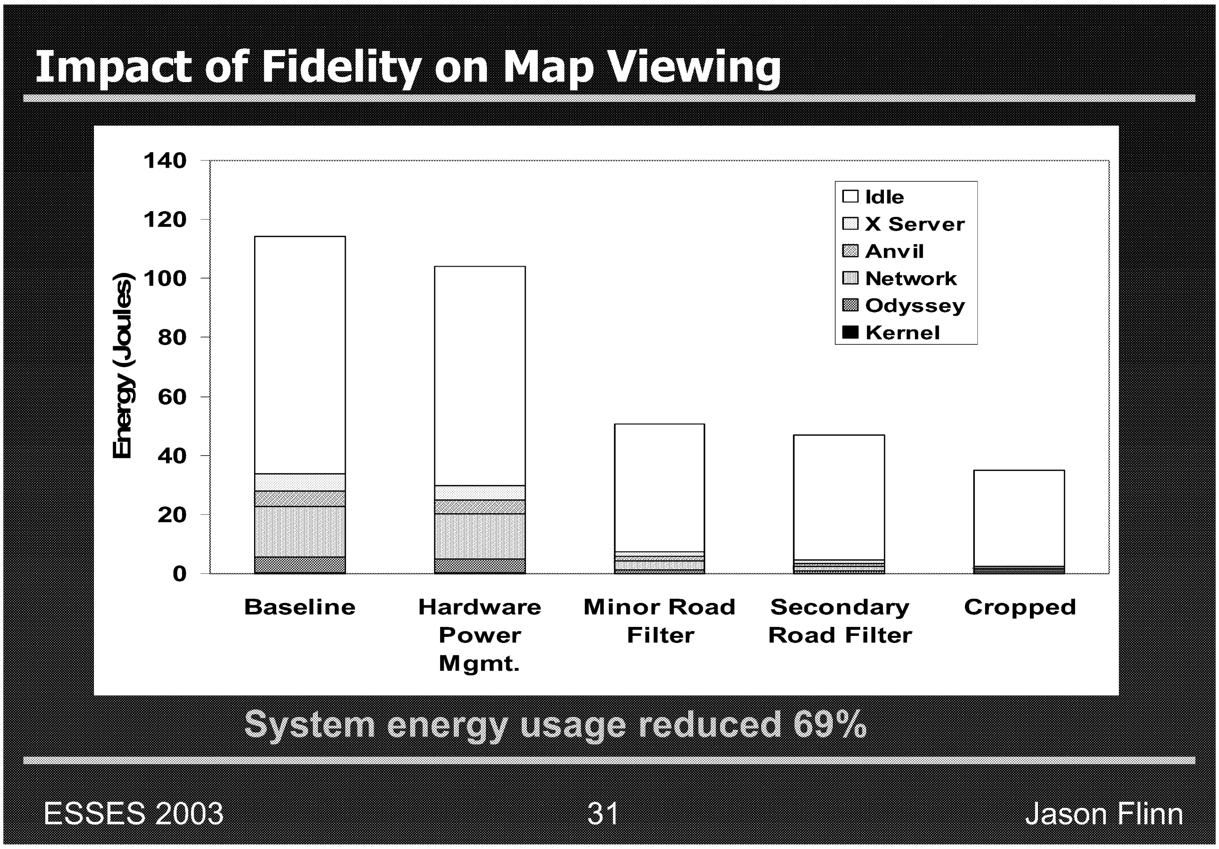
Map Viewer

Fetch maps from remote server, display on client

Fidelity: remove geographic features, minor roads

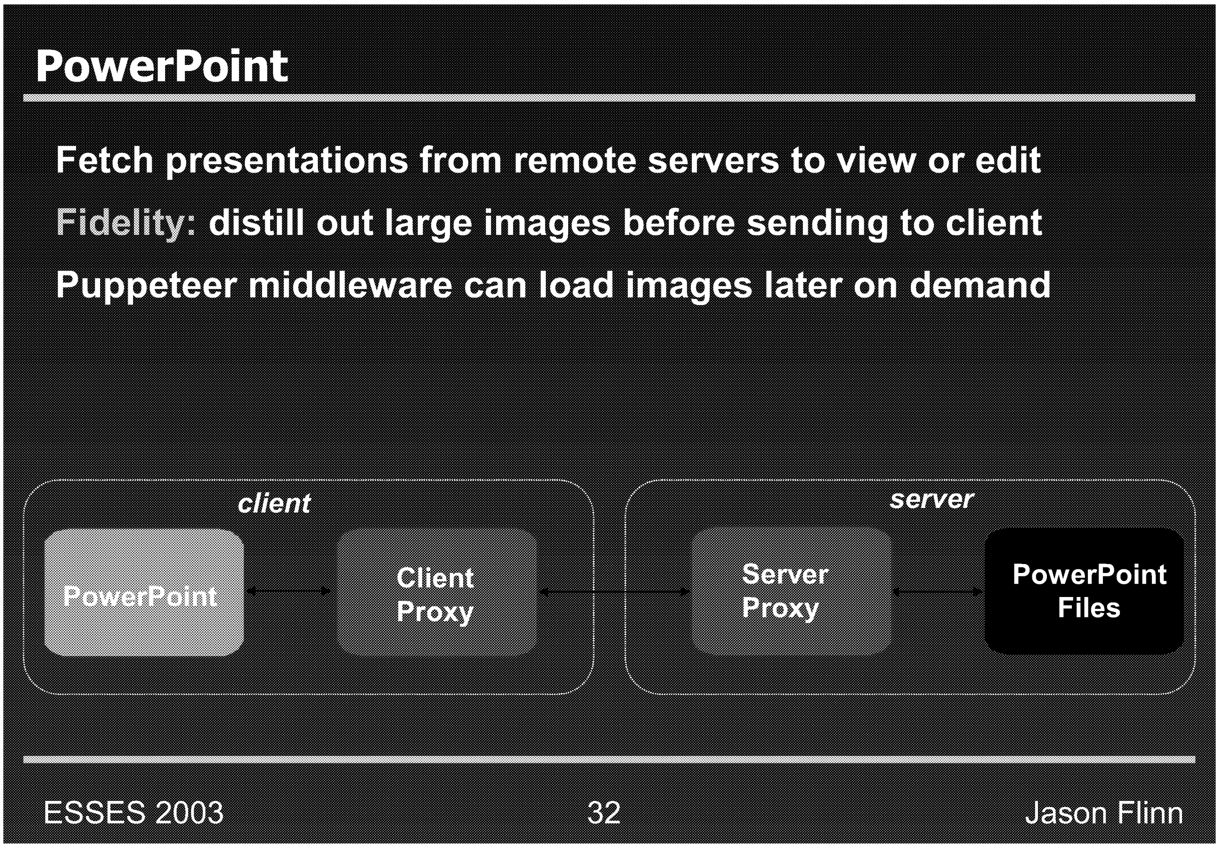
Fidelity: crop map by zooming in on specific area





Impact of Fidelity on Map Viewing

System energy usage reduced 69%



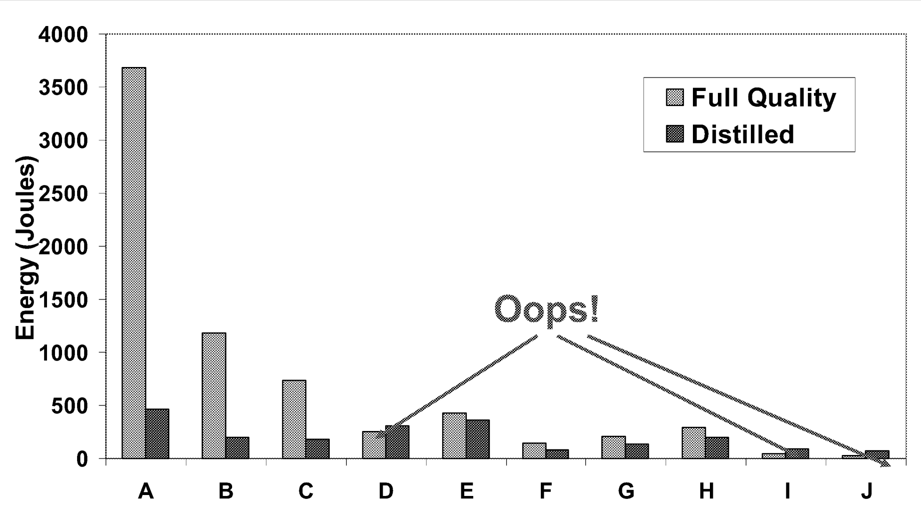
PowerPoint

Fetch presentations from remote servers to view or edit

Fidelity: distill out large images before sending to client

Puppeteer middleware can load images later on demand

Impact of Fidelity for Loading Documents

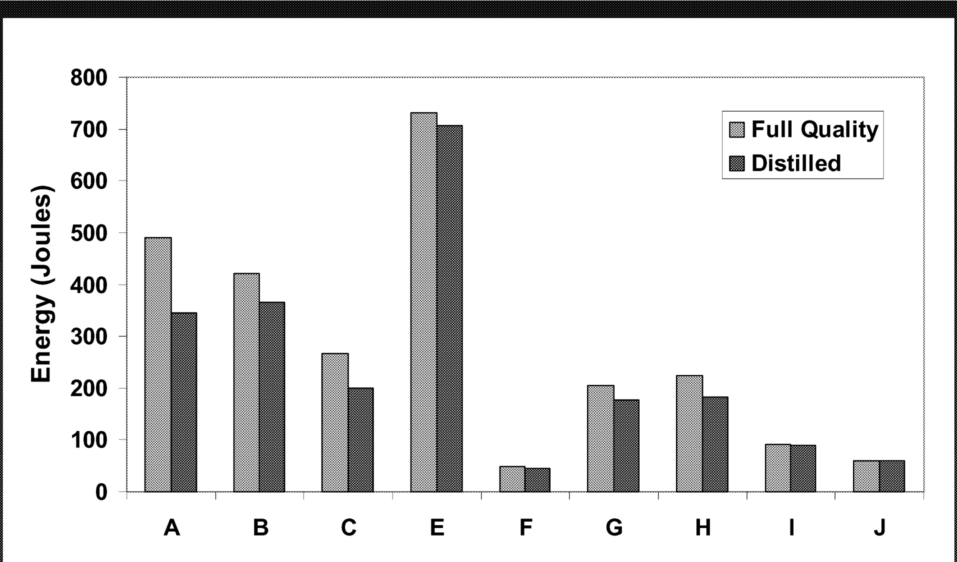


Oops!

Average energy usage reduced 40%!

With simple filter, energy usage reduced 49%!

Impact of Fidelity for Editing Documents



Reduces energy to page through document by 13%

Summary: Energy-Aware Adaptation

Application	Baseline	Power Mgmt.	Fidelity Reduction	Combined
Video	1.00	0.91	0.84	0.65
Speech	1.00	0.66	0.53	0.24
Map	1.00	0.85	0.51	0.40
Web	1.00	0.76	0.93	0.69
PowerPoint	1.00		0.51	0.51

Energy-aware adaptation:

- significantly reduces energy usage
- complements hardware power management
- is more effective on energy-efficient platforms
- often exhibits predictable effects
- can be implemented in closed-source environments

Roadmap

- Energy Measurement
- Energy-Aware Adaptation
- Operating System Support for Energy Management
- Remote Execution
- Self-Tuning Power Management
- Wrap-Up

Aside: Designing an Intuitive Interface

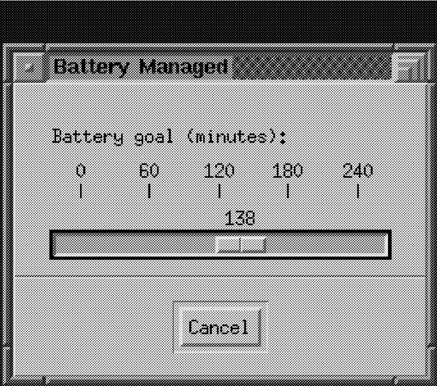
What are the properties of an intuitive interface?

- Minimally distracting: only a few parameters that do not constantly need to be reset
- Meaningful: parameters expressed in terms that directly impact the user.
- Well-calibrated: effect of changing each parameter should be obvious.

Are current OS power mgmt. interfaces intuitive?

How should we define an intuitive interface for energy-aware adaptation?

Goal-Directed Adaptation



Battery Managed

Battery goal (minutes): 138

User specifies needed battery duration

System ensures that it is met whenever possible

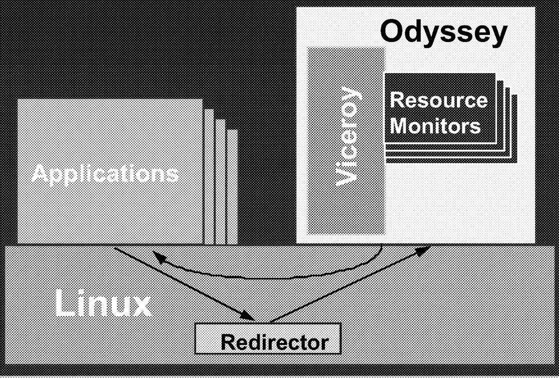
The system has the following goals:

- Meet the specified duration whenever possible
- Maximize application fidelity
- Minimize the number of adaptations

Odyssey: A Little More Detail

Applications describe operations, possible fidelities
Odyssey advises which fidelity they should use

Odyssey periodically:
– measures energy supply
– predicts energy demand
– triggers adaptation



Determining Energy Supply

Energy supply is the residual energy in the battery

How should we measure energy supply?
- PowerScope inappropriate; need on-line msmts.

Methods for measuring residual energy:
- Standardized interfaces (ACPI)
- Hardware device drivers (Smart Battery)
- Modulated energy supply

Predicting Future Energy Demand

Use smoothed observations of past power usage:

$$New = (1-\alpha) \bullet (sample) + \alpha \bullet Old$$

Multiply by time remaining to predict energy demand

α varies as energy drains:

- When goal is distant, large α yields stability
- When goal is near, small α yields agility

Calculate α so that half-life of decay function is 10% of time remaining

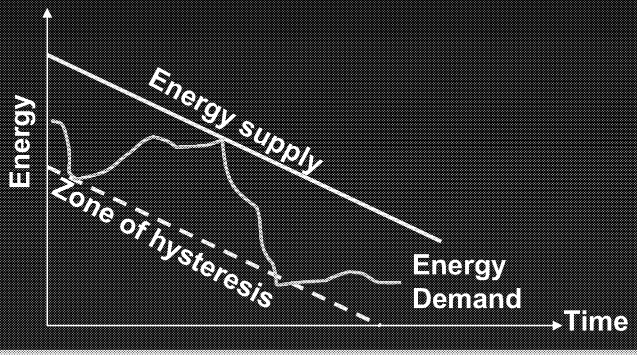
Triggering Adaptation

When demand exceeds supply:

- Applications adapt to conserve energy usage

When supply significantly exceeds demand:

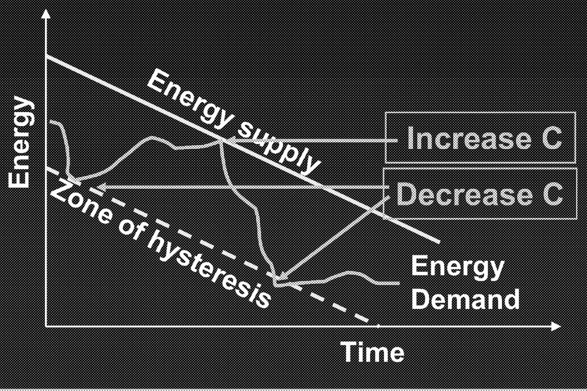
- Applications increase data fidelity



Changing Application Fidelity

C: represents importance of energy conservation

- Ranges from 0 to 1
- Adjusted when demand leaves zone of hysteresis
- Applications use C to choose fidelity levels



How Much Should Applications Adapt?

How should each application's energy use change?

Determined by adaptation policy

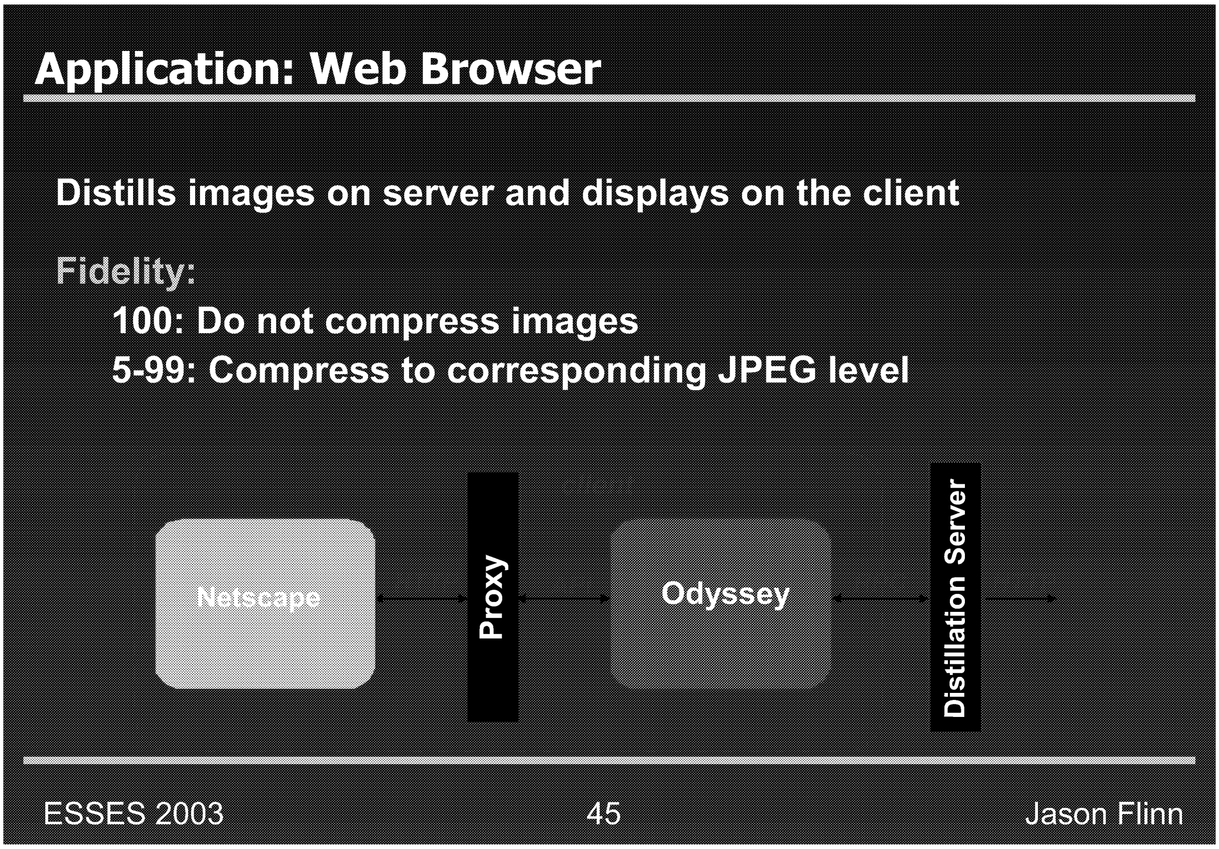
- Example: "Degrade each application equally"
- C = 0.2: "Decrease energy usage by 20%"

What fidelity produces needed energy usage?

Feedback approach is possible

- Adjust behavior by a small amount
- Pause to see if needed change is enough
- May not be sufficiently agile

Alternative: Use model of application energy usage

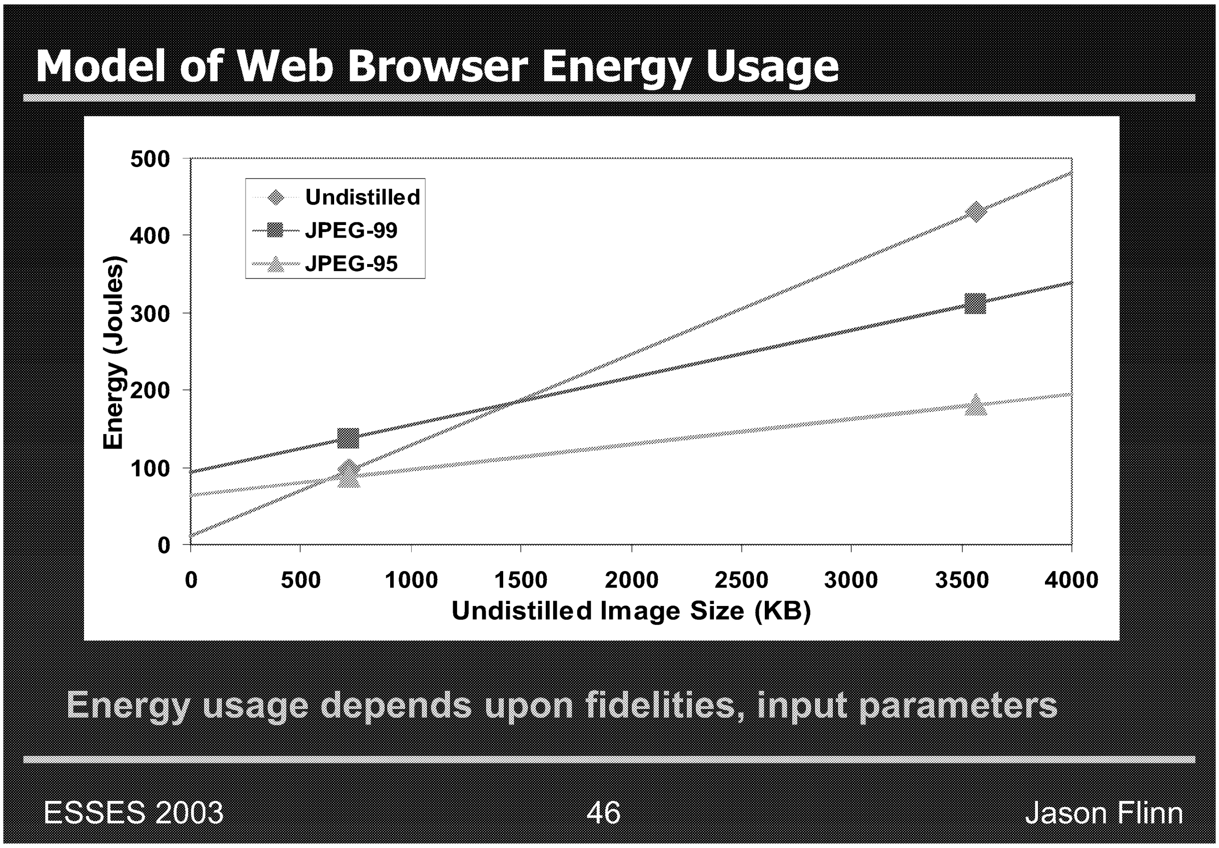


Application: Web Browser

Distills images on server and displays on the client

Fidelity:

- 100: Do not compress images
- 5-99: Compress to corresponding JPEG level



Model of Web Browser Energy Usage

Energy usage depends upon fidelities, input parameters

Modeling Application Energy Usage

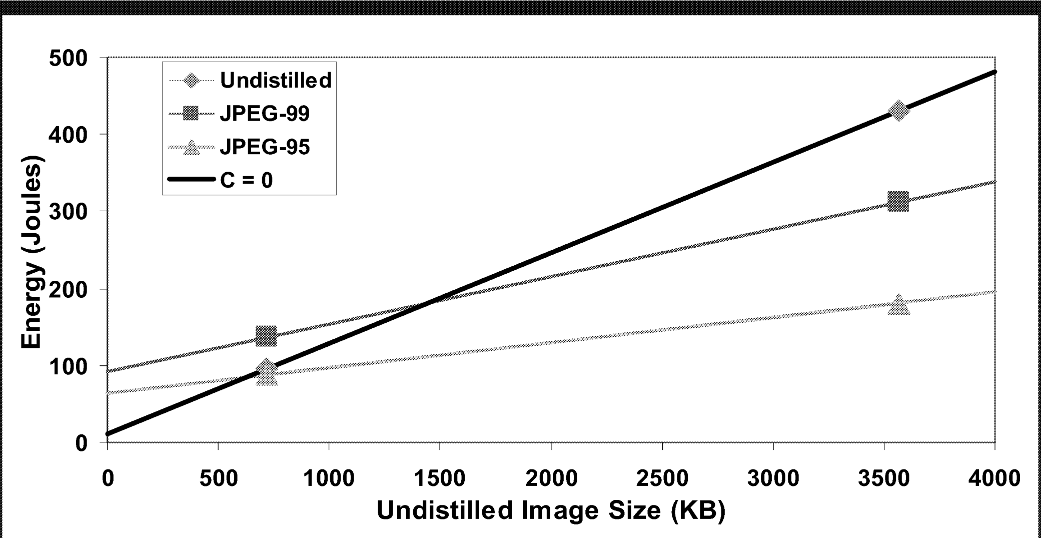
Can’t ask applications to specify their energy usage!

Odyssey builds a model by:
- Observing application behavior
- Logging energy usage
- Building models of energy usage

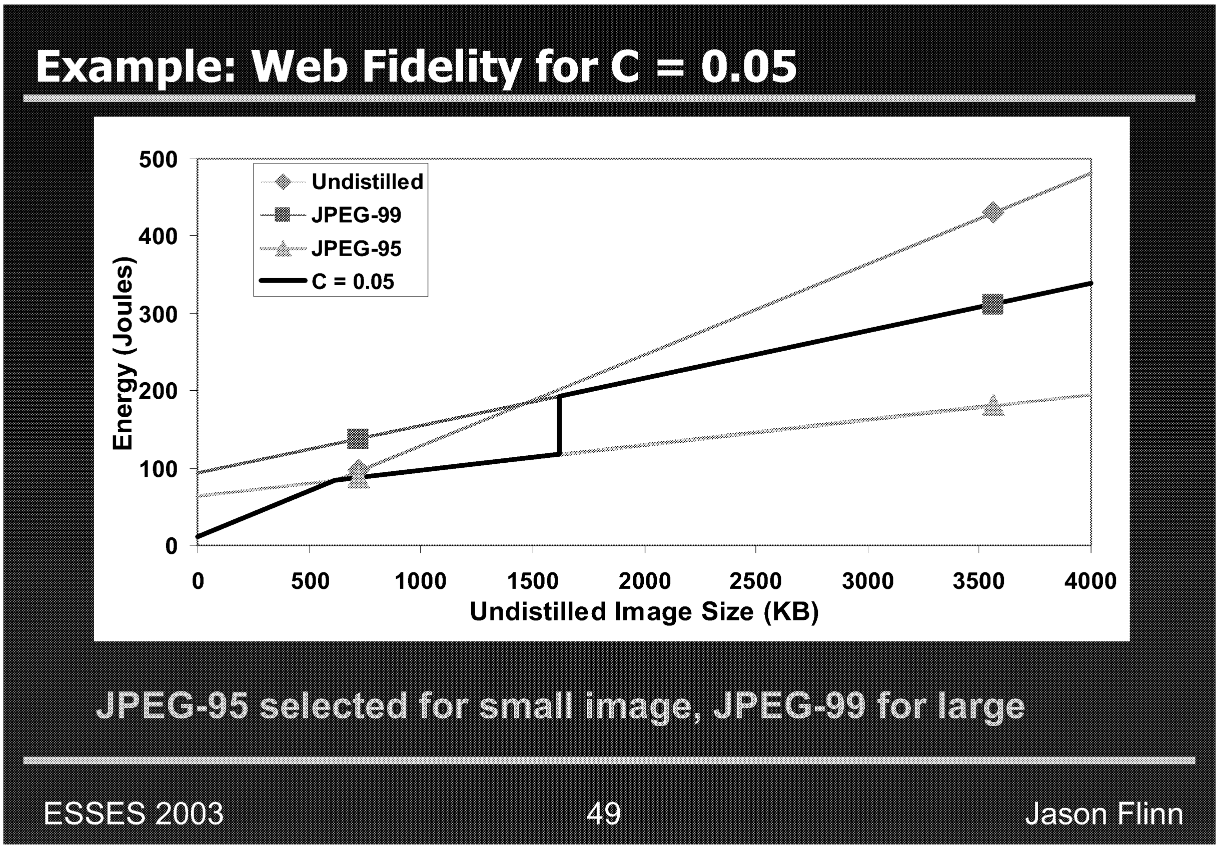
Applications describe types of operations to Odyssey
- Possible fidelities (i.e. JPEG compression)
- Input parameters (i.e. image size)

Applications signal start and end of operation execution

Example: Web Fidelity on Wall Power

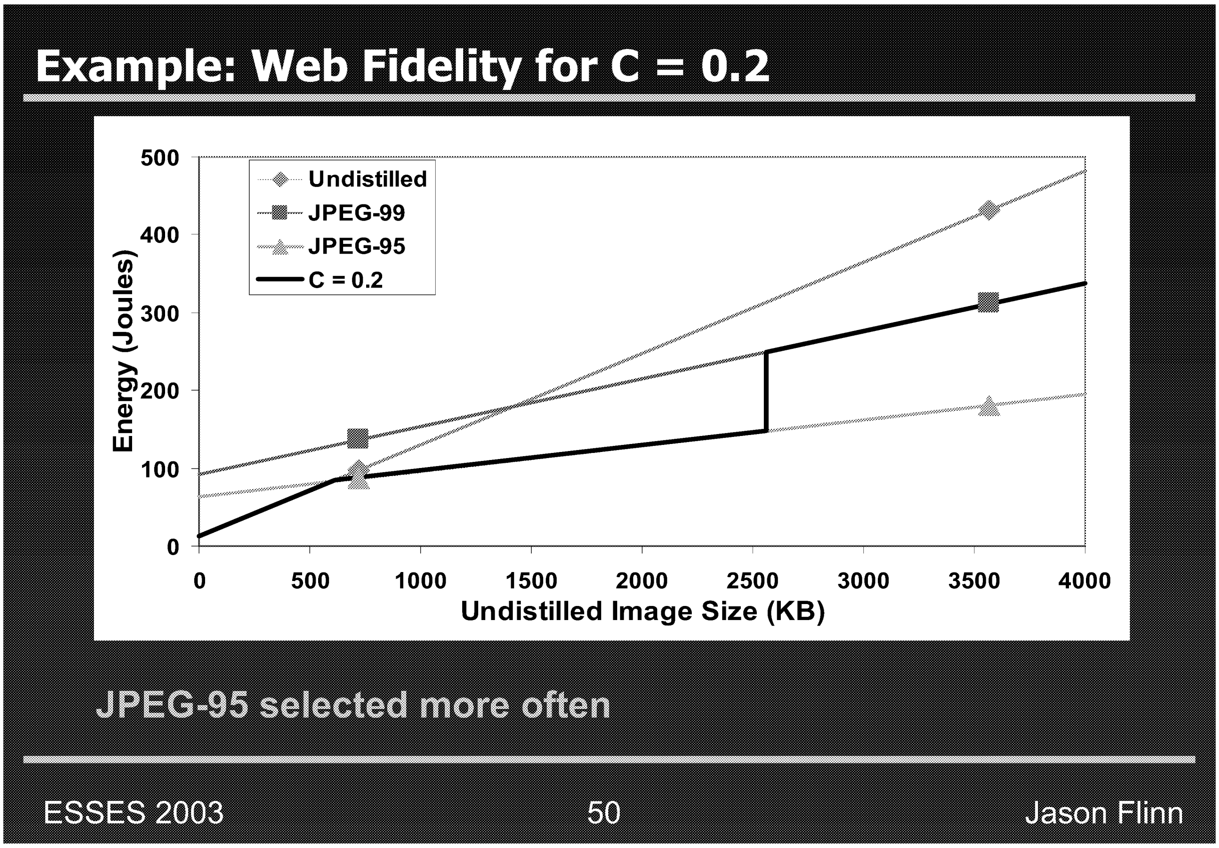


On wall power, C = 0 (Choose the highest fidelity always)



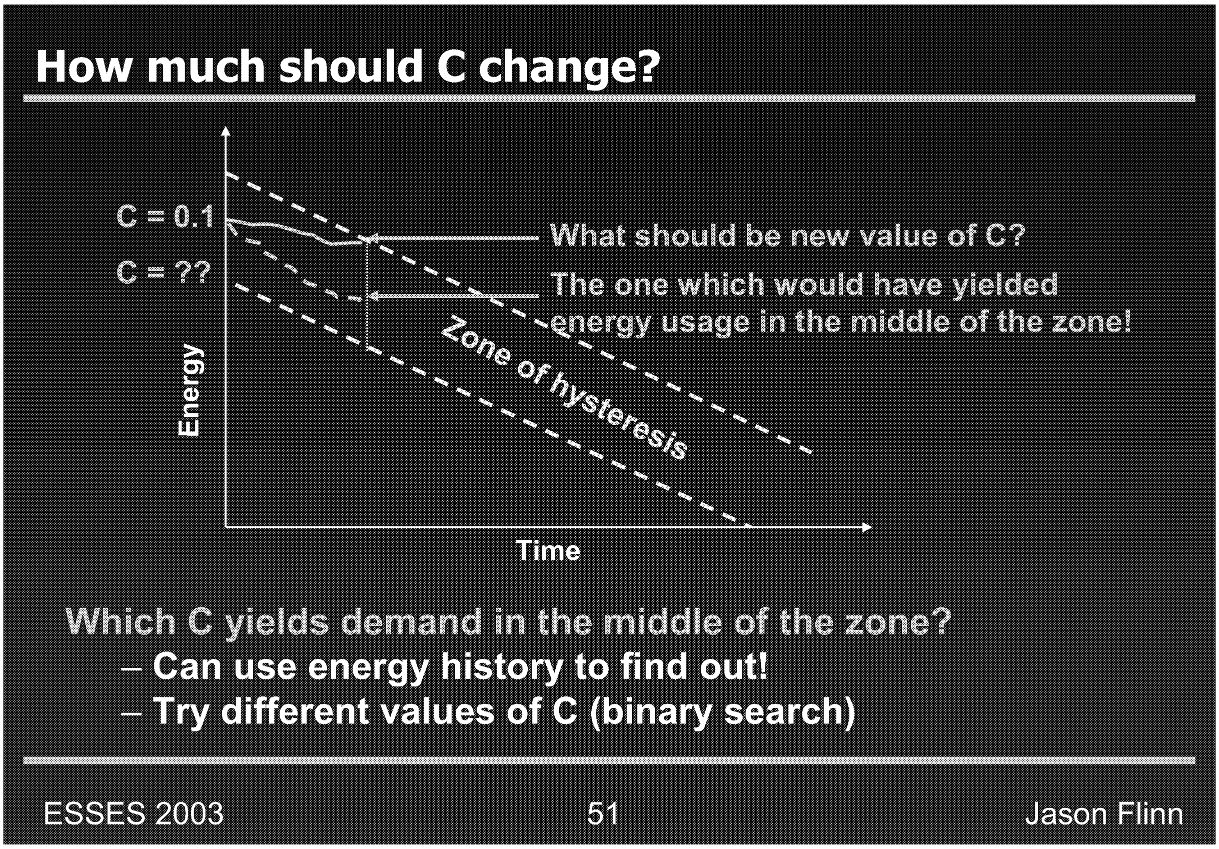
Example: Web Fidelity for C = 0.05

JPEG-95 selected for small image, JPEG-99 for large



Example: Web Fidelity for C = 0.2

JPEG-95 selected more often



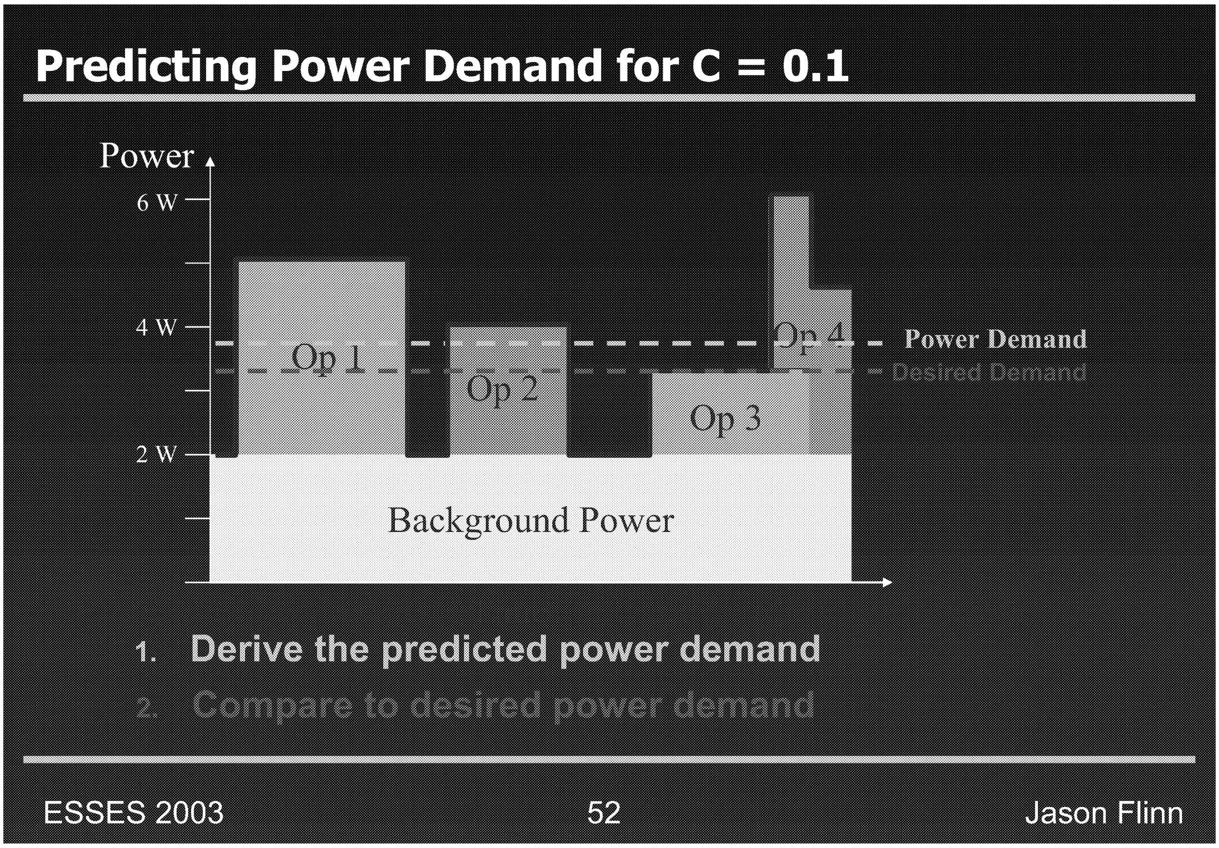
How much should C change?

What should be new value of C?

The one which would have yielded energy usage in the middle of the zone!

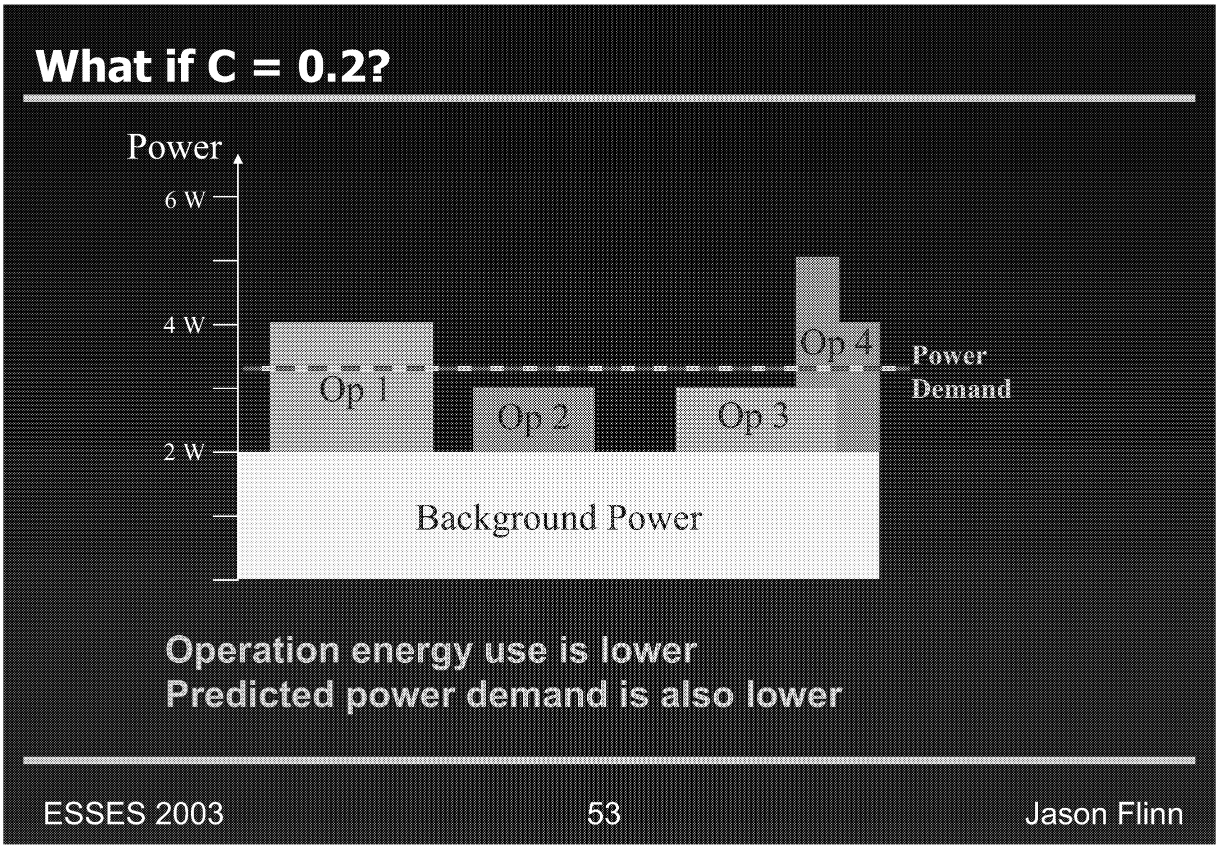
Which C yields demand in the middle of the zone?

- Can use energy history to find out!
- Try different values of C (binary search)



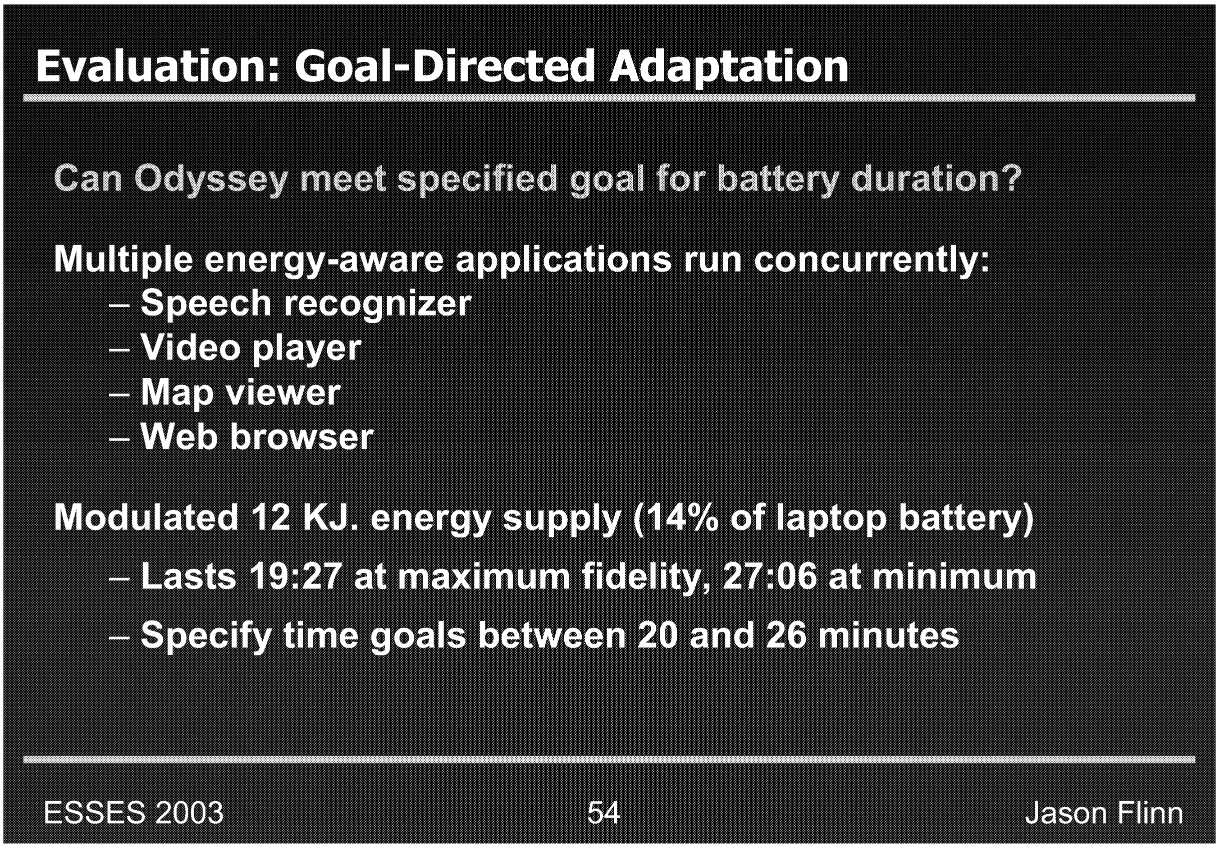
Predicting Power Demand for C = 0.1

1. Derive the predicted power demand
2. Compare to desired power demand



What if C = 0.2?

Operation energy use is lower
Predicted power demand is also lower



Evaluation: Goal-Directed Adaptation

Can Odyssey meet specified goal for battery duration?

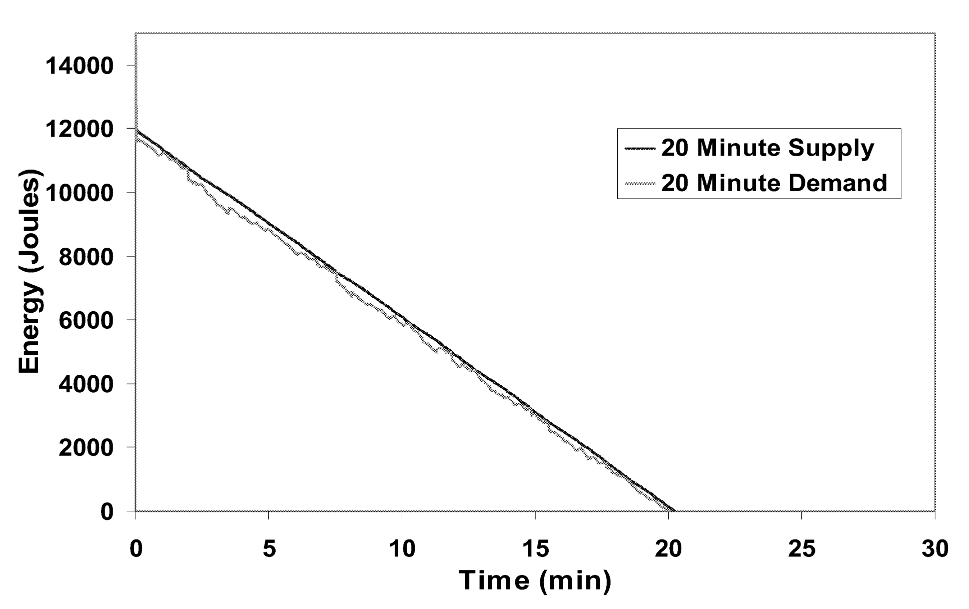
Multiple energy-aware applications run concurrently:

- Speech recognizer
- Video player
- Map viewer
- Web browser

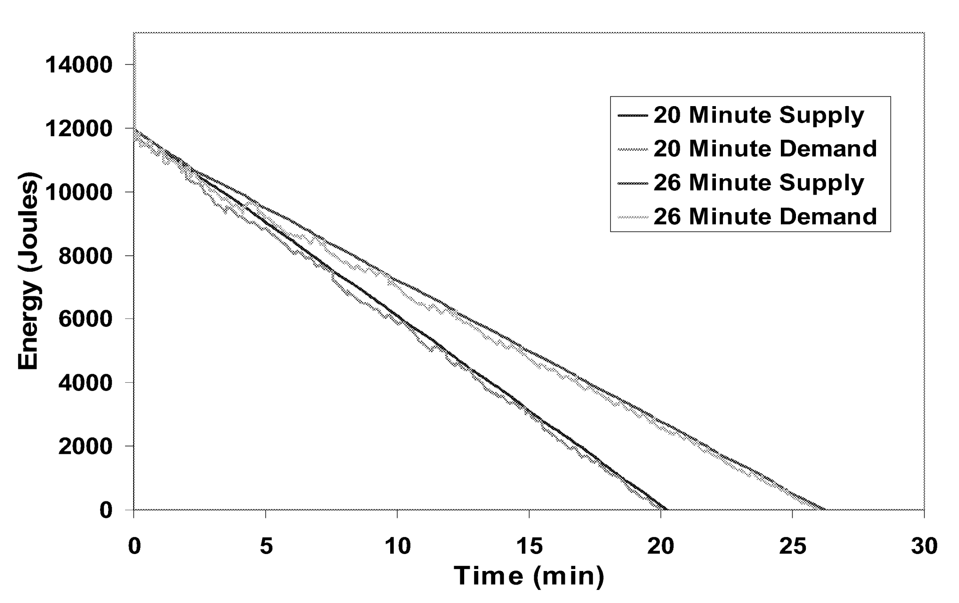
Modulated 12 KJ. energy supply (14% of laptop battery)

- Lasts 19:27 at maximum fidelity, 27:06 at minimum
- Specify time goals between 20 and 26 minutes

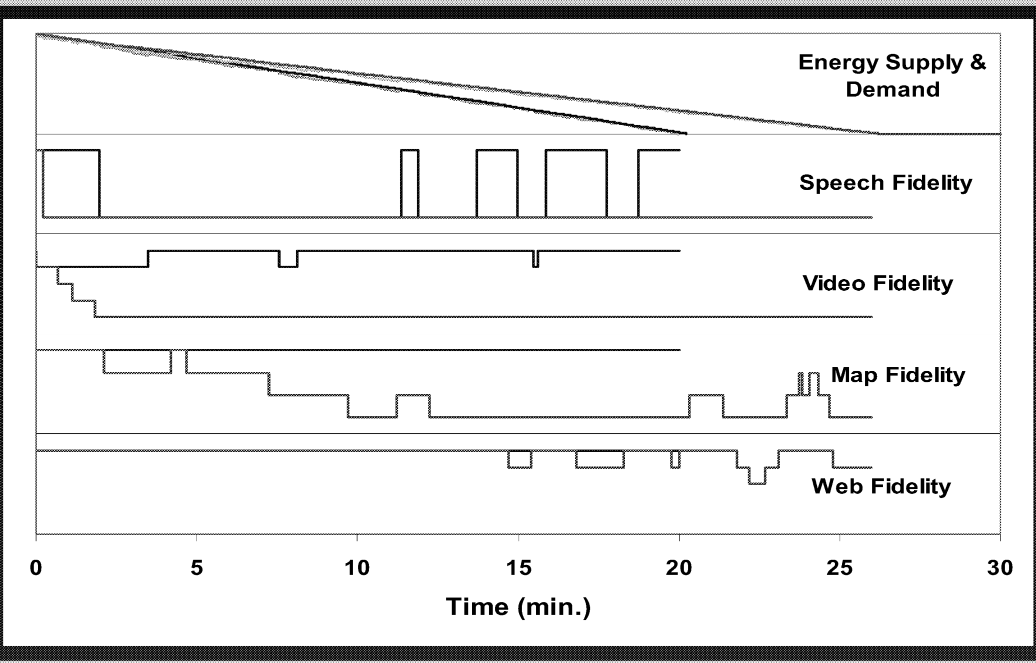
Energy Supply & Demand



Energy Supply & Demand (2 Examples)



Changing Fidelity



Goal-Directed Adaptation: It Works!

Specifed Duration (s.)	Goal Met	Residue Energy (%)	Residue Time (s)
1200	100%	1.21%	15.3
1320	100%	0.90%	12.9
1440	100%	0.84%	13.0
1560	100%	0.50%	8.7

Goal is met in every trial; residual energy is low

Other experiments show similar results for larger energy supply, modified time goal, and bursty workload

Evaluation: Use of Application History

Does the use of history improve adaptation decisions?

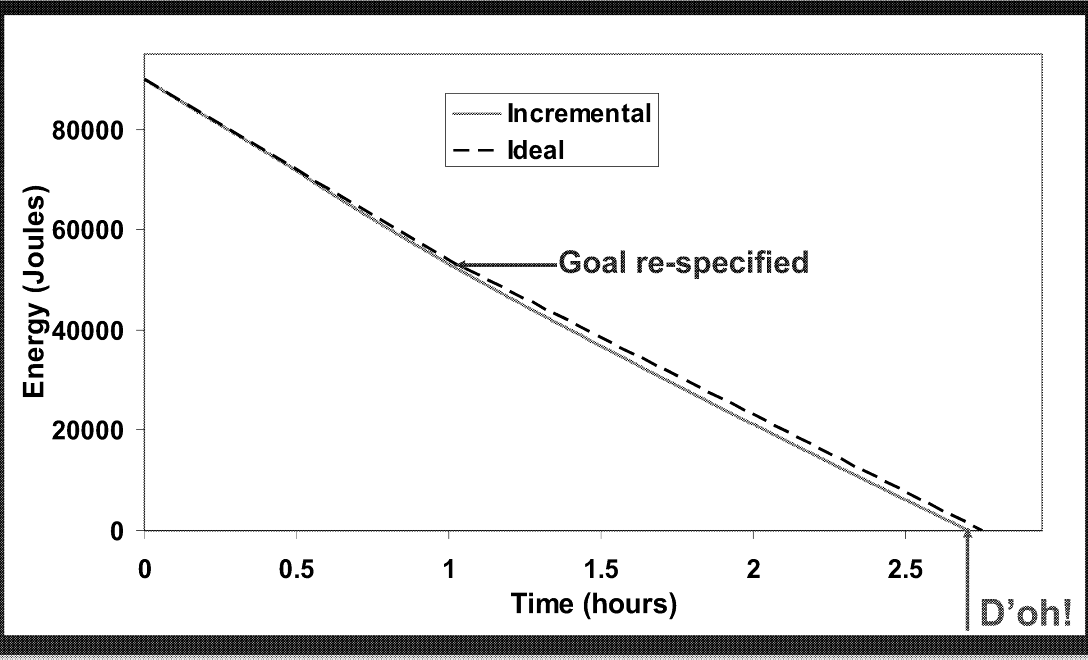
Run Web browser on ThinkPad 560X laptop

Modulated 90 KJ. energy supply

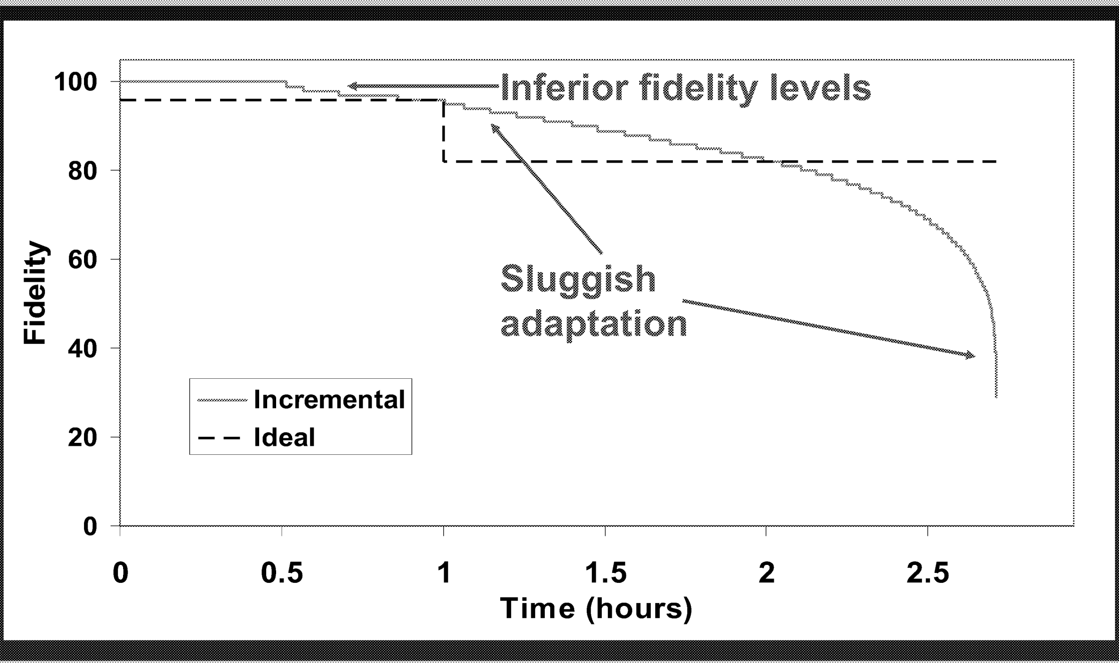
- Specify time goal of 2.5 hours
- After 1 hour, request additional 15 minutes

Compare incremental and history-based approaches

Energy Drain for Incremental Policy



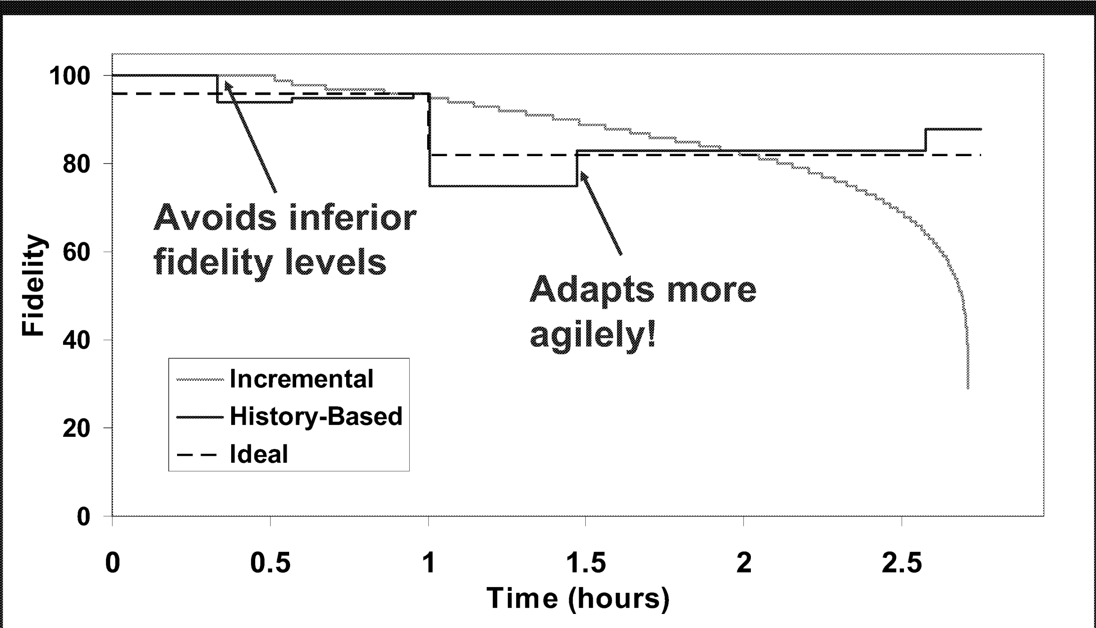
Fidelity Levels for Incremental Policy



Inferior fidelity levels

Sluggish adaptation

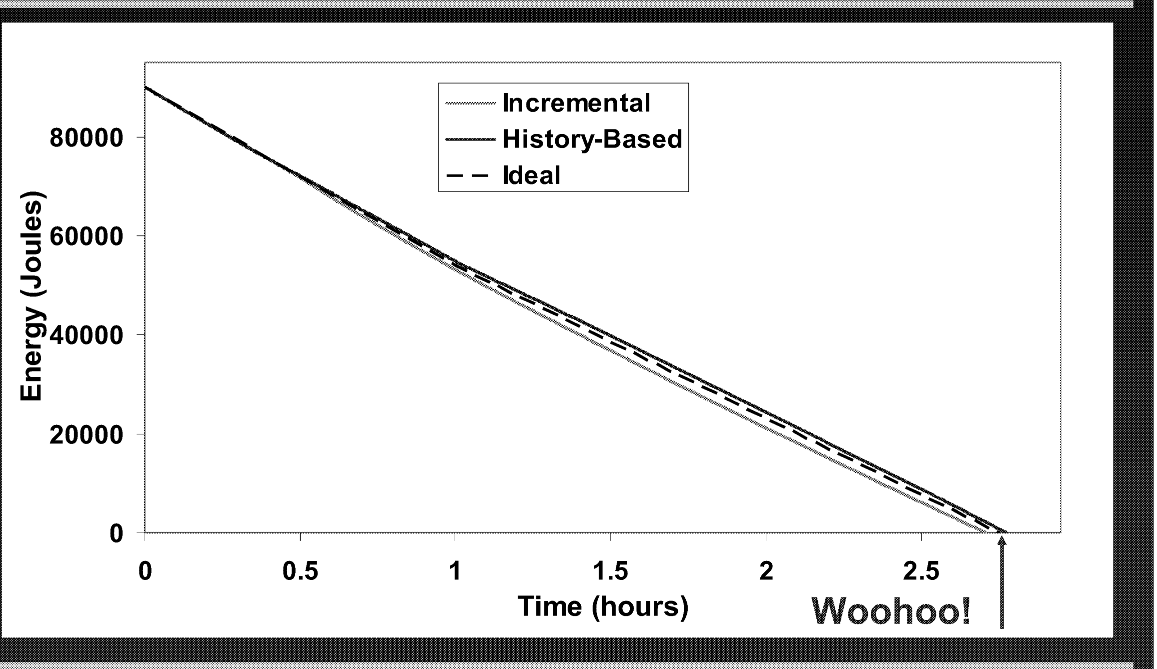
Fidelity for History-Based Policy



Avoids inferior fidelity levels

Adapts more agilely!

Energy Drain for History-Based Policy



Woohoo!

Discussion: Energy-Aware Adaptation

Operating system can effectively manage energy

- Use feedback to meet goals for battery lifetime
- Use history to decide how much apps should adapt

When multiple applications execute concurrently:

- Asks each application to sacrifice equally
- Can you imagine wanting other policies?
- How would you specify such policies?

Requires applications with multiple fidelity levels

- How else can applications conserve energy?

Roadmap

- Energy Measurement
- Energy-Aware Adaptation
- Operating System Support for Energy Management
- Remote Execution
- Self-Tuning Power Management
- Wrap-Up

Remote Execution: An Old Friend

No magic bullet for energy reduction

- Fidelity adaptation not possible for all applications
- Remote execution offers a different possibility

Very old solution for providing performance improvement

- Partition application
- Do compute-intensive portion on remote server

Can we use the same idea for energy conservation?

Remote Execution: Possibilities

Rudenko et al. examined several candidate applications:

- Compilation, Gaussian elimination, text processing
- Remote execution saves power for the first two
- Remote execution not helpful for text processing

Obstacles to using remote execution for power savings:

- Communication uses significant power
- Poor hardware power mgmt. means mobile computer still uses substantial power during remote processing
- Network interference can negate power savings

What other obstacles do you see?

Remote Processing Framework

Middleware layer that supports remote execution

Determines execution location

- Execute tasks both locally and remotely
- Keep per-task statistics – mean time, std. dev.
- Use to determine future execution location

File Consistency

- Remote and local execution must be identical
- Ship locally-modified files to the server
- Ship results back to the client

Discussion: RPF

Good news:

- Can save significant energy for some tasks
- Hardware power management has improved

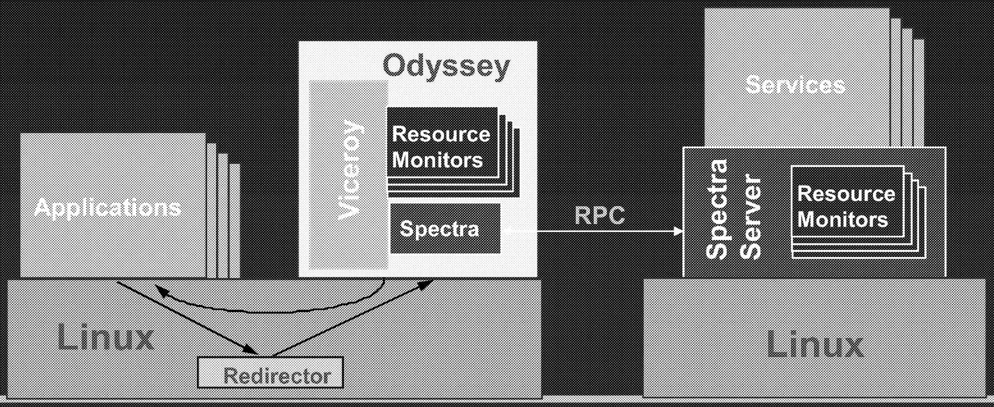
What is lacking?

- Must better estimate costs of local, remote execution
- Hybrid forms of remote execution possible
- Must balance performance and energy conservation
- Quality should also be considered

Spectra Architecture

Spectra provides:

- *Mechanism for invoking remote services*
- *Support for ensuring data consistency*
- Advice about how and where to execute operations



Data Consistency

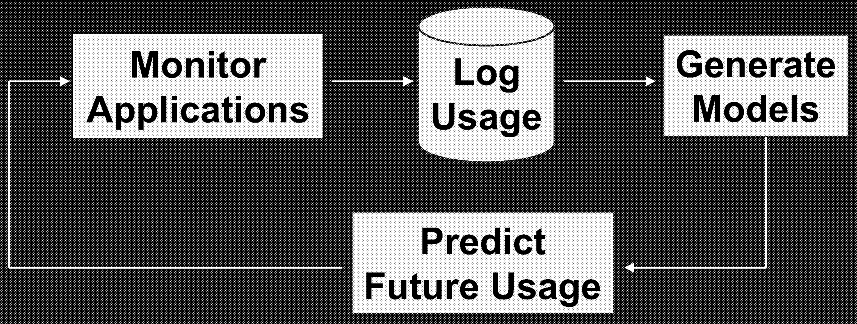
Application binaries, data reside in Coda DFS
- same file system image on client, servers
- support for poor network connections

Coda relaxes consistency for performance

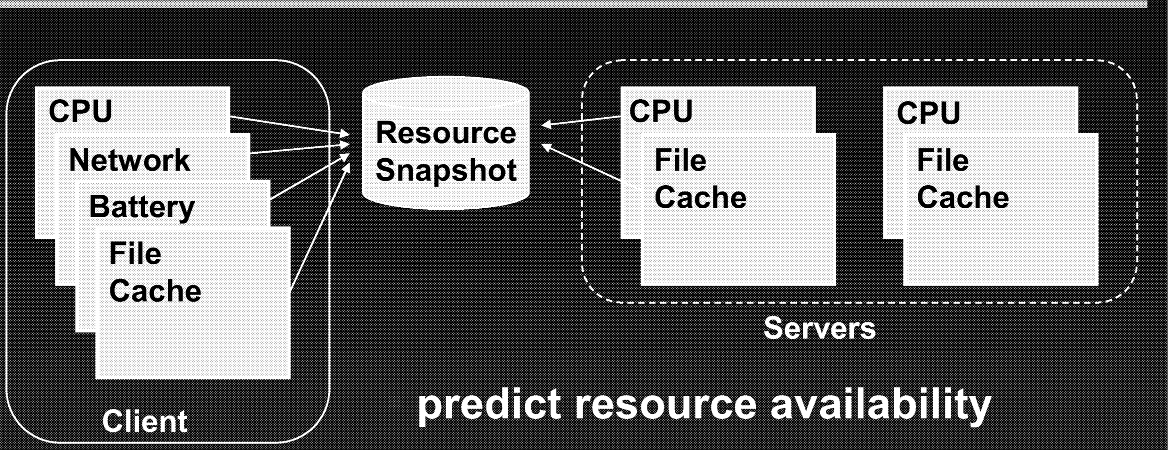
For remote operations, Spectra:
- predicts files that will be accessed
- triggers reintegration of modifications
- executes the remote operations

Self-Tuning

Model resource usage rather than performance
- Cannot ask applications to specify usage
- Instead, Spectra learns resource usage:



Resource Monitors



predict resource availability

measure resource usage

Modular framework: easy to add new resources, switch between measurement approaches

Programming Interface

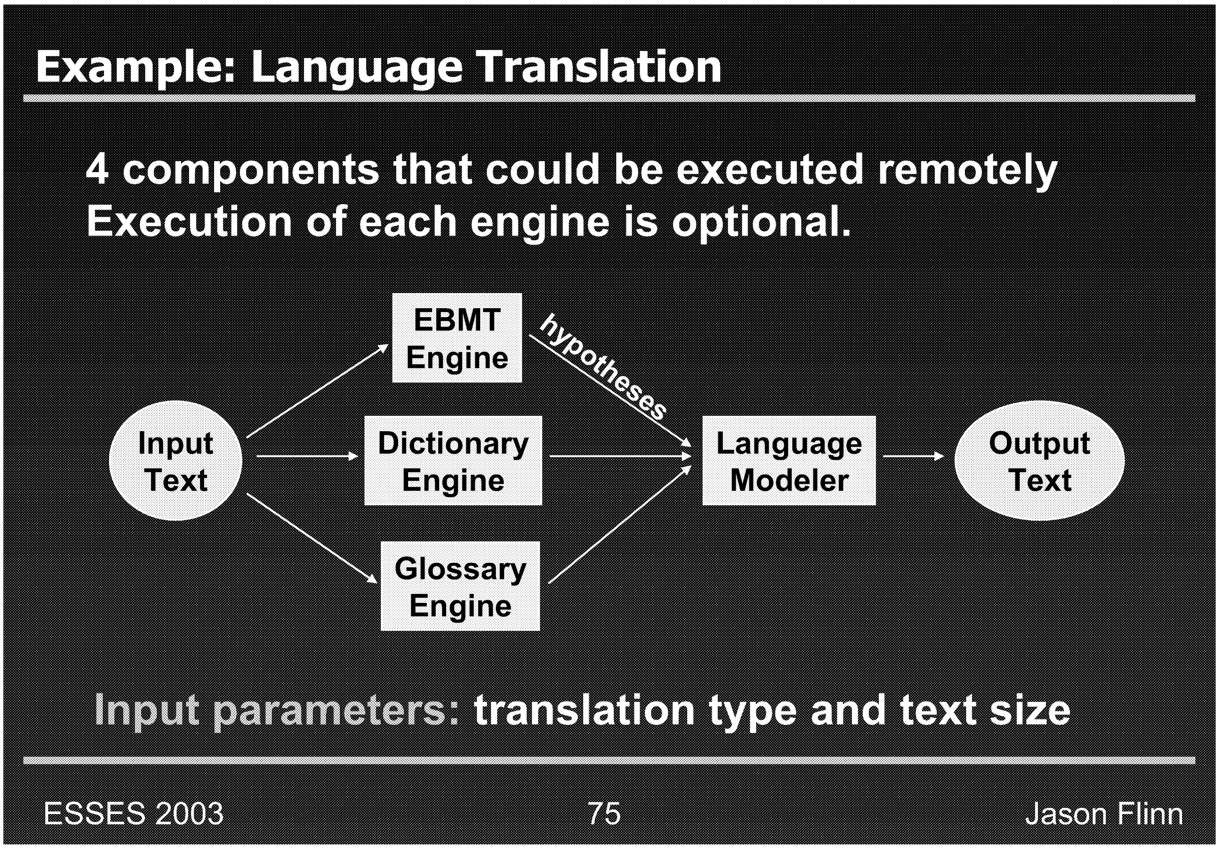
Fundamental unit is an operation

For each operation type, application describes:

- how computation may be split
- possible quality levels
- input parameters

Application signals when operations start

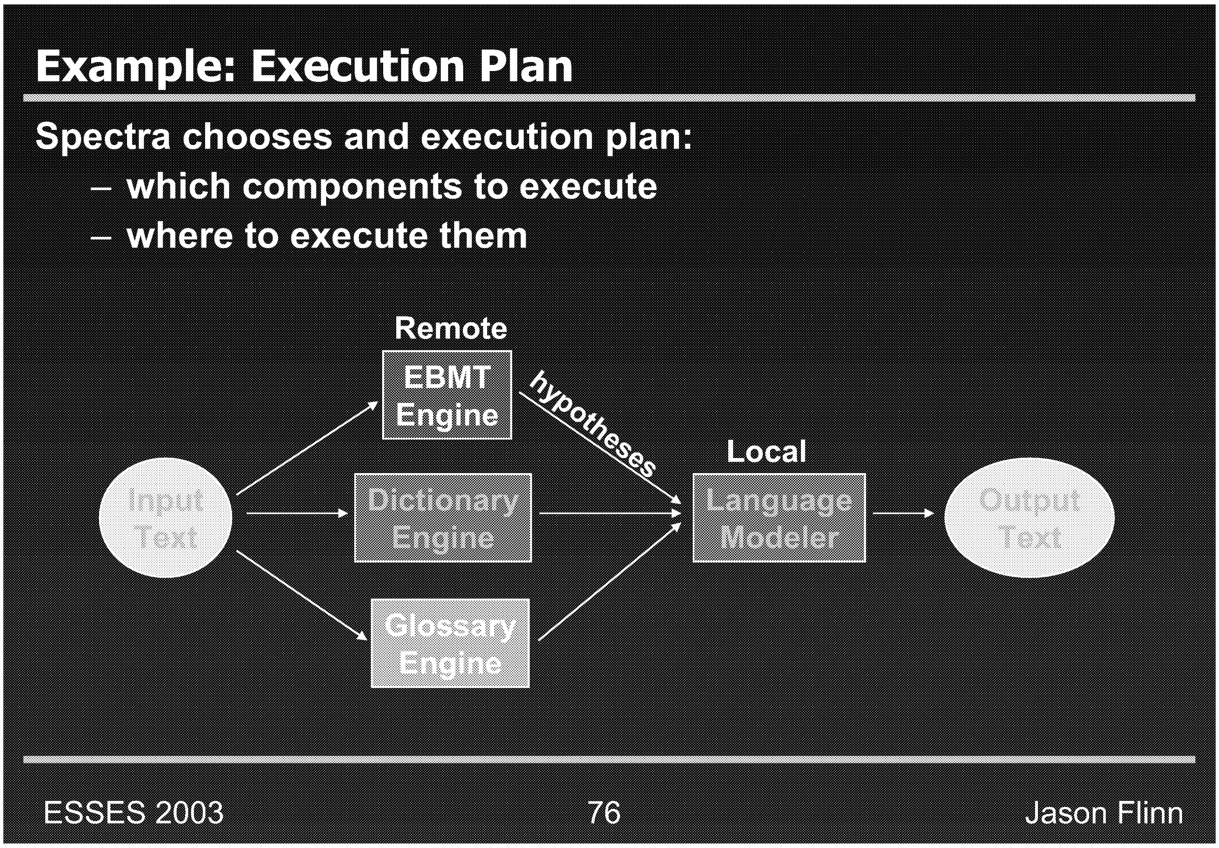
Spectra decides how and where to execute



Example: Language Translation

4 components that could be executed remotely
Execution of each engine is optional.

Input parameters: translation type and text size

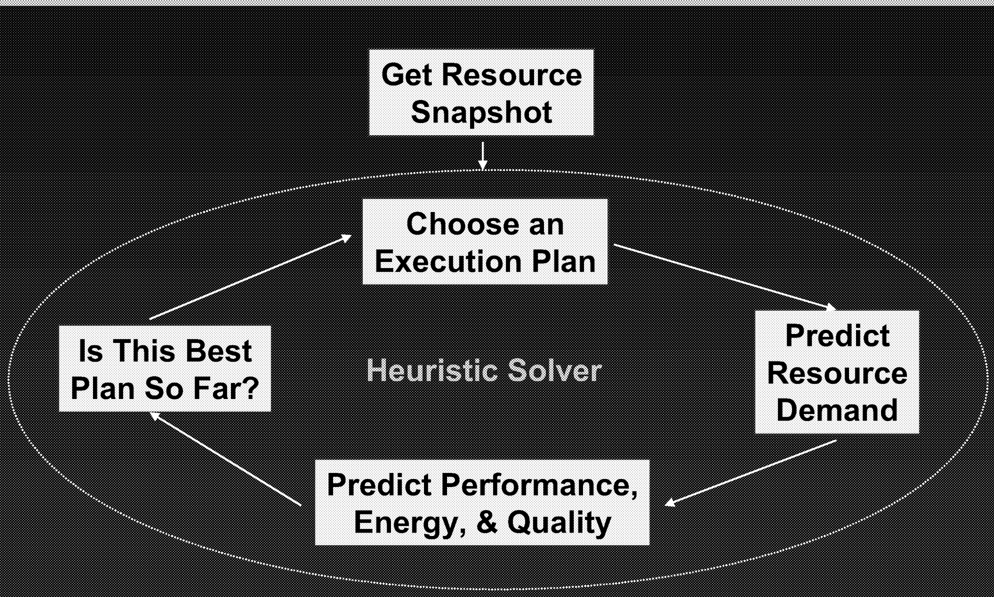


Example: Execution Plan

Spectra chooses and execution plan:

- which components to execute
- where to execute them

Choosing an Execution Plan



Selecting the best option

Spectra selects among possible location alternatives:

- Remote servers
- Execution plans

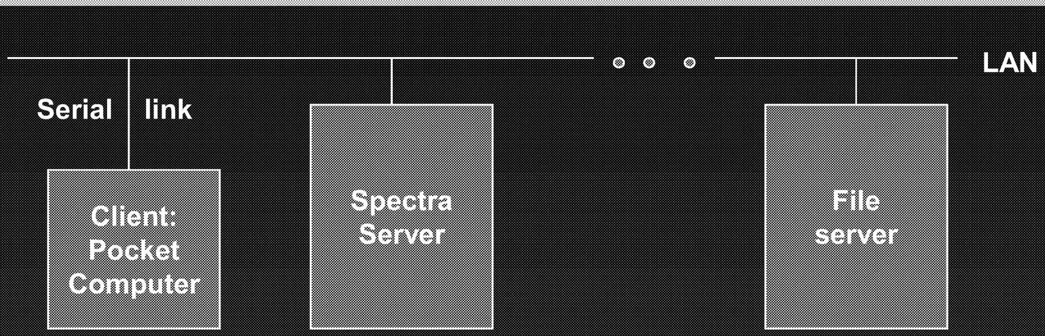
Evaluate alternatives by user metrics:

- Energy usage
- Execution time
- Fidelity

Spectra evaluates each alternative by:

1. Calculating a context-independent for each metric
2. Weighting the metric by its current importance
3. Multiplying together the weighted values

Spectra Evaluation



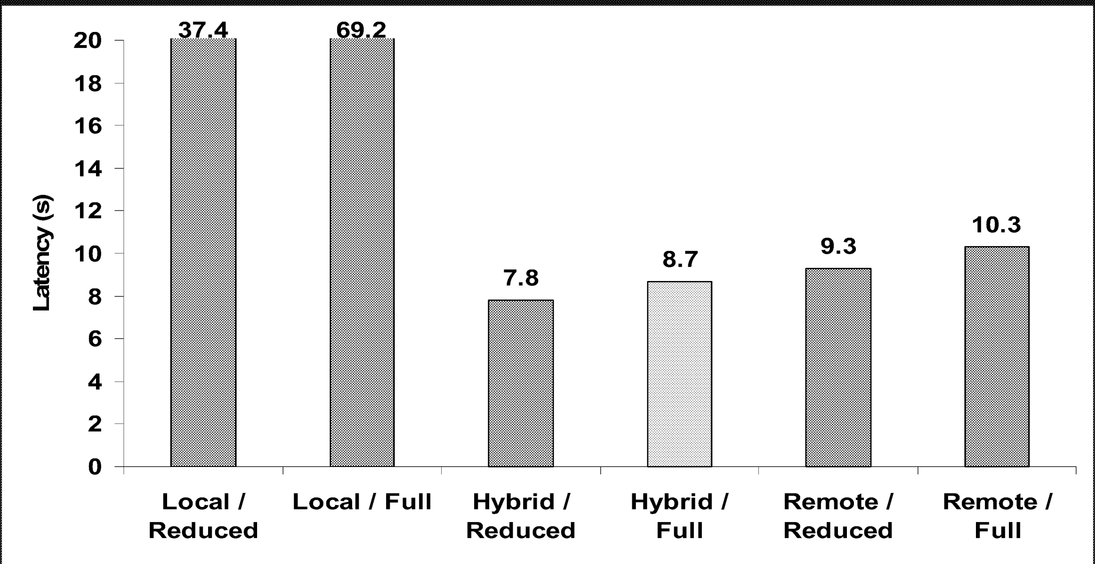
Run speech recognition with a set of 15 utterances
Specify reduced fidelity half as good as full fidelity

Experiment – recognize a new utterance:

- Scenarios vary resource availability
- Does Spectra select the best alternative?

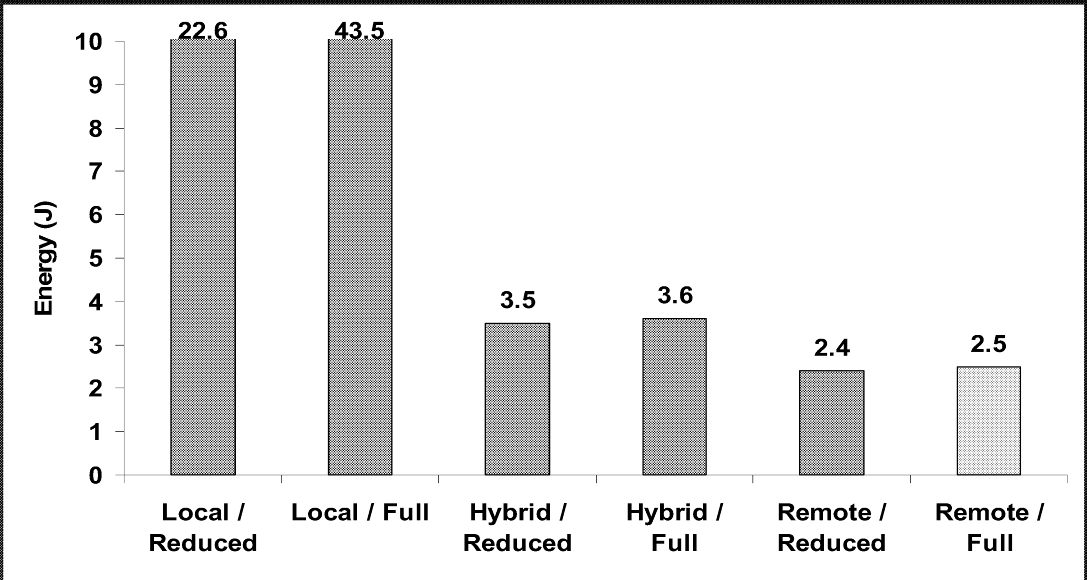
Baseline scenario

Client, server, network unloaded; client on wall power:



Battery scenario

Client battery powered with 10 hr. goal:



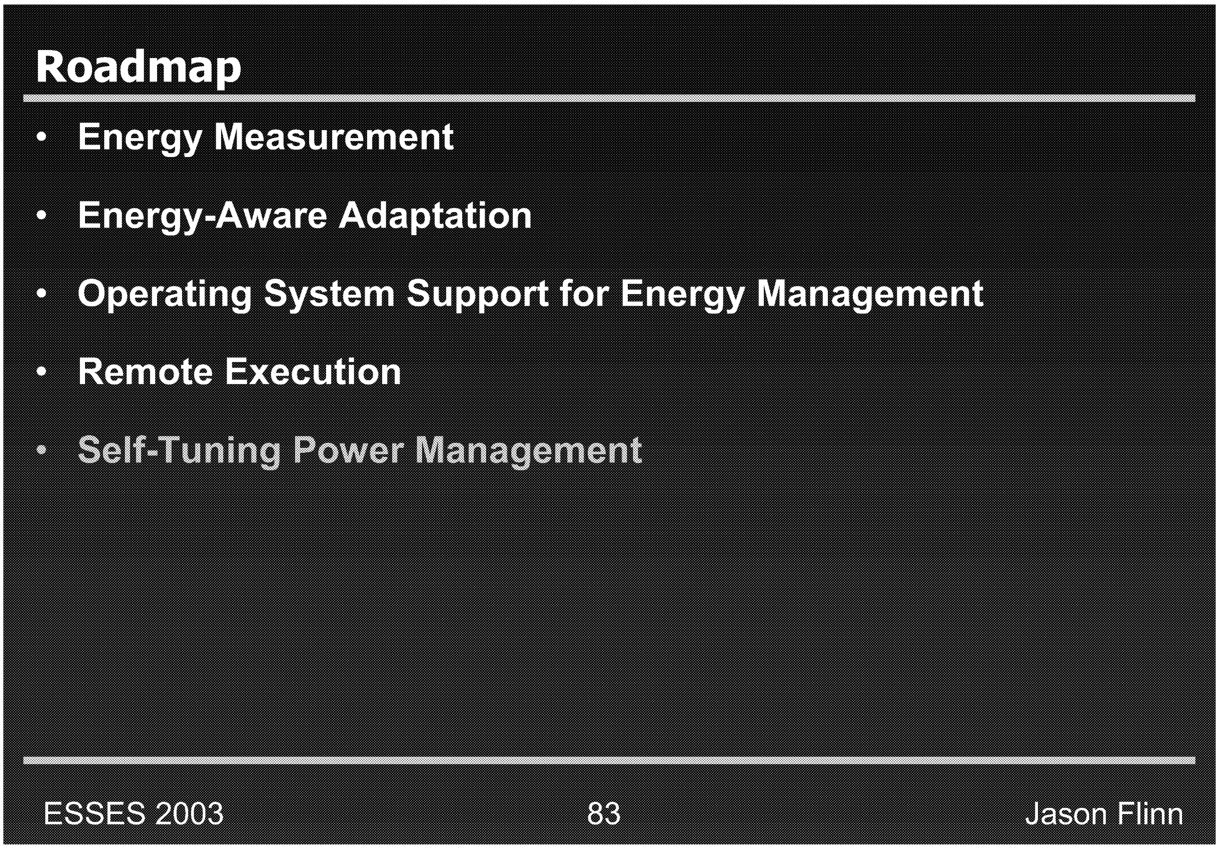
Discussion: Spectra

Positives:

- Resource-centric prediction adapts well to variability
- Distributed file system yields good consitency
- Good results for speech, graphics, other apps
- Integrates well with goal-directed adaptation

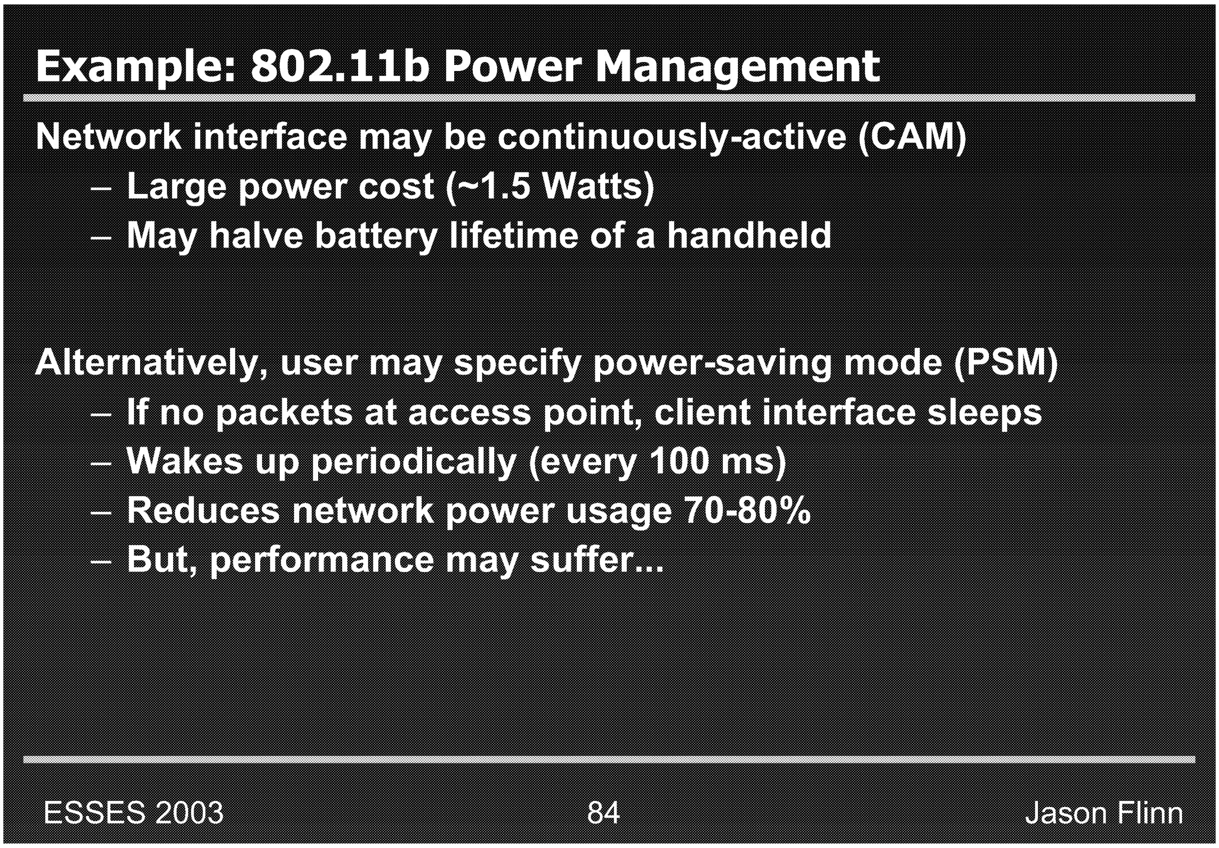
Negatives:

- Requires application modification
- Use of utility functions may be too complex



Roadmap

- Energy Measurement
- Energy-Aware Adaptation
- Operating System Support for Energy Management
- Remote Execution
- Self-Tuning Power Management



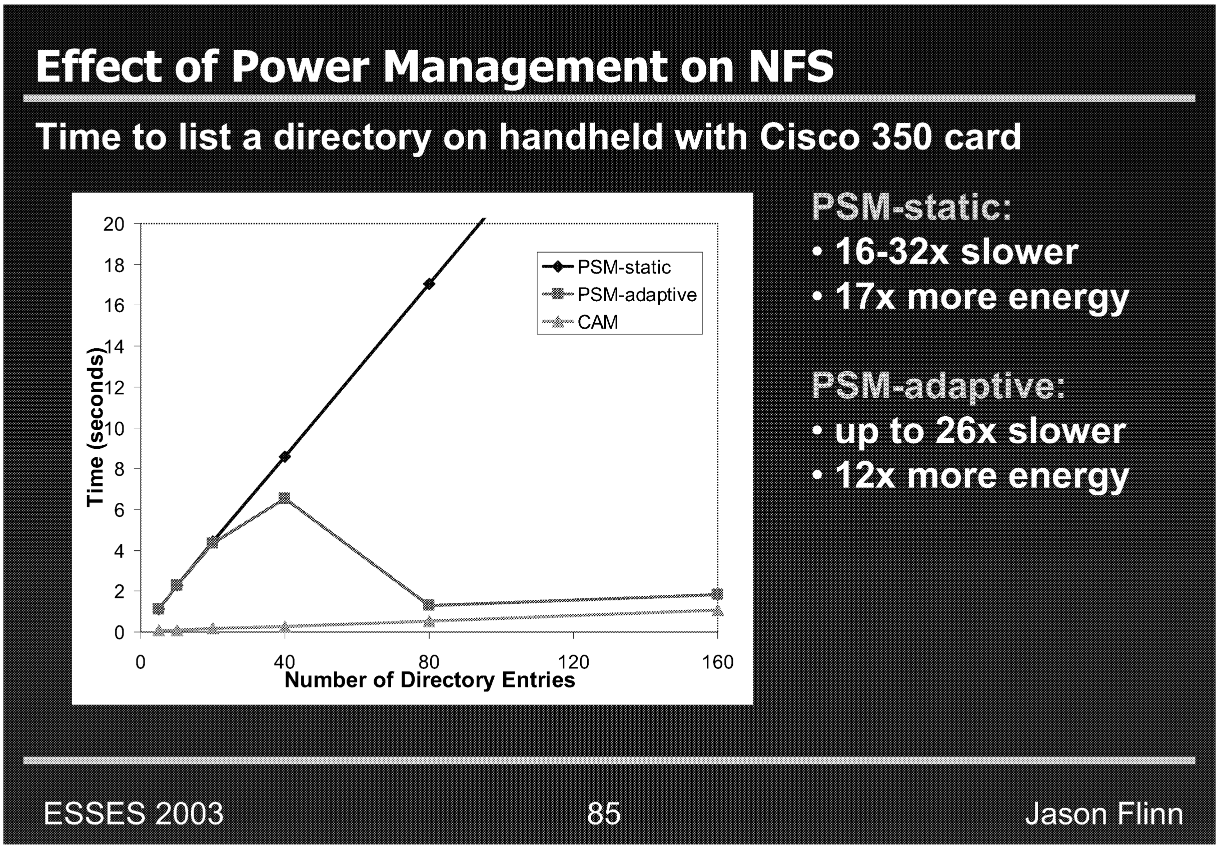
Example: 802.11b Power Management

Network interface may be continuously-active (CAM)

- Large power cost (~1.5 Watts)
- May halve battery lifetime of a handheld

Alternatively, user may specify power-saving mode (PSM)

- If no packets at access point, client interface sleeps
- Wakes up periodically (every 100 ms)
- Reduces network power usage 70-80%
- But, performance may suffer...

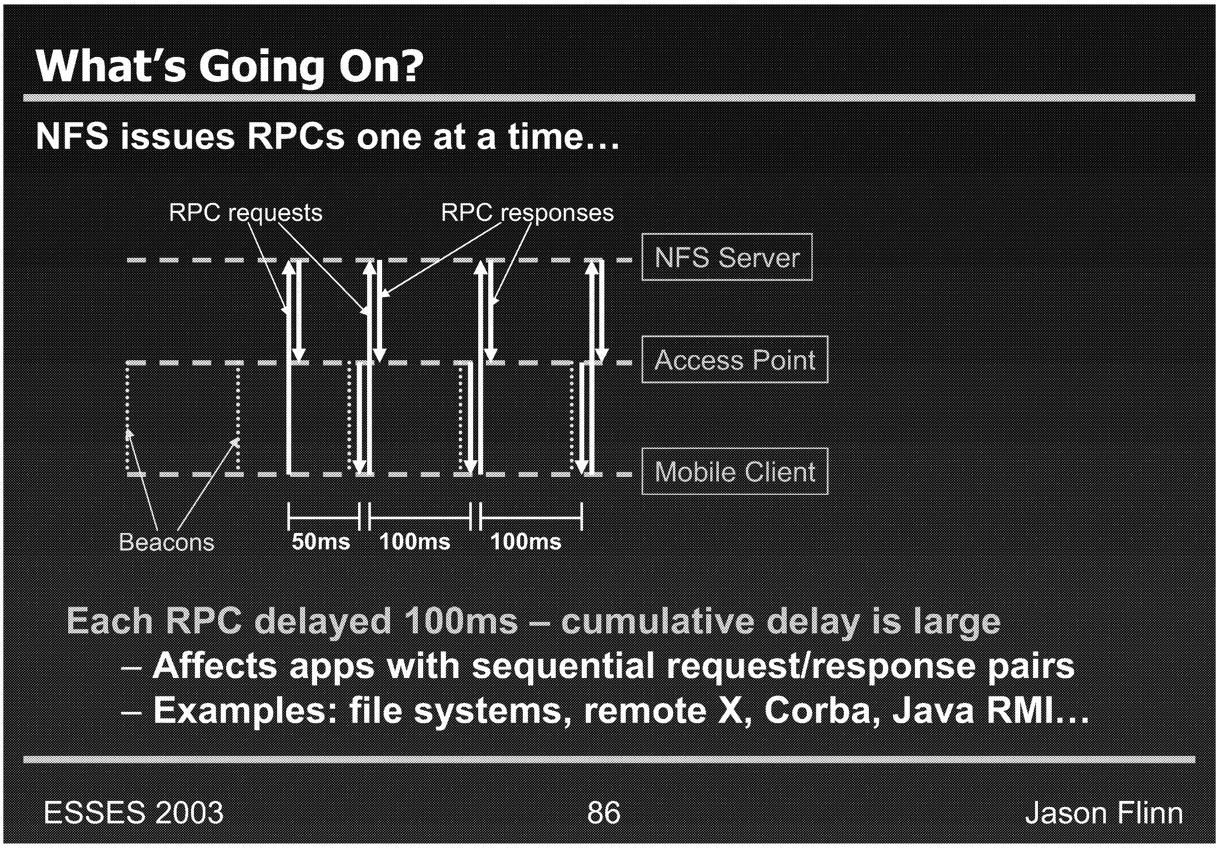


Effect of Power Management on NFS

Time to list a directory on handheld with Cisco 350 card

PSM-static:
- 16-32x slower
- 17x more energy

PSM-adaptive:
- up to 26x slower
- 12x more energy



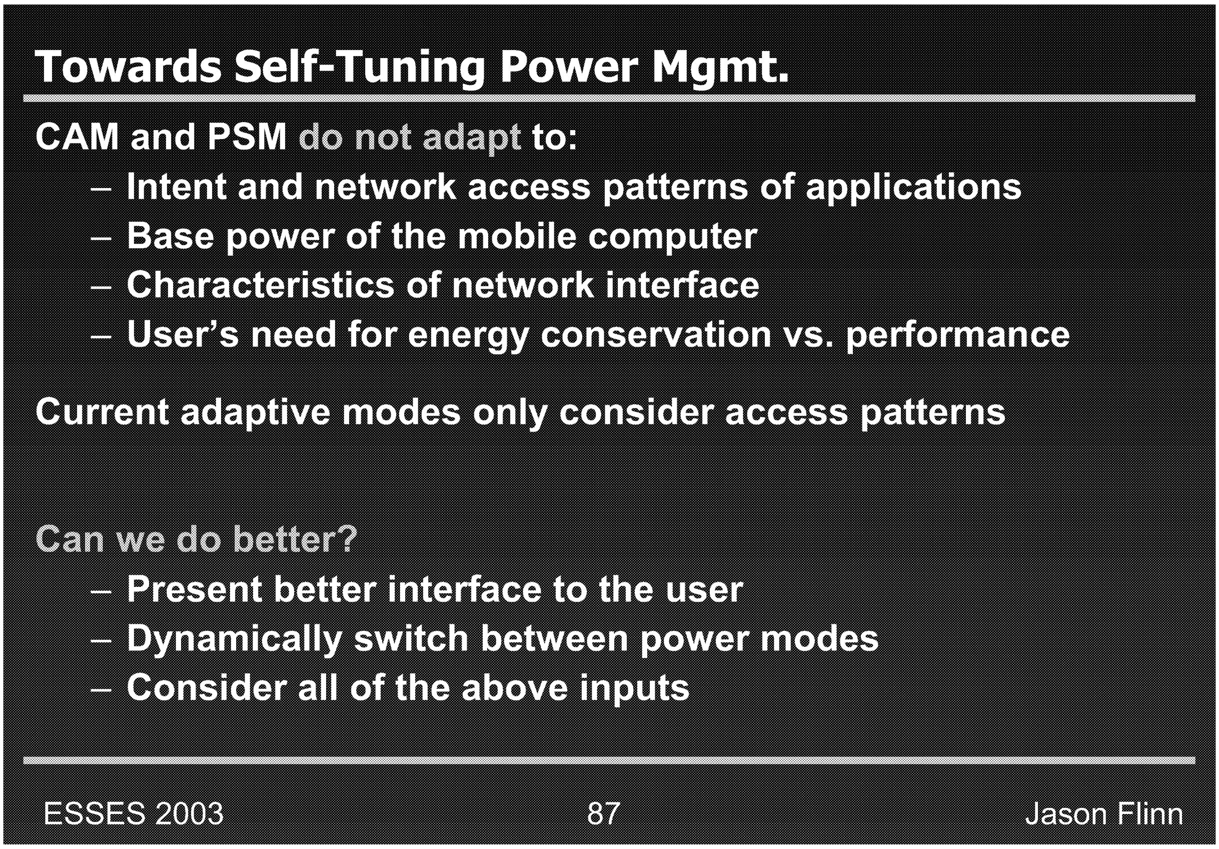
What’s Going On?

NFS issues RPCs one at a time…

Each RPC delayed 100ms – cumulative delay is large

– Affects apps with sequential request/response pairs

– Examples: file systems, remote X, Corba, Java RMI…



Towards Self-Tuning Power Mgmt.

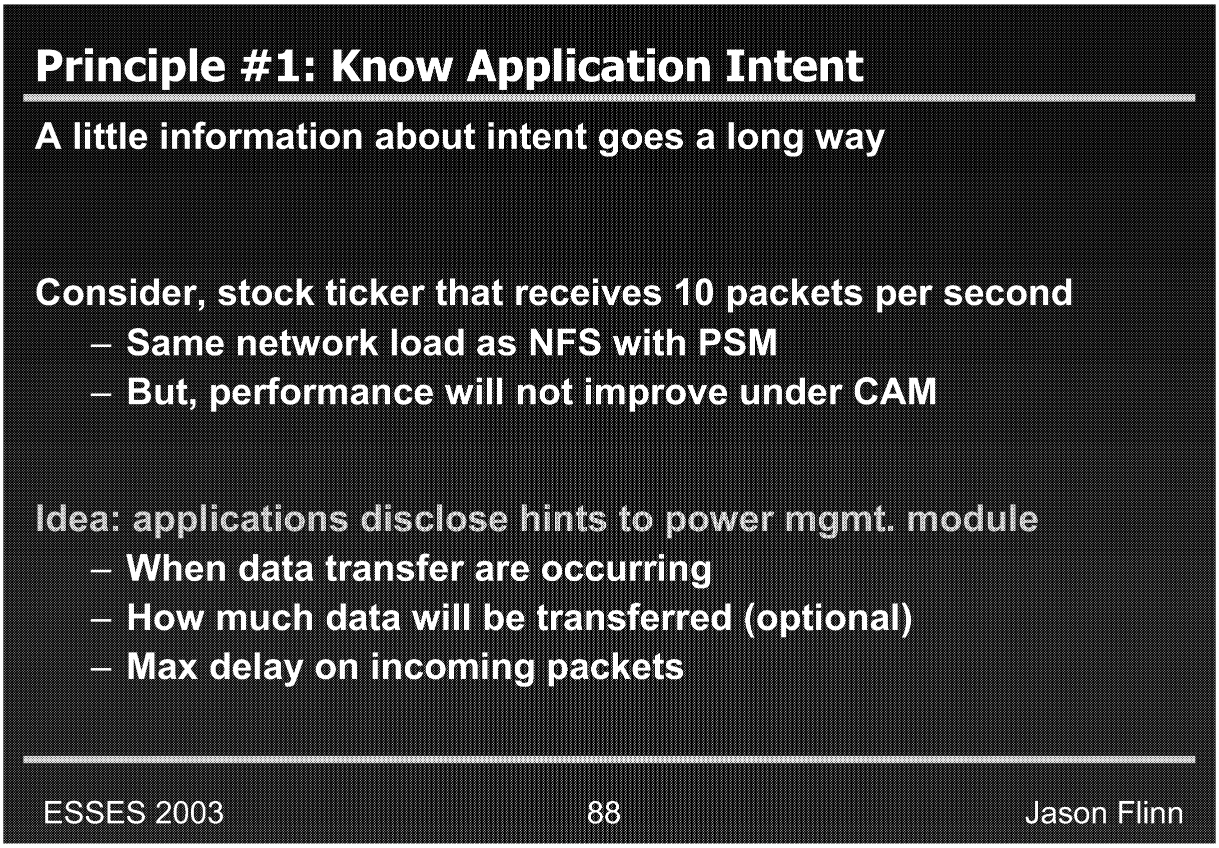
CAM and PSM do not adapt to:

- Intent and network access patterns of applications
- Base power of the mobile computer
- Characteristics of network interface
- User's need for energy conservation vs. performance

Current adaptive modes only consider access patterns

Can we do better?

- Present better interface to the user
- Dynamically switch between power modes
- Consider all of the above inputs



Principle #1: Know Application Intent

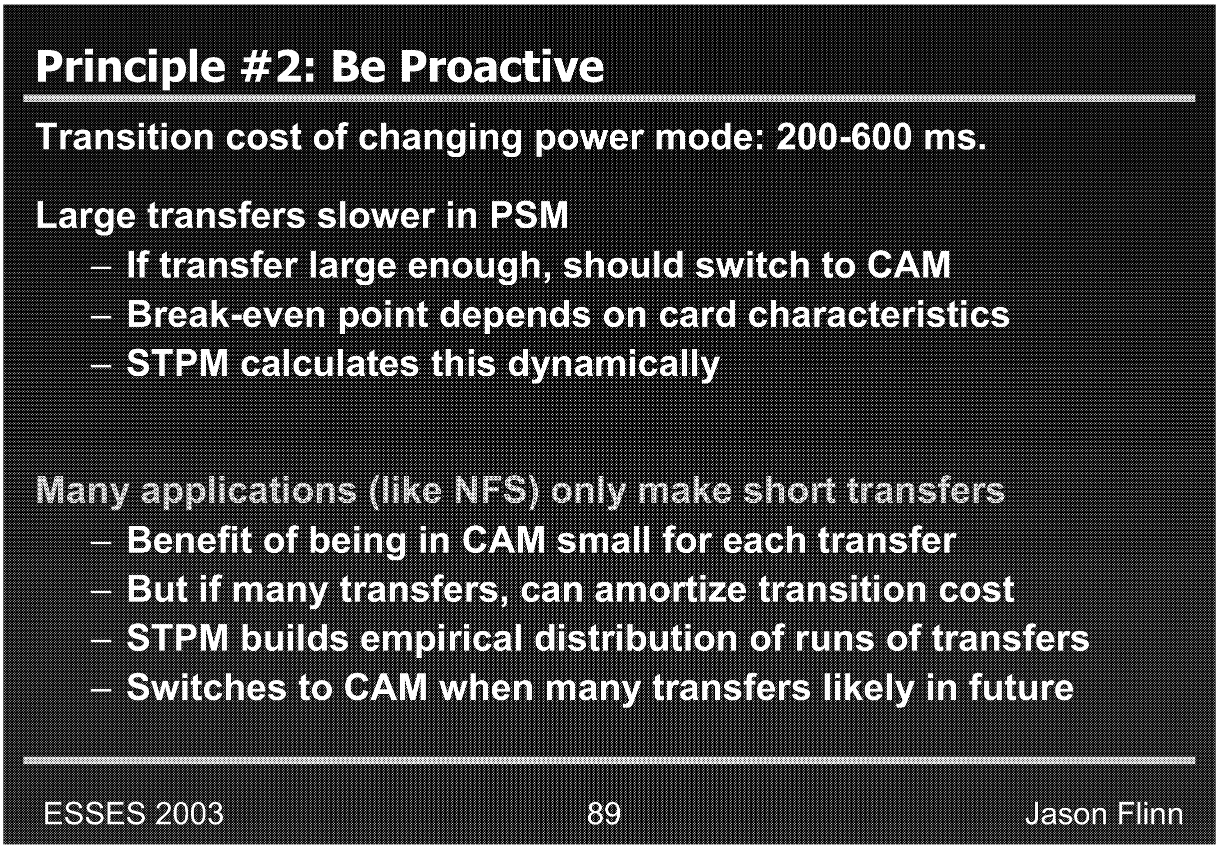
A little information about intent goes a long way

Consider, stock ticker that receives 10 packets per second

- Same network load as NFS with PSM
- But, performance will not improve under CAM

Idea: applications disclose hints to power mgmt. module

- When data transfer are occurring
- How much data will be transferred (optional)
- Max delay on incoming packets

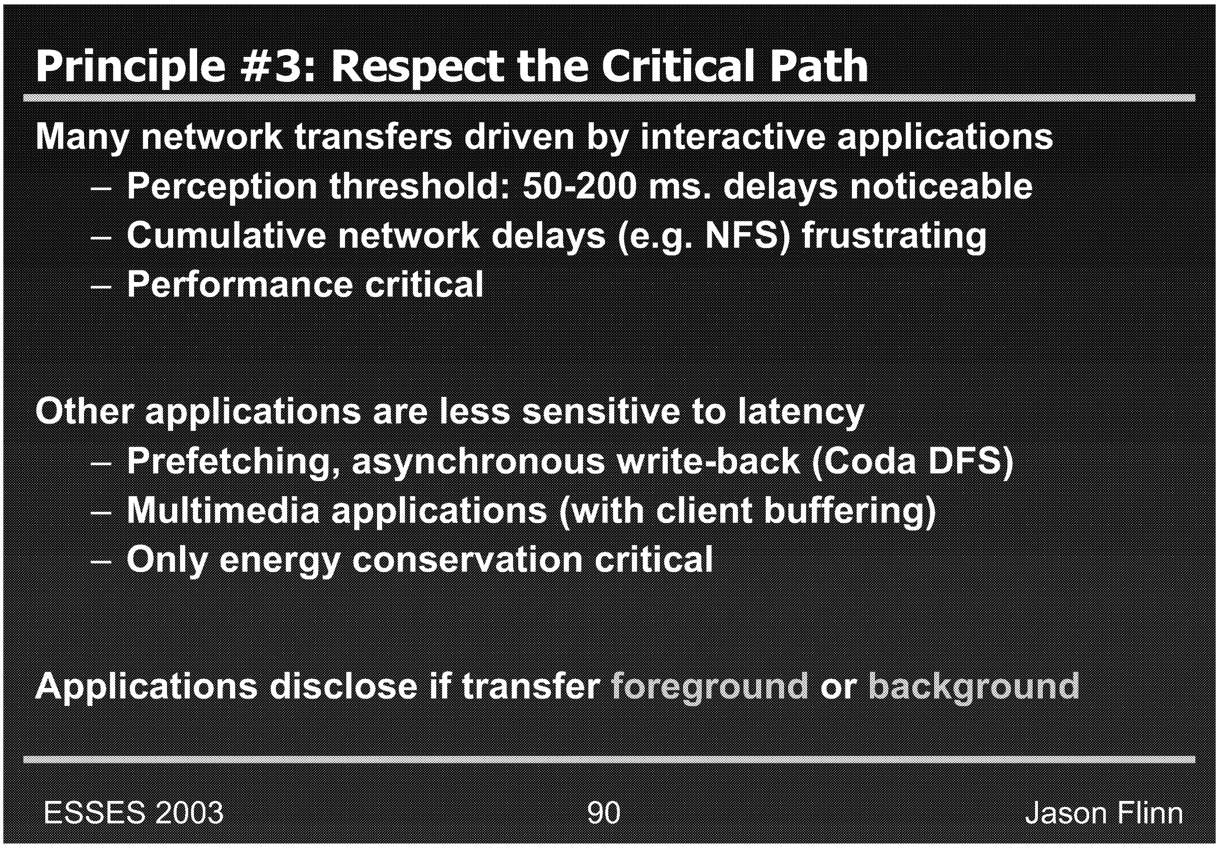


Principle #2: Be Proactive

Transition cost of changing power mode: 200-600 ms.

Large transfers slower in PSM
- If transfer large enough, should switch to CAM
- Break-even point depends on card characteristics
- STPM calculates this dynamically

Many applications (like NFS) only make short transfers
- Benefit of being in CAM small for each transfer
- But if many transfers, can amortize transition cost
- STPM builds empirical distribution of runs of transfers
- Switches to CAM when many transfers likely in future

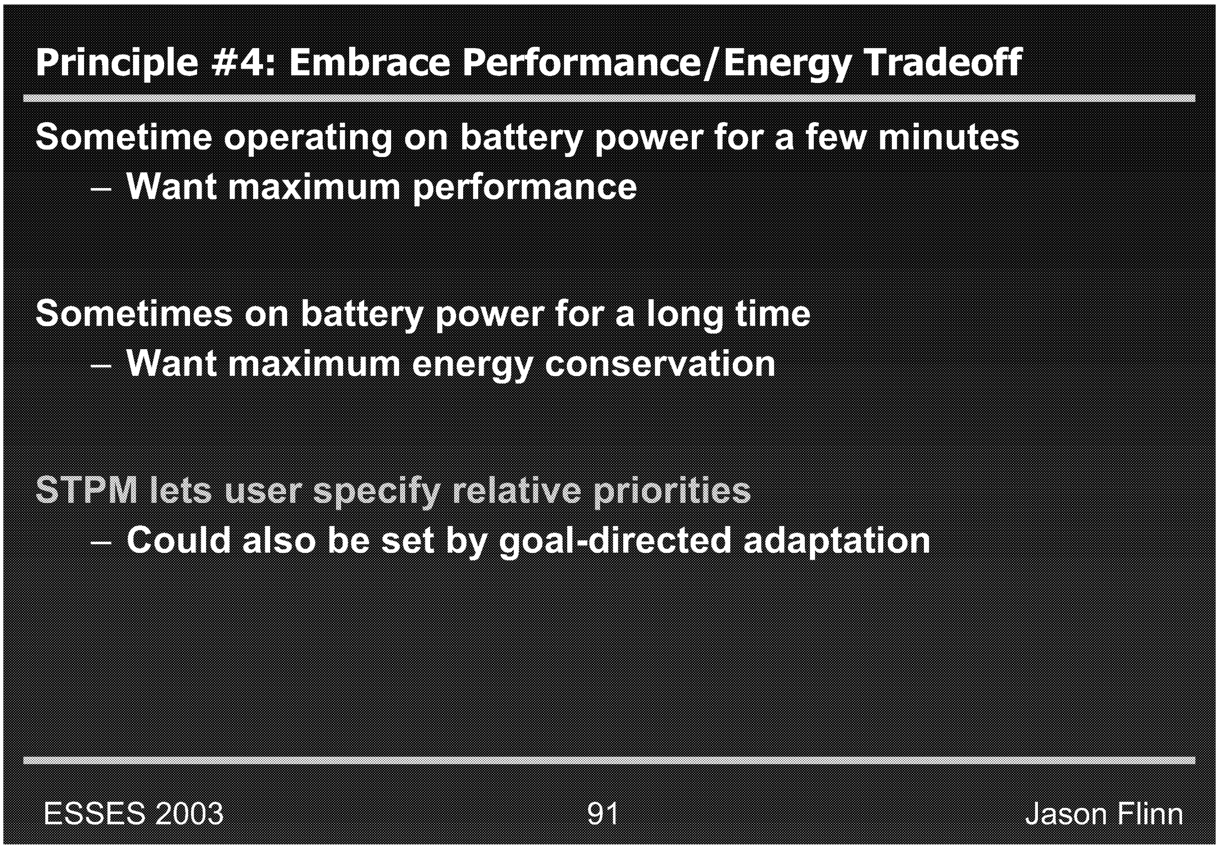


Principle #3: Respect the Critical Path

Many network transfers driven by interactive applications
- Perception threshold: 50-200 ms. delays noticeable
- Cumulative network delays (e.g. NFS) frustrating
- Performance critical

Other applications are less sensitive to latency
- Prefetching, asynchronous write-back (Coda DFS)
- Multimedia applications (with client buffering)
- Only energy conservation critical

Applications disclose if transfer foreground or background

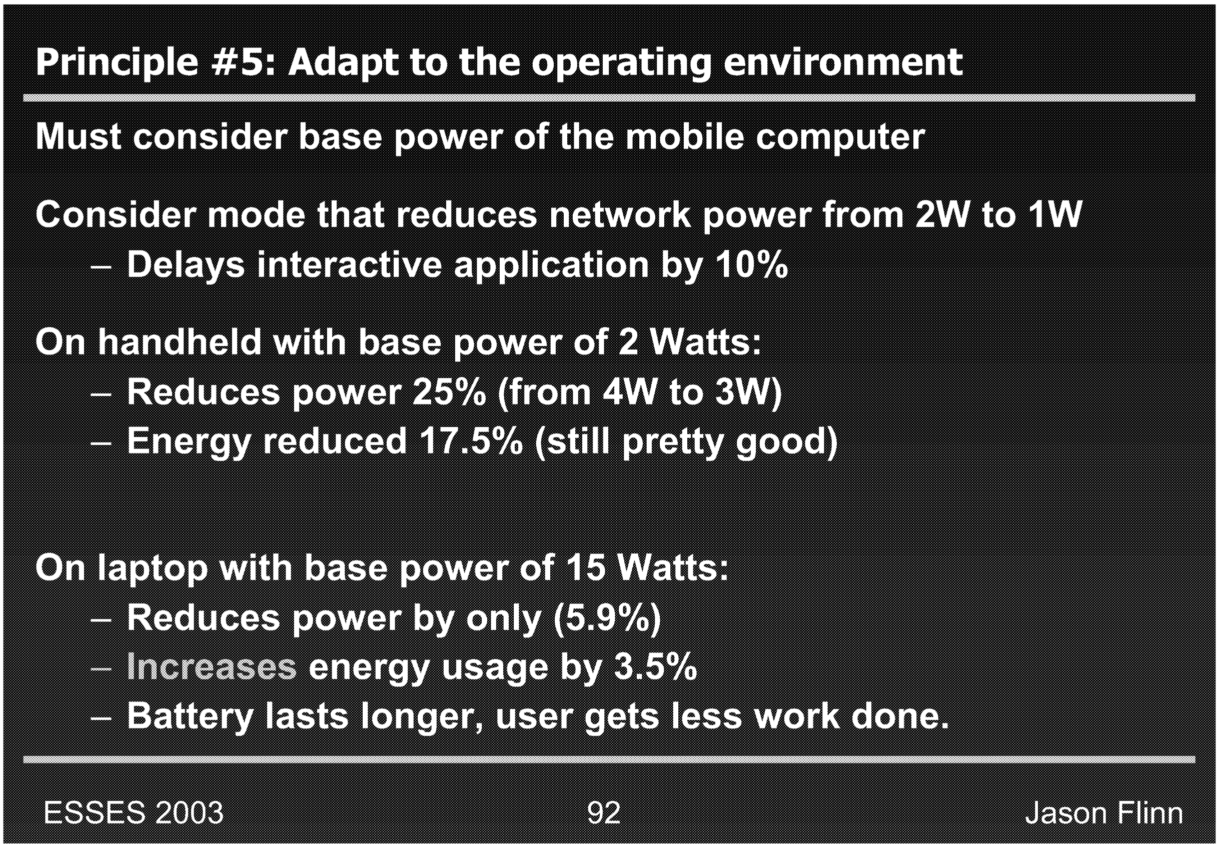


Principle #4: Embrace Performance/Energy Tradeoff

Sometime operating on battery power for a few minutes
- Want maximum performance

Sometimes on battery power for a long time
- Want maximum energy conservation

STPM lets user specify relative priorities
- Could also be set by goal-directed adaptation



Principle #5: Adapt to the operating environment

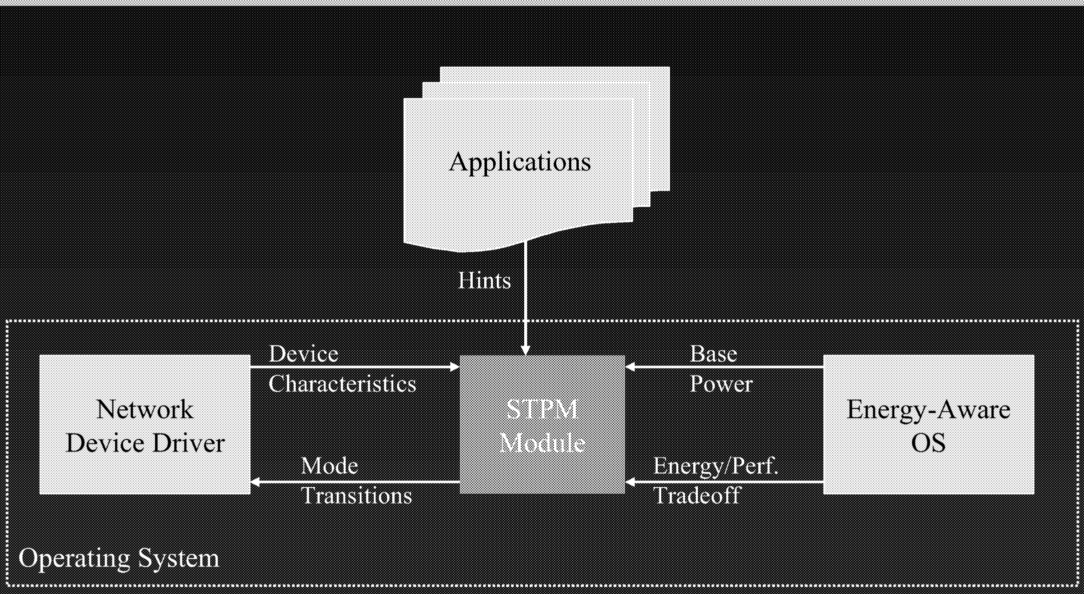
Must consider base power of the mobile computer

Consider mode that reduces network power from 2W to 1W
- Delays interactive application by 10%

On handheld with base power of 2 Watts:
- Reduces power 25% (from 4W to 3W)
- Energy reduced 17.5% (still pretty good)

On laptop with base power of 15 Watts:
- Reduces power by only (5.9%)
- Increases energy usage by 3.5%
- Battery lasts longer, user gets less work done.

STPM Architecture



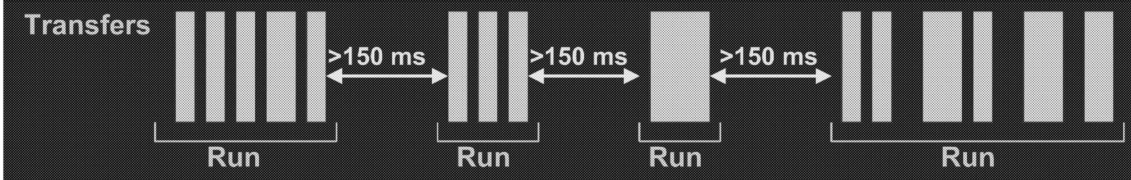
Switching to PSM

STPM switches from CAM to PSM when:

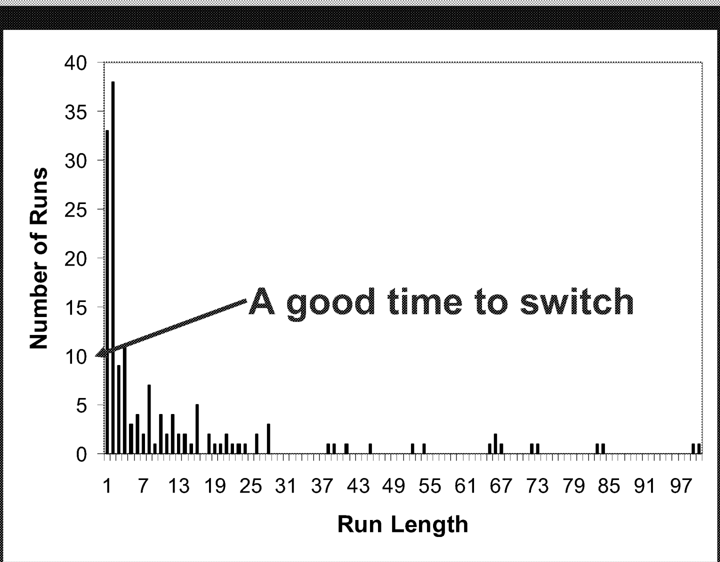
1. Any application specifies max delay < beacon period
2. Disclosed transfer size > Break-even size
3. Many forthcoming transfers are likely

First two pretty straight-forward, last more complicated…

– Generate empirical distribution of run lengths:



Intuition: Using the Run-Length History



Switch when expected # of transfers remaining in run is high

More Specifically…

For example, consider switching after 2 transfers:

Then expected time and energy to complete the next run is:

T_{PSM} * (P(L≥1) + P(L≥2)) + $T_{PSM\text{->}CAM}$ * P(L>2) + T_{CAM} * (P(L≥3) …

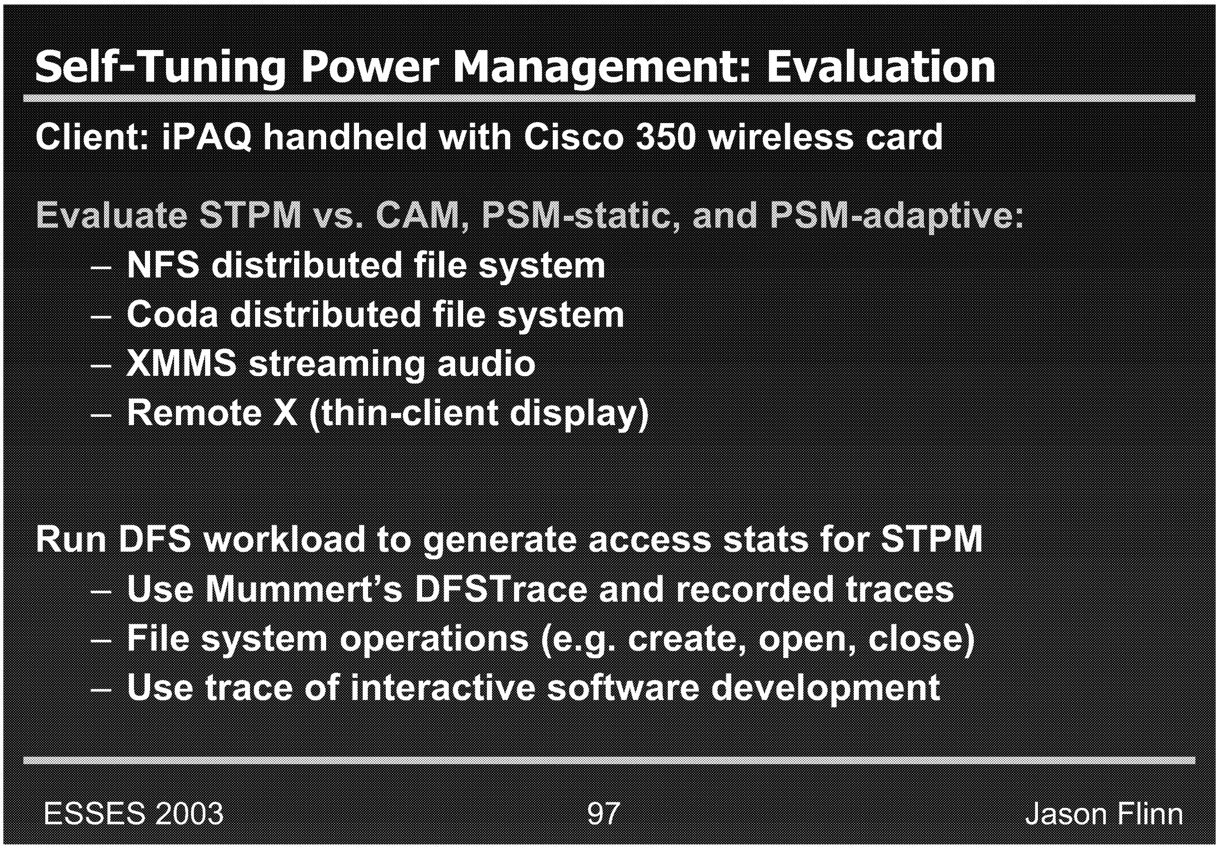
E_{PSM} * (P(L≥1) + P(L≥2)) + $E_{PSM\text{->}CAM}$ * P(L>2) + E_{CAM} * (P(L≥3) …

(Energy here is that of entire mobile computer)

Calculate expected time and energy to switch after each # of transfers

- What if these goals conflict?
- Refer to knob value for relative priority of each goal!

$$C_n = (T_n / T_{mean}) * knob + (E_n / E_{mean}) * (100 – knob)$$

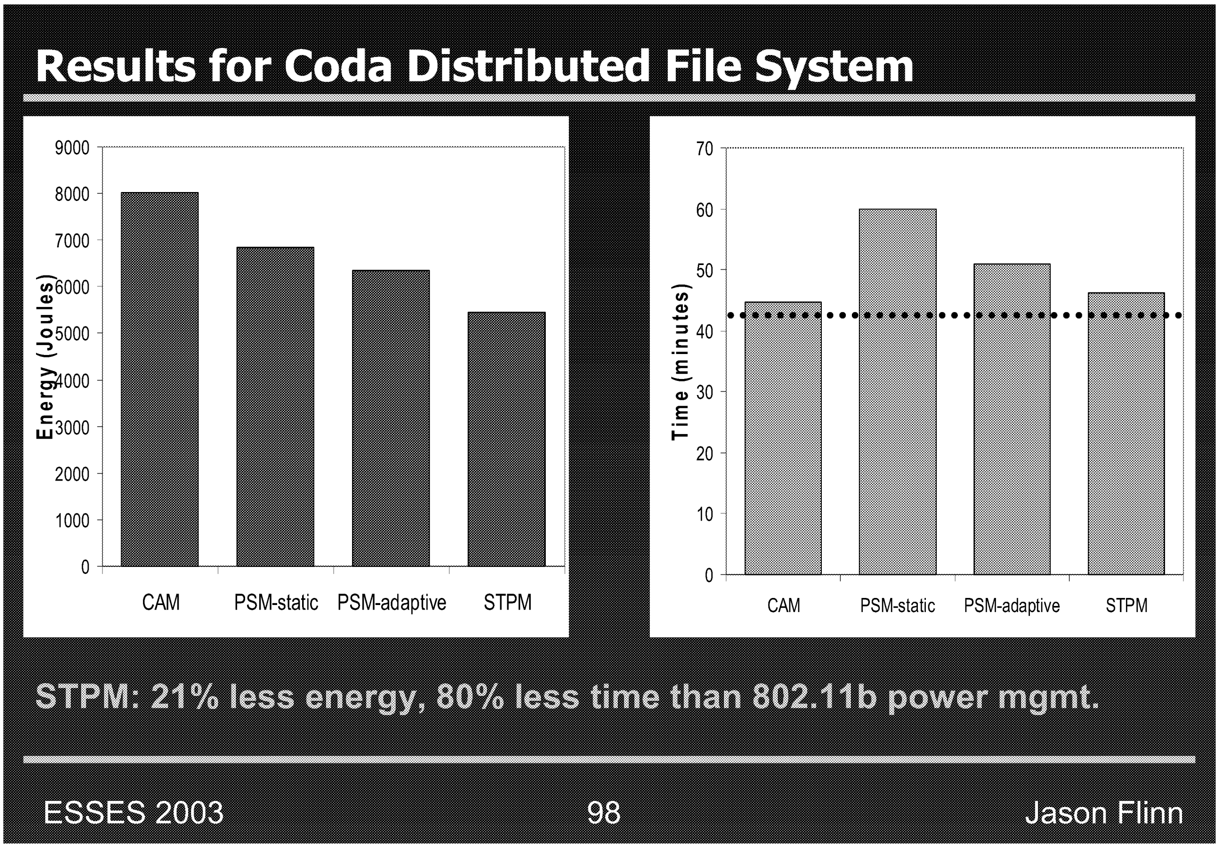


Self-Tuning Power Management: Evaluation

Client: iPAQ handheld with Cisco 350 wireless card

Evaluate STPM vs. CAM, PSM-static, and PSM-adaptive:
- NFS distributed file system
- Coda distributed file system
- XMMS streaming audio
- Remote X (thin-client display)

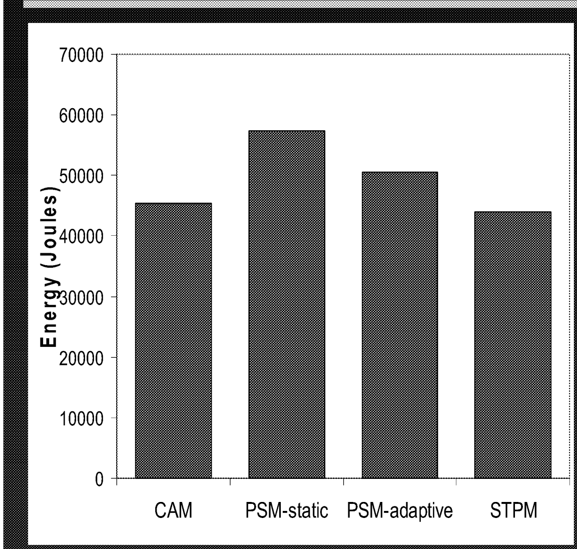
Run DFS workload to generate access stats for STPM
- Use Mummert's DFSTrace and recorded traces
- File system operations (e.g. create, open, close)
- Use trace of interactive software development

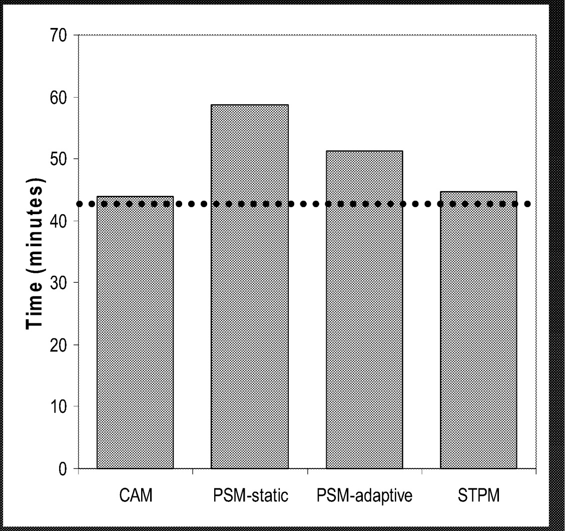


Results for Coda Distributed File System

STPM: 21% less energy, 80% less time than 802.11b power mgmt.

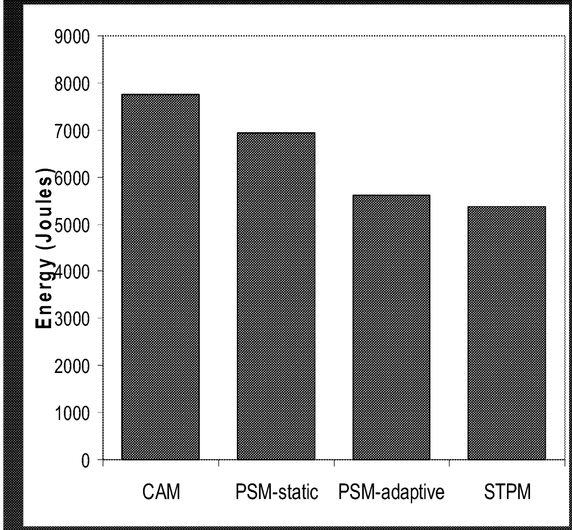
Results for Coda on IBM T20 Laptop

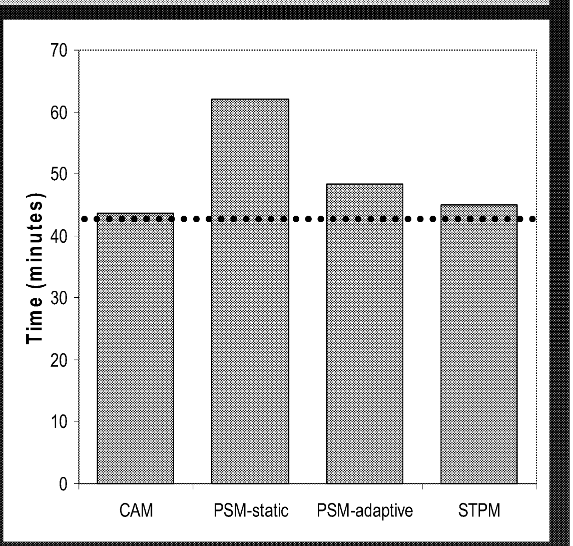




PSM-Static and PSM-Adaptive use more energy than CAM!

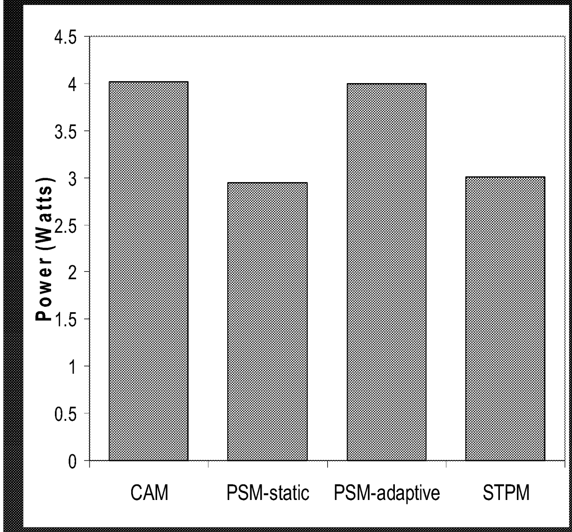
Results for NFS





STPM: 21% less energy, 80% less time than 802.11b power mgmt.

Results for XMMS Streaming Audio

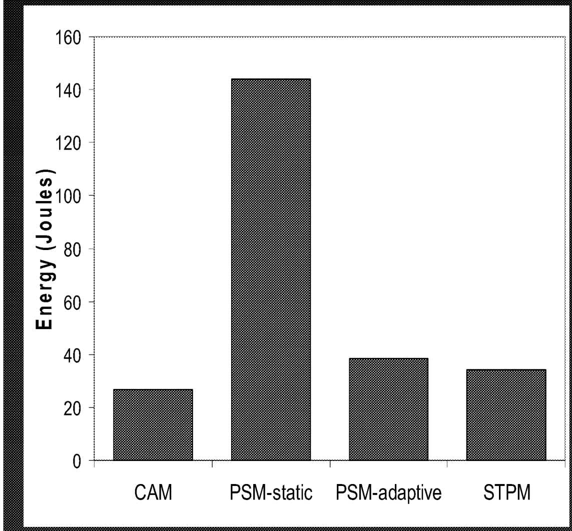


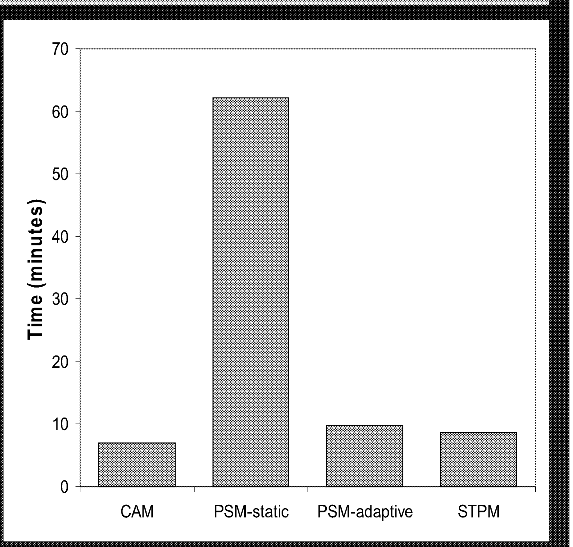
XMMS buffers data on client:

- App not latency sensitive
- PSM uses least power

STPM: 2% more power usage than PSM-static – no dropped pkts

Results for Remote X (No Think Time)





STPM uses less energy than CAM if think time > 6.5 seconds

Discussion: Self-Tuning Power Management

STPM tunes power management by considering:

- Base power of mobile computer
- Application hints and access patterns
- Relative priority of performance and energy saving
- Characteristics of network interface

What if the application does not disclose hints?

- Can insert hooks in network stack
- Issue hints on behalf of non-hinting applications

Managing Other Devices

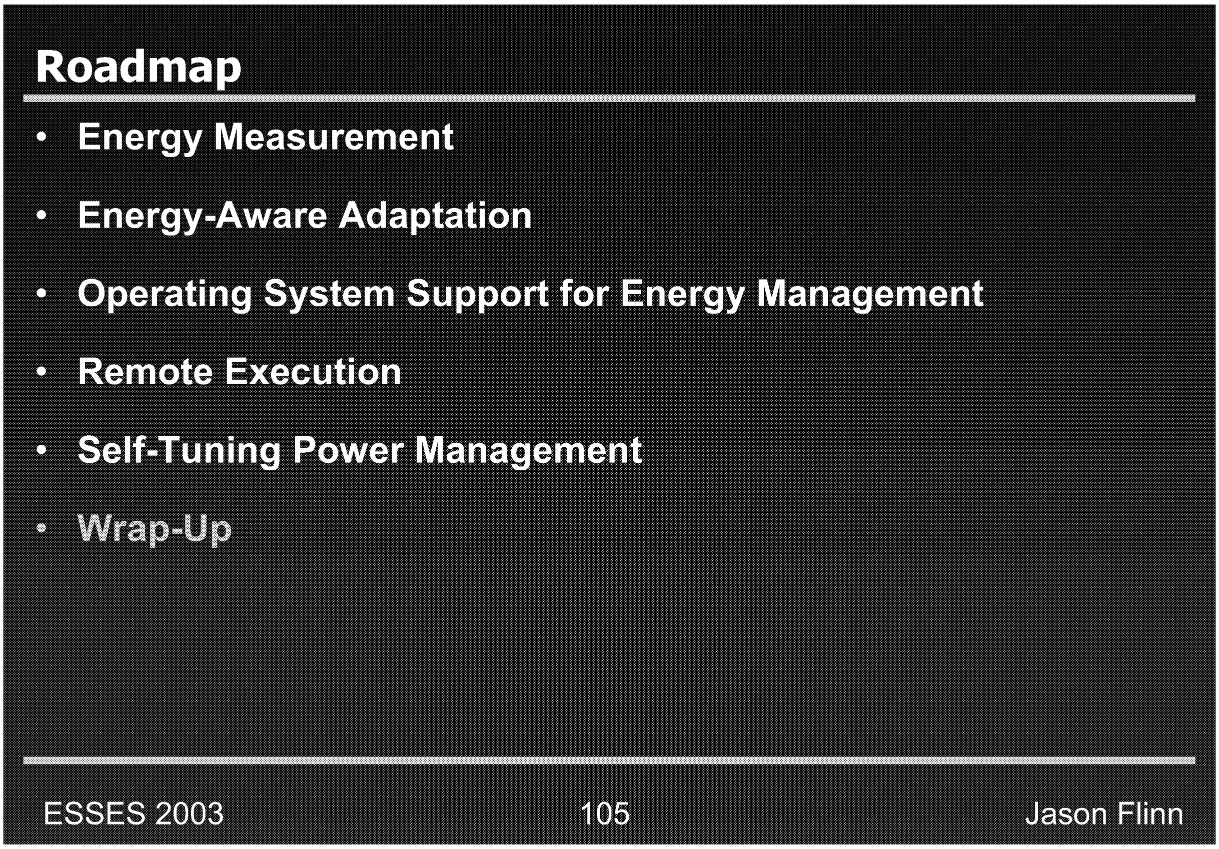
STPM well-suited for power management when:

- Performance / energy conservation tradeoff exists
- Transition costs are substantial

Consider disk power management:

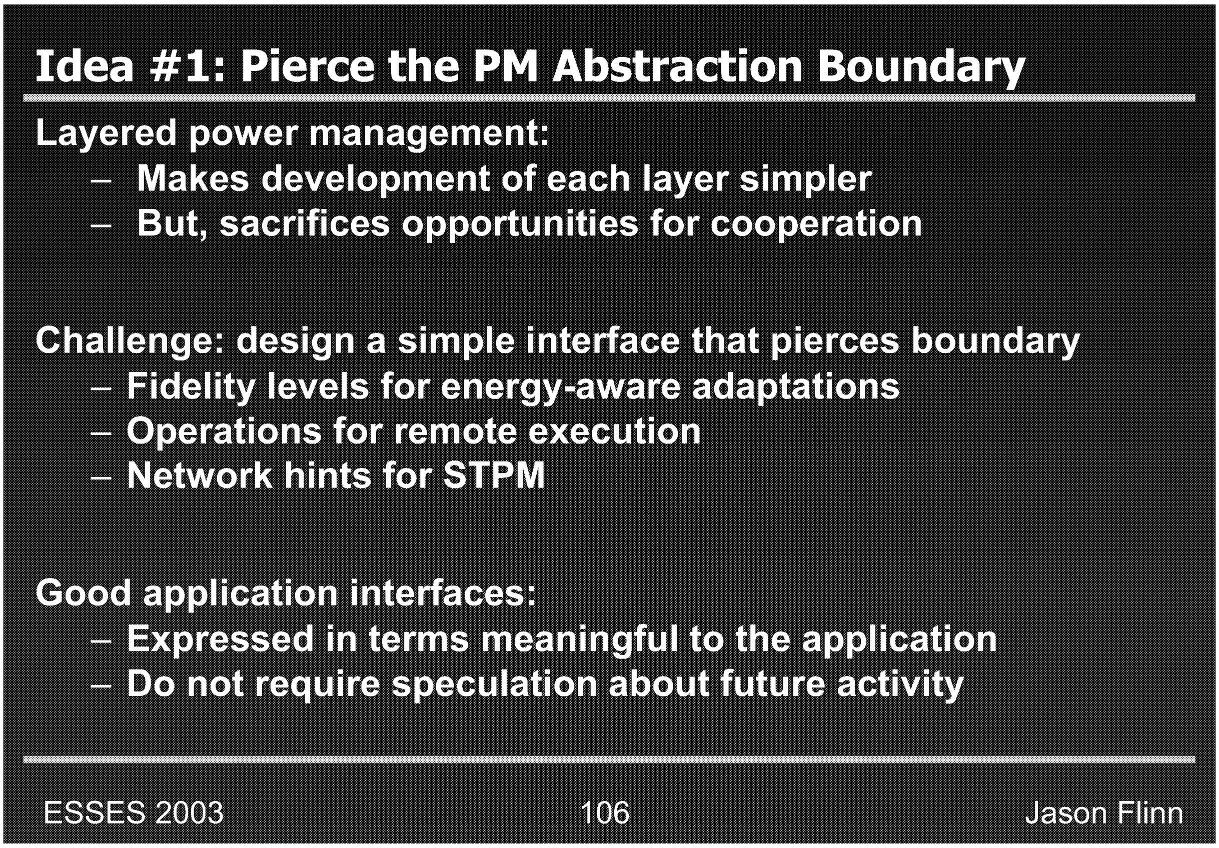
- Web browser, DFS, mobile DB cache data locally
- Hard drive spins down for power saving
- Significant transition cost to resume rot. latency
- Faster, less energy to read small object from server
- But, if many accesses, want to spin-up disk

For what other devices can STPM be applied?



Roadmap

- Energy Measurement
- Energy-Aware Adaptation
- Operating System Support for Energy Management
- Remote Execution
- Self-Tuning Power Management
- Wrap-Up



Idea #1: Pierce the PM Abstraction Boundary

Layered power management:

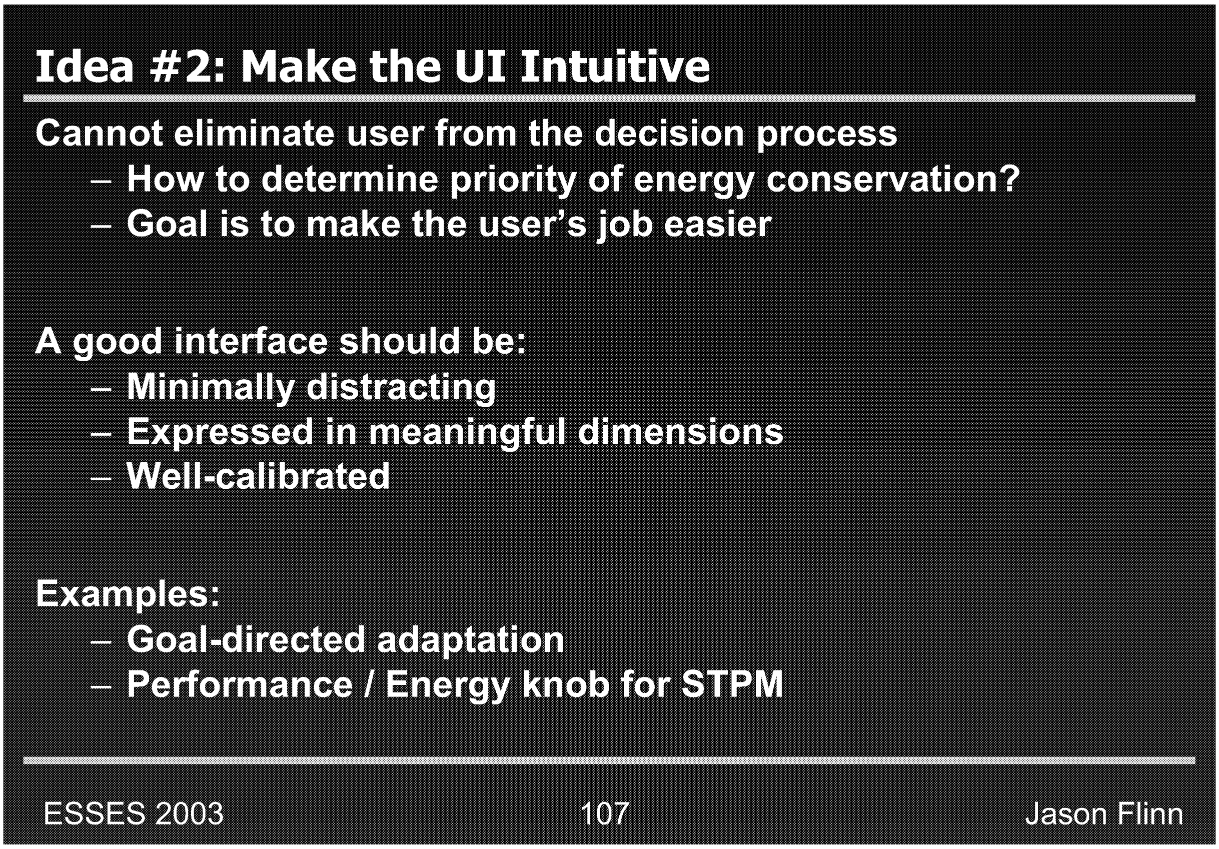
- Makes development of each layer simpler
- But, sacrifices opportunities for cooperation

Challenge: design a simple interface that pierces boundary

- Fidelity levels for energy-aware adaptations
- Operations for remote execution
- Network hints for STPM

Good application interfaces:

- Expressed in terms meaningful to the application
- Do not require speculation about future activity



Idea #2: Make the UI Intuitive

Cannot eliminate user from the decision process

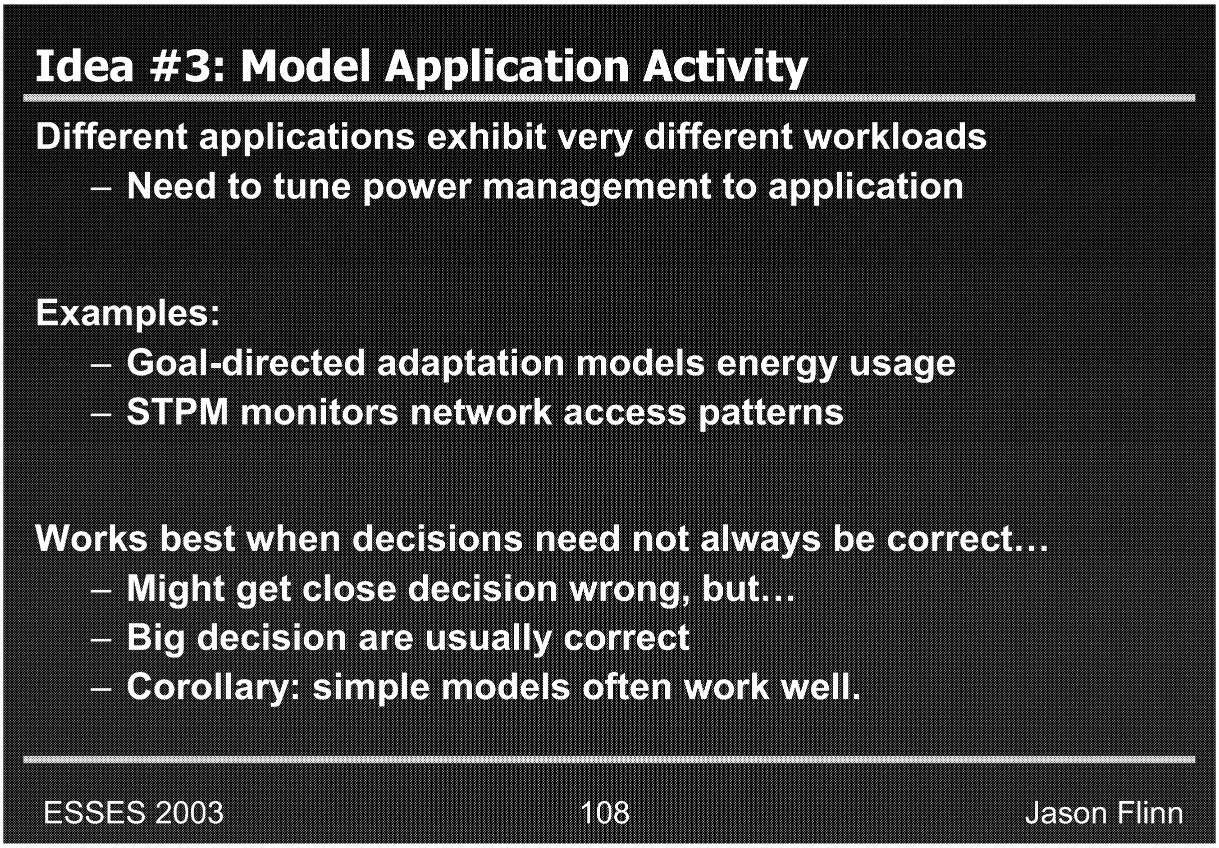
- How to determine priority of energy conservation?
- Goal is to make the user's job easier

A good interface should be:

- Minimally distracting
- Expressed in meaningful dimensions
- Well-calibrated

Examples:

- Goal-directed adaptation
- Performance / Energy knob for STPM



Idea #3: Model Application Activity

Different applications exhibit very different workloads

- Need to tune power management to application

Examples:

- Goal-directed adaptation models energy usage
- STPM monitors network access patterns

Works best when decisions need not always be correct…

- Might get close decision wrong, but…
- Big decision are usually correct
- Corollary: simple models often work well.

Idea #4: Don’t Put All Your Eggs in One Basket

No magic bullet for power management

Many approaches work well for some apps, not other:
- Energy-aware adaptation
- Remote execution
- Device power management

Best approach: Develop a toolbox of power management techniques – use them all!

Cooperative I/O

Frank Bellosa

Department of Computer Science 4 (Operating Systems)
University of Erlangen, Germany
bellosa@cs.fau.de

Cooperative I/O

Frank Bellosa

Department of Computer Science 4 (Operating Systems)
University of Erlangen, Germany
bellosa@cs.fau.de

Contents

A Outline

- Coop I/O: A Novel I/O Semantics
- Disk Energy Characterization
- Power Management of Hard Disks
- Cooperative I/O
- Conclusion

B Cooperative I/O: Principle of Operation

- “Traditional” OS power management policies:
  - ◆ Timing of disk operations issued by user applications is unknown and cannot be influenced
- Cooperative I/O: more flexible timing of disk operations

  ➔*deferrable* and *abortable* I/O requests

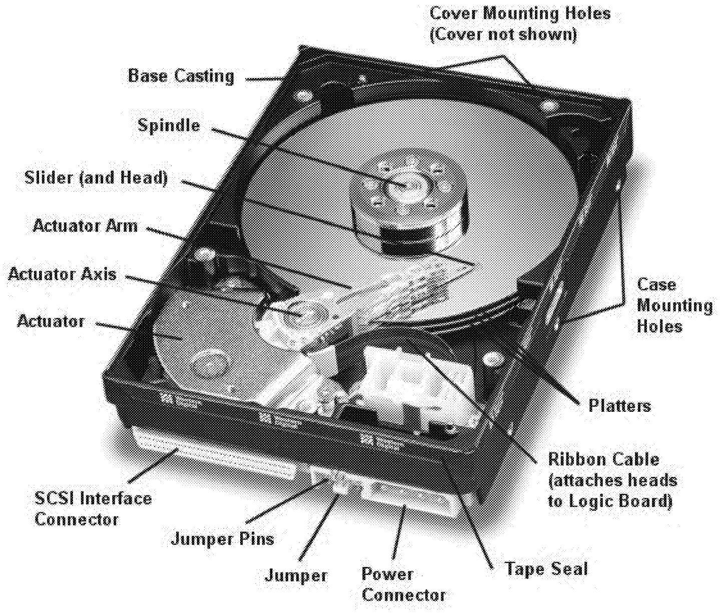
  ➔new system calls (in addition to the original interface)

```
read_coop(int fd, void *buf, size_t count,
          int time-out, int abort);
write_coop(int fd, void *buf, size_t count,
           int time-out, int abort);
open_coop(const char *pathname, int flags,
          int time-out, int abort);
```

  ➔the OS can decide when to serve these requests

Reference:[WBB02]

C Disk Energy Characterization



C.1 Device States of Hard Disks

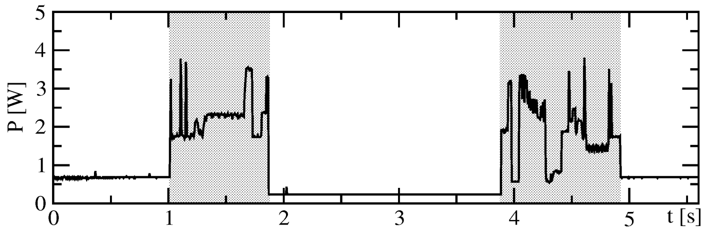
- Hard disks support several modes with low power consumption
- Drawback of low-power modes: access delays (20 ms – 10s)
- Typical modes of operation (IBM Travelstar)

Power Mode	Properties	Power consumption	Access delay
Active	read and write operations	2.1–4.7 W	—
Performance Idle	entered after I/O operation	1.85 W	—
Active Idle	heads in the middle, servo system off	0.85 W	20 ms
Low-Power Idle	heads on parking ramp	0.7 W	300 ms
Standby	spindle motor off	0.25 W	1.0–9.5 s
Sleep	(almost) all electronics off	0.1 W	3.0–9.5 s

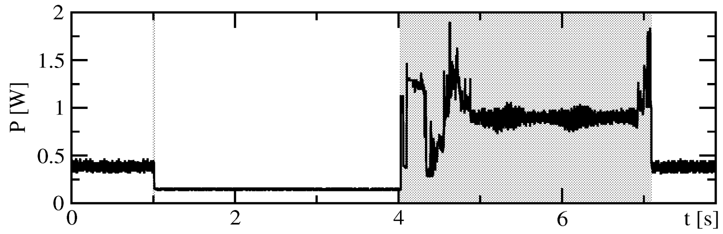
C.2 Mode Transitions

- Mode transitions consume time and energy:
  - ◆ Parking and positioning of heads
  - ◆ Spindle motor activation and slow down

IBM Travelstar (2.5" Notebook-HD)



IBM Microdrive (CF form factor HD)



C.3 Break-Even Time

- Break-even time is the time span where

  energy consumption in low-power mode + transition to low-power mode and back = energy consumption in idle mode

- Energy characteristics of a HD for transitions between idle and standby:

Transition	Delay	Energy Consumption
spin up	t_{su}	E_{su}
spin down	t_{sd}	E_{sd}

Power Mode	Time	Power
Idle	t_i	P_i
Standby	t_s	P_s

  - Definition of the *break-even time* t_{be}:

$$t_{be}P_i = (t_{be} - t_{sd} - t_{su})P_s + E_{sd} + E_{su}$$

$$t_{be} = \frac{E_{sd} + E_{su} - P(t_{sd} + t_{su})}{P_i - P_s}$$

  - Energy is saved by switching to standby-mode if $t_i > t_{be}$
  - IBM Travelstar: break-even time = 8.7 s (typical value 5s – 30s)

C.4 Side-Effects of Low-Power Modes

- Starting/stopping the spindle motor & parking of heads causes wear
  - Common reason for device failure
  - 50,000–300,000 mode transitions
  - Trade-off between energy savings, latency and “time to failure”
- Drives with multiple rotation speeds
  - Low rotation speed as an additional power-saving mode
  - Low rotation speed for operation at low temperatures
  - High rotation speed for maximum performance

D Hard Disk Power Management

- Spin-down policies
  - Offline policies
  - Time-out policies
  - Predictive policies
  - Stochastic policies
  - Notification mechanisms
- Cooperative I/O: *Deferrable* and *abortable* I/O requests

D.1 Spin-Down Policies

- Spin-down threshold:
  - Idle time without device access before a mode switch takes place
- Spin-down policy:
  - Policy to determine the spin-down threshold for each request
    - Offline
    - Online
- Policy for spin-down can be implemented on the level of
  - Drive firmware
  - Device driver
- Autonomous policies of the drive firmware can interfere with OS directed spin-down policies.
- Spin-down policies can interfere with buffer cache management and update policies.

1 Offline Policies

- Traces are the basis of offline spin-down decisions
  - ◆ Useful for embedded systems with statically scheduled activities
  - ◆ Useful for the evaluation of dynamic online policies
- "best fixed time-out"
  - ◆ Determination of a fixed spin-down threshold for all drive requests so that maximum power savings are achieved
  - ◆ Optimal non-adaptive policy
- "oracle algorithm"
  - ◆ Spin-down after finishing a request, if the idle time exceeds the break-even time
  - ◆ Spin-up right in time before the next request in order to minimize latency (t_{su})
  - ◆ Adaptive policy with minimal energy consumption and optimal performance

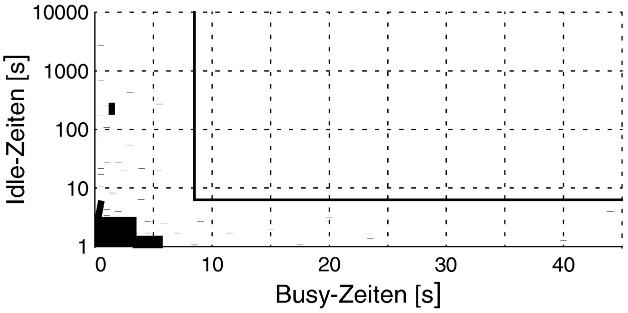
References:[DKM94, HLS96]

2 Fixed Time-Out Policy

- Simple and proven on-line spin-down policy
  - ◆ Switch to standby-mode is initiated by the firmware or the operating system.
  - ◆ Spin-down threshold can be configured (e.g., by the user/administrator. Is he competent to make a power/performance trade-off?)
- Typical values for the threshold are in the range of minutes in contemporary operating systems:
  - ➜Trade-off between high energy consumption in long idle-mode periods and the drawbacks of mode-switches (latency and wear)
- Break-even time as spin-down threshold:
  - ➜ $t_i < t_{be}$: energy consumption is equivalent to the oracle policy
  - ➜$t_i > t_{be}$: energy consumption is maximal twice the energy consumption of the oracle policy, because the energy consumption in idle-mode for t_{be} is the same as for a mode-switch.

3 Predictive Policies

- Prediction of the idle time in the future by analyzing the past
- Superior to fixed time-out policies:
  - ◆ no unneeded waiting in the idle mode if predicted idle-time > t_{be}
- L-shape policy
  - ◆ short busy periods follow long idle periods.
  - ◆ long busy periods follow short idle periods
  - ➔Spin-down after short bursts

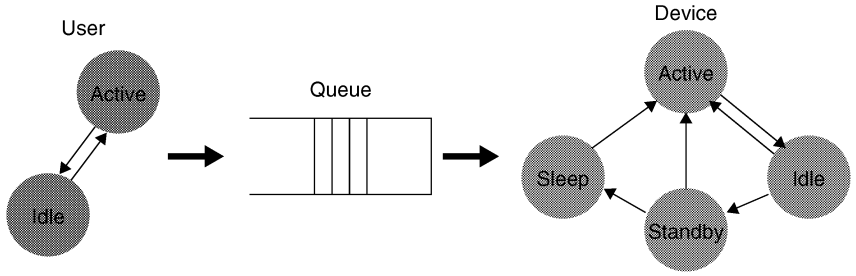


- Adaptive learning tree
  - ◆ Decision tree maps the sequence of idle-periods in the past on a prediction for the next idle period.
- Exponential average: $p[n+1] = a \cdot t_i[n] + (1-a) \cdot p[n]$ with $0 < a < 1$

References:[LM01, CBM99]

4 Stochastic Policies

- Use of stochastic processes for modelling of
  - ◆ Arrival times of requests
  - ◆ Time between two requests
  - ◆ Probability of a mode switch
  - ◆ Duration of a mode switch
  - ◆ Queue length
- Superior to predictive and timeout



References:[KMMO94, PBBDM98, KLV99, Sim02]

5 Notification Mechanisms

- New system calls for bi-directional application-OS notification
  - ◆ `RequireDevice(device, time, callback)`
  - ◆ `RequireDevice(device, tolerance, time, callback)`
  - ◆ `RequireDevice(device, type, period, wait)`

    ```
    type:  always / periodic / once / delete
    wait:  maximal delay
    ```

  References:[LBM02, LBM00]
- Compiler framework for automatic code transformation
  - ◆ `FILE *streamed_fd`, `FILE *non-streamed_fd`
  - ◆ I/O operations are mapped to a run-time library
    - Run-time disk drive characterization
    - Implementation of buffered I/O incl. management of buffers of variable size
    - Notification of the operating system about the following idle period
    - New system call: `next_R(time_t t)`

  Reference:[HPH+02]

D.2 Cooperative I/O

1 Principle of Operation

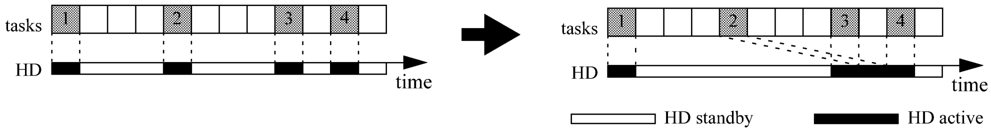
- *Deferrable* and *abortable* I/O requests

```
read_coop(int fd, void *buf, size_t count,
          int time-out, int abort);
write_coop(int fd, void *buf, size_t count,
           int time-out, int abort);
open_coop(const char *pathname, int flags,
          int time-out, int abort);
```

- Hard disk in active or idle mode:
  - ◆ deferrable operations are executed immediately
- Hard disk in standby mode:
  - ◆ operations are deferred until hard disk is activated by another process
  - ◆ *or* until user-defined time-out is reached
  - ◆ then: force activation of hard disk or cancel the operation

1 Principle of Coop I/O

➜ Disk operations are clustered/grouped together



➜generate long periods of inactivity

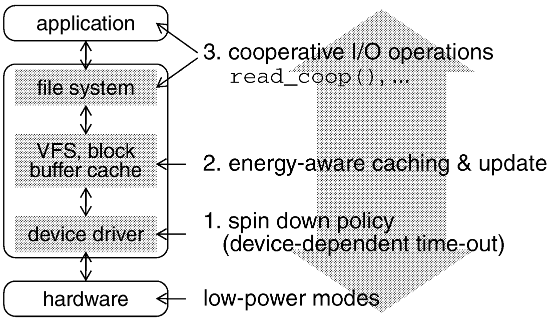
➜fewer mode transitions

➜hard disk can be kept longer in standby mode

- Examples:
  - ◆ audio-/video player
  - ◆ web browser
  - ◆ background processes
  - ◆ auto-save

1 Principle of Coop I/O

- Integration of all layers—from the hardware to the application



application

file system

VFS, block buffer cache

device driver

hardware

3. cooperative I/O operations `read_coop()`, ...

2. energy-aware caching & update

1. spin down policy (device-dependent time-out)

low-power modes

- Implementation
  - ◆ Linux Kernel 2.4.20
  - ◆ Modifications to the IDE device driver, VFS (block buffer cache and update mechanism) and ext2 file system

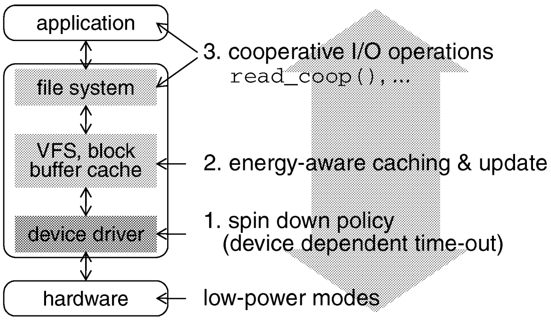
2 Coop I/O Interface

- New system call interface to file-I/O

```
read_coop(int fd, void *buf, size_t count,
          int time-out, int abort);
write_coop(int fd, void *buf, size_t count,
           int time-out, int abort);
open_coop(const char *pathname, int flags,
          int time-out, int abort);
```

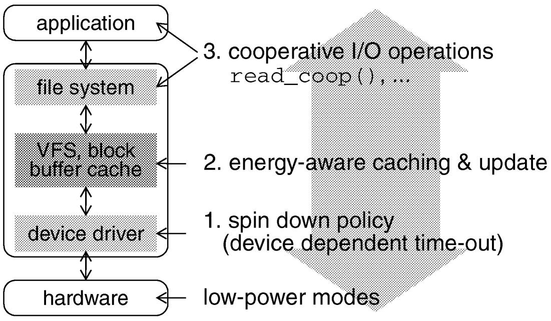
- Legacy calls can be mapped to coop-calls
  - `abort = FALSE;`
  - Default value for `time-out: 0`
  - Environment variable `COOP_TIME_OUT` defines time-out value for uncritical (background) tasks

3 Spin-Down Policy



1. Transition to standby mode if hard disk is idle
   - simple, efficient and proven algorithm:
     Device-Dependent Time-out [LM01]
   - spin down if:
     current time – time of last access > break-even time

4 Energy-aware Caching & Update



2. Energy-aware caching & update

   ➔goal: clustering of disk operations

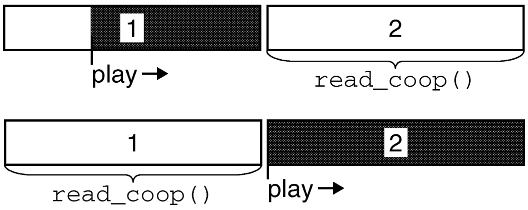
   ◆ update writes all “dirty” blocks to disk, independent of their age

   ◆ updates are attached to other disk accesses

   ◆ If device driver decides to switch to standby mode, force update before the mode transition

5 Measurements

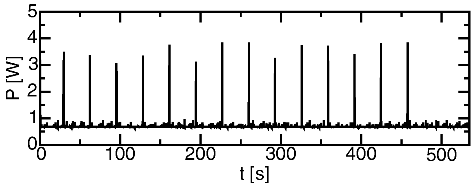
- DAQ system
  - ◆ measurement of voltage drop at defined resistors in the 5V supply line of the hard disk
  - ◆ resolution: 256 steps; 20000 samples per second
- MP3 player AMP using deferrable `read_coop()` requests
- Player with two read buffers
  - ◆ audio data is read from one buffer
  - ◆ thread fills other buffer with deferrable read calls.
  - ◆ modifications: ~ 150 lines



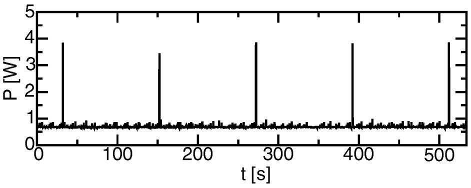
- Mail reader stores new mails (checking mail every minute) in a file on hard disk using legacy `write()`.

5 MP3-Player + Mail-Reader

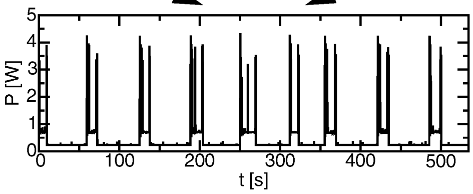
- Write requests of mail reader and read requests of AMP are grouped together



unmodified AMP using `read()` (373 J)



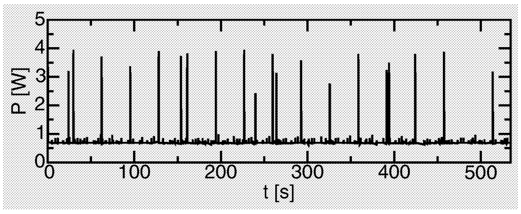
mail reader using `write()` (164 J)



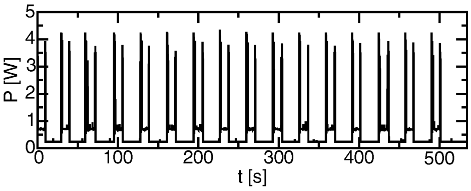
Cooperative-I/O kernel,
AMP using `read_coop()` + mail reader (210 J)

5 MP3-Player + Mail-Reader

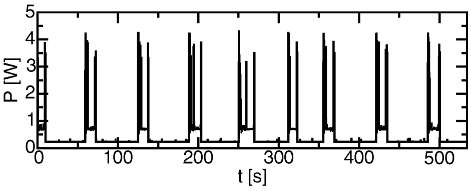
- Linux kernel w/o power management (373 J):



- Cooperative-I/O kernel, AMP using `read()` (265 J):

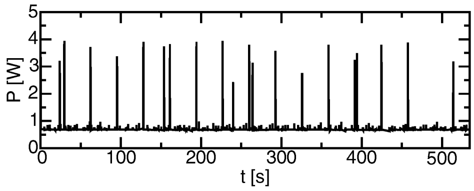


- Cooperative-I/O kernel, AMP using `read_coop()` (210 J):

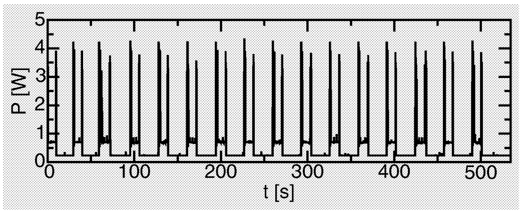


5 MP3-Player + Mail-Reader

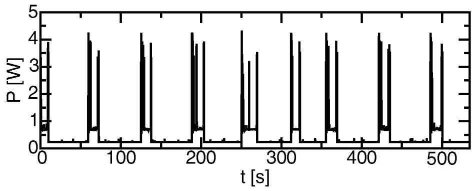
- Linux kernel w/o power management (373 J):



- Cooperative-I/O kernel, AMP using `read()` (265 J):

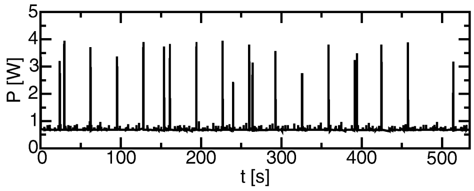


- Cooperative-I/O kernel, AMP using `read_coop()` (210 J):

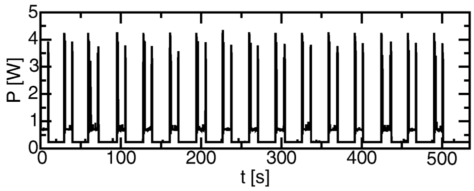


5 MP3-Player + Mail-Reader

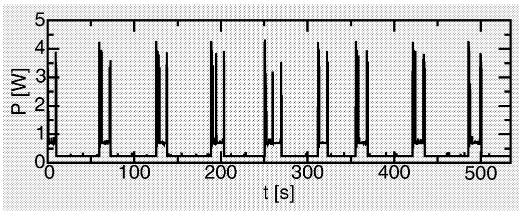
- Linux kernel w/o power management (373 J):



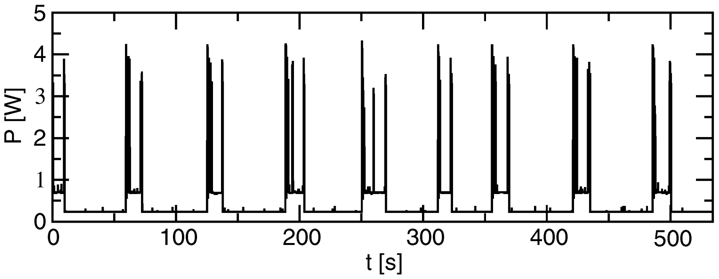
- Cooperative-I/O kernel, AMP using `read()` (265 J):



- Cooperative-I/O kernel, AMP using `read_coop()` (210 J):



5 MP3-Player + Mail-Reader



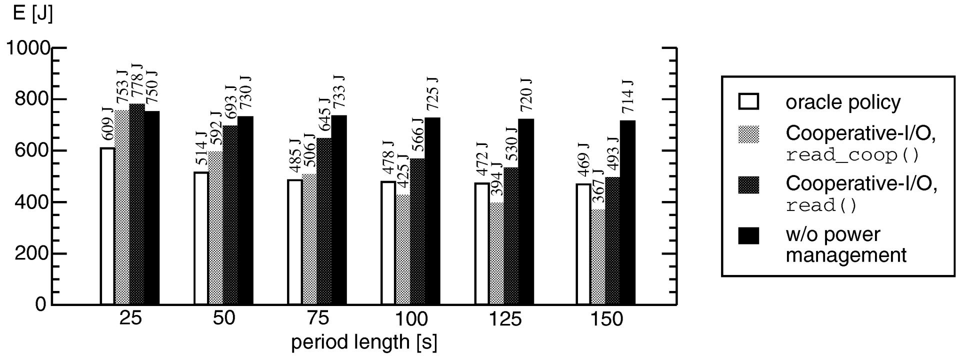
Cooperative-I/O, AMP using `read_coop()` (210 J)

- Two `read_coop()` requests to fill the two read buffers
- If device is in standby mode:
  - ◆ first request is deferred until active buffer is empty
  - ◆ force spin-up to serve request; device is in idle mode
  - ➔`read_coop()` request for second buffer is executed immediately
  - ➔effectively two read operations are grouped together
- Write requests are attached to read operations of AMP

5 Synthetic Tests

- Five processes wake up periodically and issue cooperative read or write requests after random wait times
- Comparison of four policies:
  - ◆ Linux kernel without power management
  - ◆ Cooperative-I/O kernel, test programs use `read()` or `write()`
  - ◆ Cooperative-I/O kernel, use of `read_coop()` or `write_coop()`
  - ◆ Oracle
- “Oracle” spin-down policy
  - ◆ knows timing of future disk operations (traces!)
  - ◆ transition to standby mode immediately if energy savings are possible
  - ➔optimal strategy with respect to energy consumption
  - ➔no influence on times of disk operations!

5 `read_coop()`

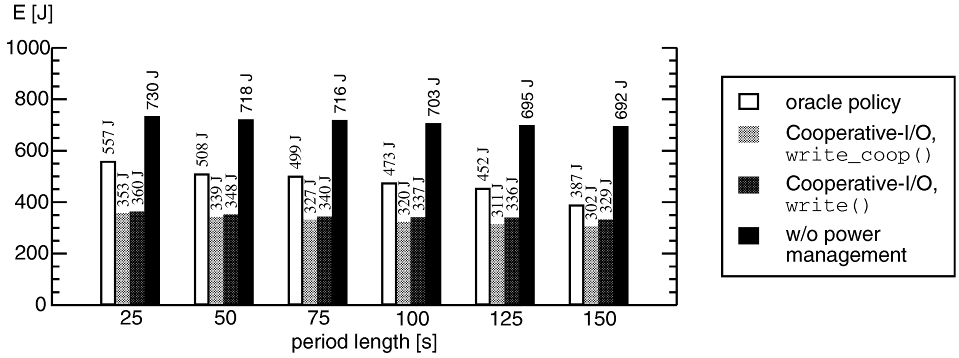


- Higher energy savings than (uncooperative) oracle policy
- Cooperative I/O clusters accesses
  ➔reduction of mode transitions
  ➔more time in standby mode

mode / strategy	active + transitions	idle	standby
cooperative	29 s	153 s	868 s
oracle	107 s	132 s	811 s

- Oracle policy does not defer or cluster requests!

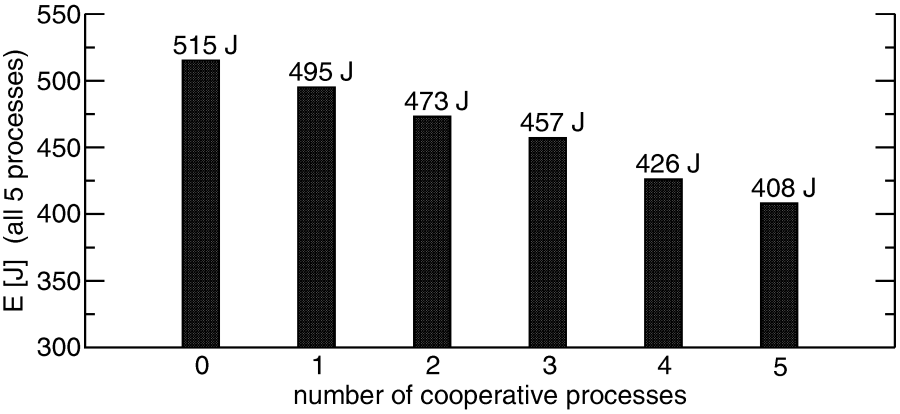
5 `write_coop()`



- `write()` and `write_coop()` requests are already deferred by the update mechanism
  ➔only little additional savings when using `write_coop()`
- Oracle has no influence on the times of disk operations
  ➔requests are not deferred (synchronous writes)

5 Varying number of cooperative processes

- 5 processes run in parallel,
  0–5 of them using `read_coop()`, the other `read()`



- The more processes using `read_coop()` instead of `read()`,
  the higher the energy savings

D.3 Conclusion

- Cooperative I/O achieves higher energy savings than oracle policy
  ➔higher energy savings than “traditional” spin-down policies

- Applicable to all devices with expensive state transitions
  ◆Rotating media
  ◆Micro-/Nanoelectromechanical Systems (MEMS)
  ◆Wireless networks

- Cooperation of energy-aware and legacy applications

- Cooperation of device power management and buffer management

References

[CBM99] Eui-Young Chung, Luca Benini, and Giovanni De Micheli. Dynamic power management using adaptive learning tree. In *Proceedings of the Internation Conference on Computer Aided Design ICCAD*, pages 274–279, 1999.

[DKM94] Fred Douglis, P. Krishnan, and Brian Marsh. Thwarting the power-hungry disk. In *USENIX Winter*, pages 292–306, 1994.

[HLS96] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. In *Mobile Computing and Networking*, pages 130–142, 1996.

[HPH+02] Taliver Heath, Eduardo Pinheiro, Jerry Hom, Ulrich Kremer, and Ricardo Bianchini. Application transformations for energy and performance-aware device management. In *Proceedings of the Eleventh International Conference on Parallel Architectures and Compilation Techniques PACT '02*, Sept 2002.

[KLV99] P. Krishnan, Philip Lon, and Jeffrey Scott Vitter. Adaptive disk spindown via optimal rent-to-buy in probabilistic environments. Number 23, pages 31–56, 1999.

[KMMO94] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, June 1994.

[LBM00] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Requester-aware power reduction. In *International Symposium on System Synthesis*, pages 18–23. Stanford University, September 2000.

[LBM02] Yung-Hsiang Lu, Luca Benini, and Giovanni De Micheli. Power-aware operating systems for interactive systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(2), April 2002.

[LM01] Yung-Hsiang Lu and Giovanni De Micheli. Comparing system-level power management policies. *IEEE Design & Test of Computers special issue on Dynamic Power Management of Electronic Systems*, pages 10–19, March/April 2001.

[PBBDM98] G. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *Proceedings of the 35th Design Automation Conference DAC'98*, 1998.

[Sim02] T. Simunic. *Power Aware Computing*, chapter Dynamic Management of Power Consumption; T. Simunic. Kluwer Academic Publishers, 2002.

[WBB02] Andreas Weissel, Bjoern Beutel, and Frank Bellosa. Cooperative I/O: A novel I/O semantics for energy-aware applications. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation OSDI'2002*, Dec 2002.

Event-Driven Energy Characterization

Frank Bellosa

Department of Computer Science 4 (Operating Systems)
University of Erlangen, Germany
bellosa@cs.fau.de

Event-Driven Energy Characterization

Frank Bellosa

Department of Computer Science 4 (Operating Systems)
University of Erlangen, Germany
bellosa@cs.fau.de

Contents

A Outline

- The Playground for Power Management
- The Case for Energy Characterization
- Event-Driven Energy Characterization
- Event-Driven Energy-Accounting
- Event-Driven Thermal Management
- Event-Driven Clock Scaling
- Conclusion

B The Playground for Power Management

B.1 Increasing the Lifetime of Batteries

- Improvement of the efficiency of energy consuming components
  - CPU: clock/voltage scaling
  - Energy-aware virtual memory & garbage collection
  - Energy-aware I/O (disk, WLAN)
  - TFT display power management
- Improvement of battery efficiency
  - Battery-aware task & I/O scheduling
  - Battery-aware dynamic clock/voltage scaling

B.2 Limiting Peak Power

- Peak power consumption may not exceed limits
  - ◆ Battery/UPS power
  - ◆ Solar power
  - ◆ Power over Ethernet
  - ➔How to guarantee mission critical services?

B.3 Thermal Management

- Temperature may not exceed limits ($T_{skin} < 55$ °C, $T_{chip} < 75$ °C)
  - ◆ Increased reliability of devices without fans
  - ◆ Safe operation in case of cooling failure
  - ➔Power estimation
  - ➔Power throttling policy
  - ➔Thermal response mechanism

C Energy Characterization

- Power Management has to decide
  - ◆ When
  - ◆ Where
  - ◆ How fast
  - ◆ Under which environmental conditions (power & cooling)

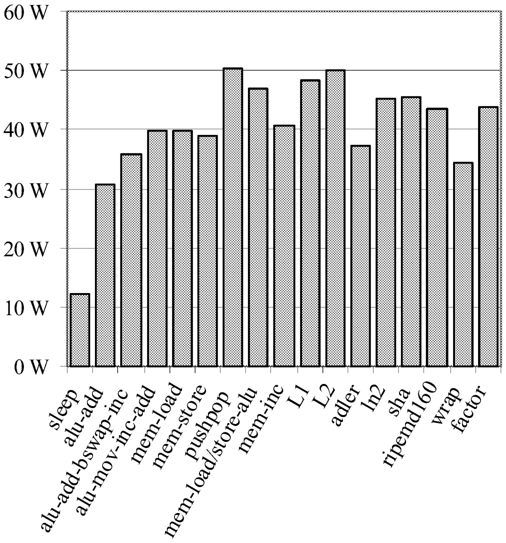
  any activity in the system is allowed to take place.
- The energy-related behavior of the system is the basis of all power-management decisions.
- If we cannot predict the behavior of the system off-line, we have to observe the system.

➔ The Case for On-Line Energy Characterization

C.1 Energy Characteristics

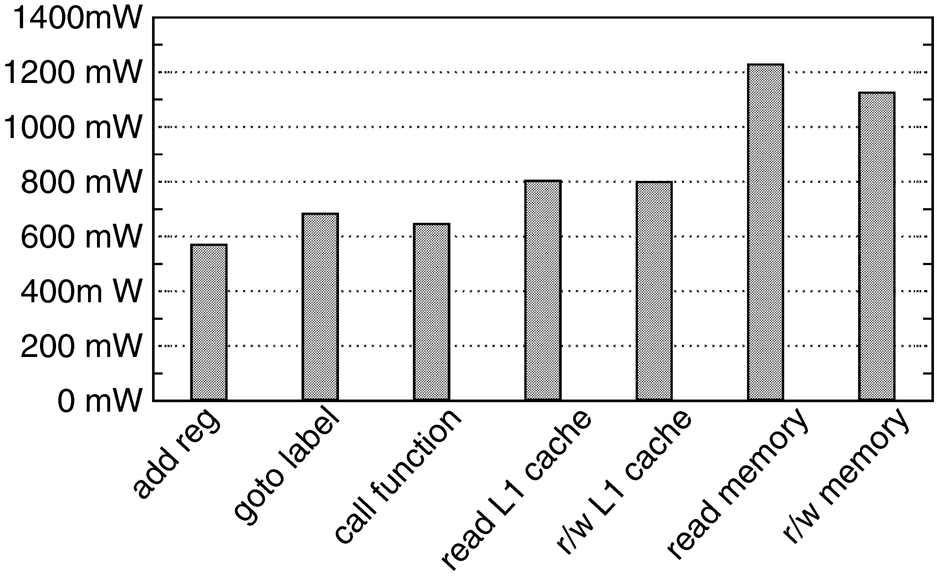
1 CPU Power: Application Dependency

- Pentium 4@2GHz



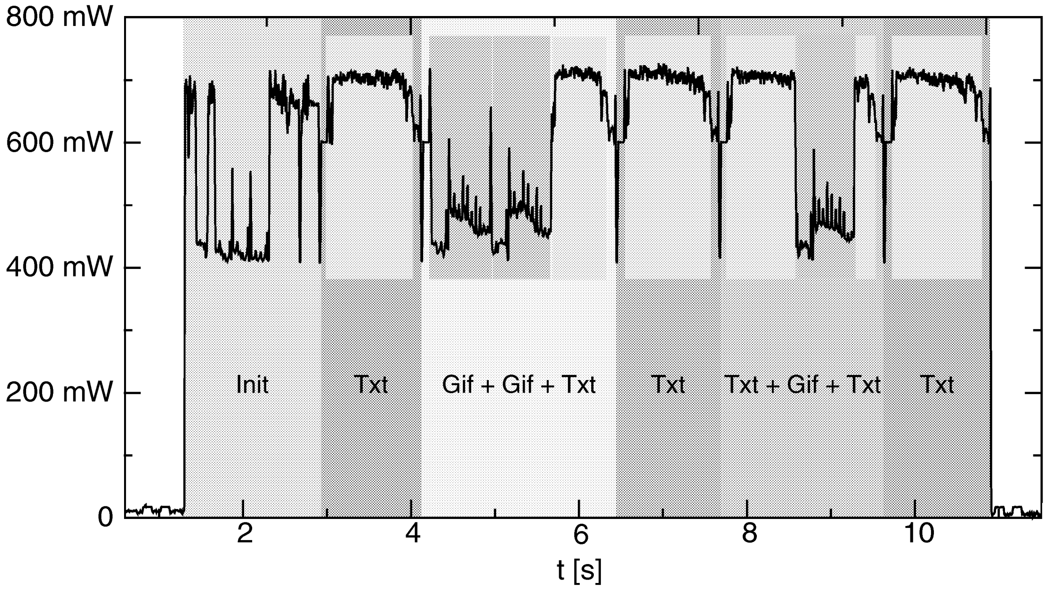
2 System Power: Application Dependency

- Intel IQ80310 (XScale@733 MHz)



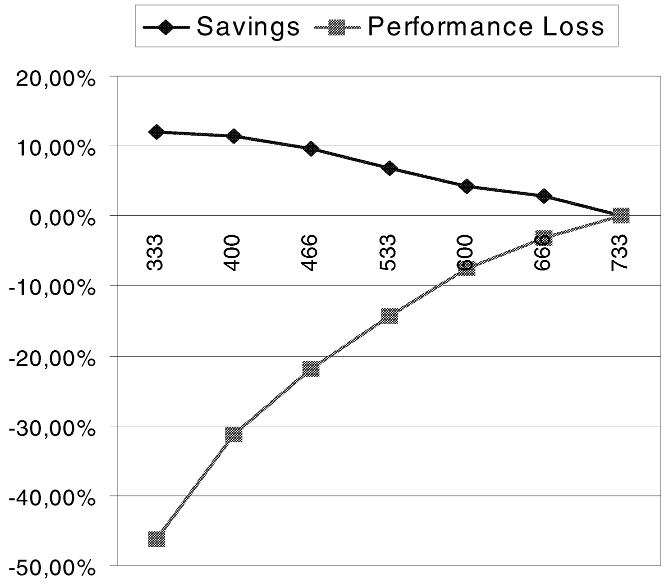
3 System Power: Data Dependency

- Ghostscript running on Intel IQ80310 (XScale@733 MHz)



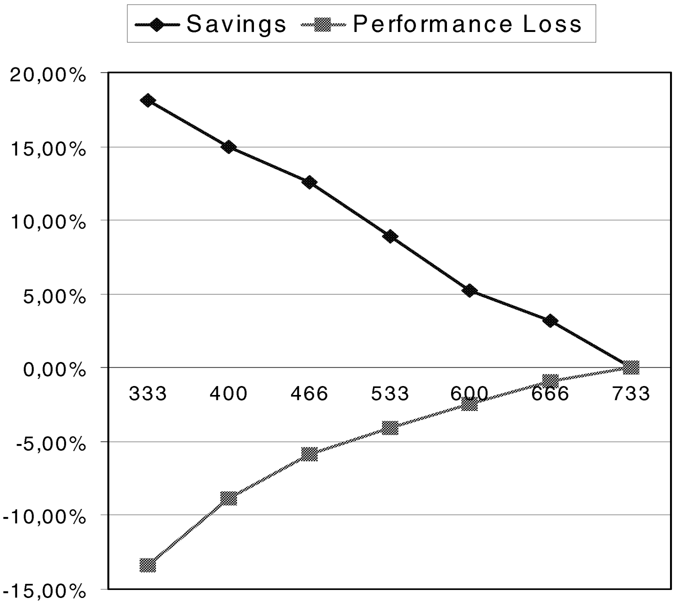
4 Clock Scaling: Energy vs. Performance Profile

- Energy savings vs. performance loss for “`factor`”



4 Clock Scaling: Energy vs. Performance Profile II

- Energy savings vs. performance loss for “grep”

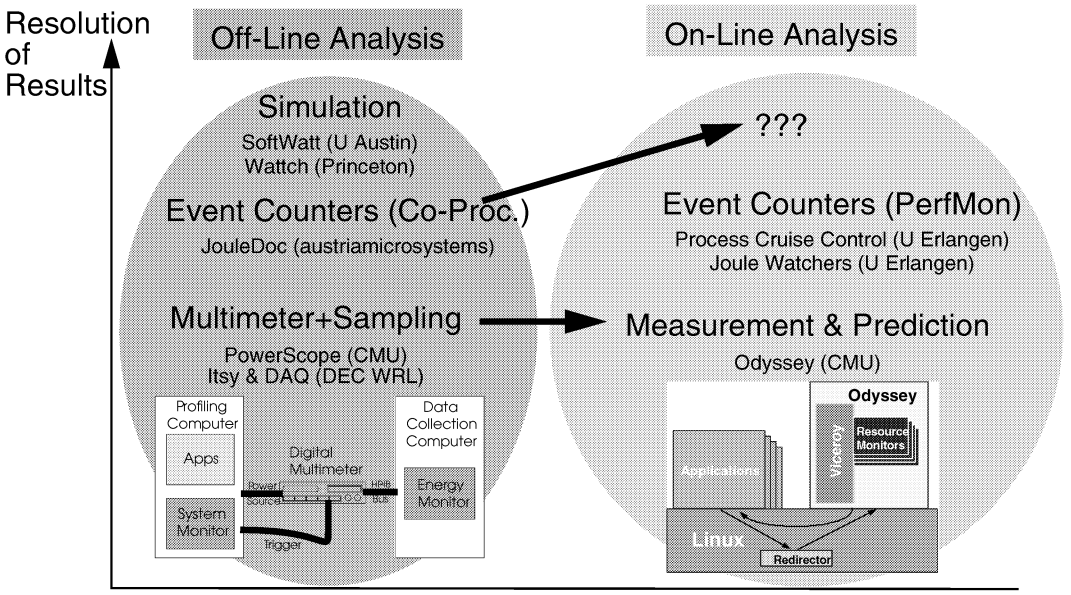


5 Energy Characteristics of Applications

- Simple applications show predictable energy consumption
  - ◆ Time is an indicator for energy consumption (for a given clock speed)
  - ◆ Example: code with fixed loop counts and branches of equal complexity
- Complex applications show unpredictable characteristics due to
  - ◆ Multiple blocks of computations with unknown execution order
  - ◆ Unknown input data
  - ◆ Unknown working set size
  - ◆ Unknown scheduling sequence in a multitasking environment
    - Cache effects (compulsory misses)
    - TLB (TLB misses)
  - ◆ Examples: interpreters, decoders, virtual machines, dynamic scheduling

➜ The case for high-resolution energy estimation

C.2 Energy Estimation



Resolution of Results

Off-Line Analysis

On-Line Analysis

Simulation
SoftWatt (U Austin)
Wattch (Princeton)

Event Counters (Co-Proc.)
JouleDoc (austriamicrosystems)

Multimeter+Sampling
PowerScope (CMU)
Itsy & DAQ (DEC WRL)

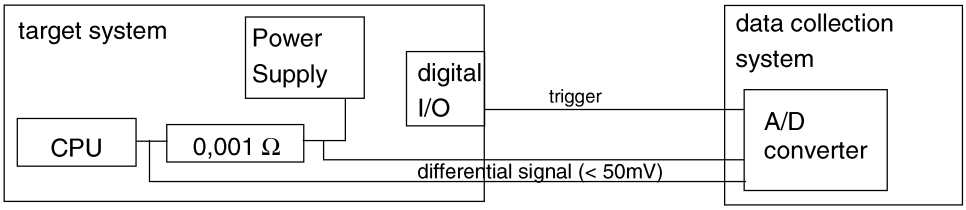
???

Event Counters (PerfMon)
Process Cruise Control (U Erlangen)
Joule Watchers (U Erlangen)

Measurement & Prediction
Odyssey (CMU)

1 Power Measurements

- Measurement methodology:



- Sampling frequency must be higher than frequency of power changes
  ➔Sampling must be faster than execution of code-sequence of interest
- Trigger must be sent with low overhead
  - ◆ Overhead of digital I/O driver (Parallel I/O: P4@2GHz 2400 cycles = 1,2µs)
  - ◆ Overhead of system call
  - ◆ Overhead due to additional cache misses
- Not applicable for SoC designs
- High-speed, high-resolution measurement environment
  ➔Feedback-driven PM of field targets is to expensive (HW, energy)

2 Simulation

- Level of simulation
  - Layout
  - Functional block
  - Instruction
- Coverage of simulation
  - Processor core
  - Processor, caches, MMU, executing user-level code
  - Complete CPU executing UL&KL code
  - Complete machine simulation (incl. memory, I/O)
- Simulation speed: 4000-10.000.000 times slower than target system
- No timing accurate simulation model available for many HW-components
- Feedback-driven power management not applicable

  References: [BTM00, SMH01, GSI+02]

D Event-Driven Energy Estimation

- Performance penalties are the result of HW latencies.

  ➜Let’s count performance critical HW activations!

  ➜ Performance monitoring counters

  Reference:[ABD+97]
- Energy consumption is the result of HW activity.

  ➜Let’s count energy intensive HW activations!

  ➜ Energy monitoring counters

  Reference:[Bel00]

D.1 Event Selection

- Challenge:
  - Selection of energy-critical events
    - Clock cycles
    - Retired (micro-) instructions
    - Mispredicted branches
    - Address translations
    - Cache operations (read/write, miss, write-back)
    - Bus transactions
    - Memory bank activations
  - Assignment of energy values to events

$$\text{Energy} = \sum_{\text{event\_types}} \#\text{events}_{\text{type}} \cdot \text{energy\_value}$$

  - Energy-critical events and energy values can be derived from low-level architectural models.

Reference:[GBCH01]

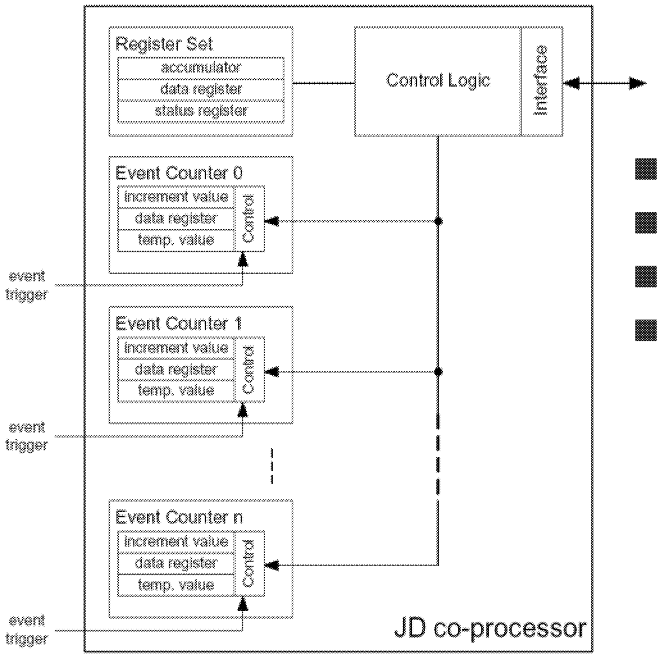
D.2 Counter Implementation

1 Event-Monitoring Counters

- Counters register energy-critical events in the complete system architecture.
- Counters should be read/written with low overhead (time/energy)
  - ➔Counters are implemented as special registers (not memory mapped)
  - ➔Counters are part of the core
    - Counters are faraway of the event location (event-signal propagation!)
- Preciseness is not required.
  (e.g., events of previous instructions must not yet be registered)
    - Event propagation delay is not critical.
- Misuse of performance monitoring counters
  - Incomplete coverage of energy-critical events (# counters, types of events)
  - High latency for reading (e.g., 40ns)

References: [Bel01, JM01, Kel03]

2 Event-Monitoring Co-Processor



- Dedicated co-processor
- Low OS-overhead
- Minimal side-effects
- Integrated in SoC design

Reference:[HKS+03]

3 Benefits

- Power/Performance characteristics become a first class element of the task context.
- Low memory overhead:
  Just one entry in the task structure per event
- Low algorithmic overhead:
  Counters are evaluated
  - ◆ in the timer interrupt handler
  - ◆ in the scheduler
- High temporal resolution:
  Characteristics can be measured for arbitrarily short periods.
- Fast response:
  Changes in the energy-related characteristics of a process can be registered promptly.

D.3 Event-Driven Energy Estimation

1 Methodology for the Determination of Energy Values

- Energy measurements of m training applications with a DAQ system:

  $\bar{e}^T = (e_1, e_2, \ldots, e_m)$

- Reading of the number of events for n types of events for m applications:

  $A = [a_{i,j}] \quad (1 \le i \le m, 1 \le j \le n)$

- Find a vector $\bar{x}^T = (x_1, x_2, \ldots, x_n)$
  with $|A \cdot \bar{x} - \bar{e}|$ minimal
  and $A \cdot \bar{x} - \bar{e} \ge 0$ so that an underestimation of the energy will not be accepted.

2 Specifics of the Pentium 4 Target

- Performance monitoring counters register energy-critical events.
- For complex floating point instructions, MMX, SSE, and SSE2 operations our quest for a set of events fails because of a lack of meaningful events.
- First- and second-level cache misses cannot be counted simultaneously. Most applications show a low 2nd-level cache miss rate

  ➜Negligence of the power contribution of 2nd-level cache misses

  ➜Underestimation of energy consumption of up to 20%

event	weight [nJ]	max. rate (events per cycle)	power contribution [Watt]
time stamp counter	6.17	1.0000	12.33 W
unhalted cycles	7.12	1.0000	14.23 W
uop queue writes	4.75	2.8430	26.99 W
retired branches	0.56	0.4738	0.53 W
mispred branches	340.46	0.0024	1.62 W
mem retired	1.73	1.1083	3.84 W
ld miss 1L retired	13.55	0.2548	6.91 W

3 Accuracy of Event-Driven Energy Estimation

application	estimation error of energy consumption
OpenOffice 1.0.2	-1.69%
Mozilla 1.0.0	-0.56%
Linux 2.5.64 kernel-build	4.16%
jvm98 1.03	2.20%
caffeine 2.5	6.09%
perl	4.95%

D.4 Energy Accounting

1 General Resource Accounting

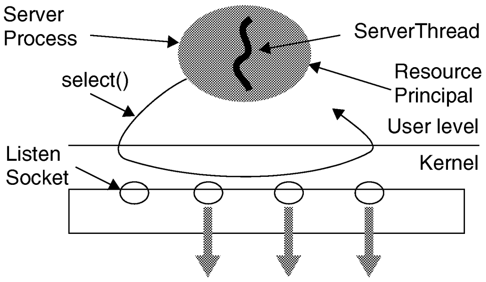
- Accounting: Measurement of resource consumption
  - ◆ Resources: CPU time, allocated memory, disk space, disk operations, network transfer volume, **energy**
- Accounting is a requirement for resource management (e.g., resource limitation, resource guarantees, resource billing)
- Accounting of a resource has to cover all locations of consumption
  - ◆ CPU-time is spent on user-level and kernel-level
  - ◆ Memory is allocated for applications and kernel data structures

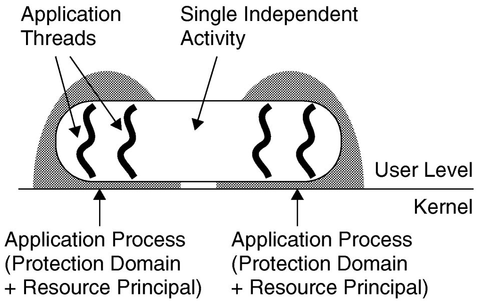
Application	% CPU (user mode)	% CPU (kernel mode)
ftp (transfer with 92 Mbps)	10	90
thttpd webserver (400 requests/s)	16	84
Realplayer (30 frames/s)	32	68

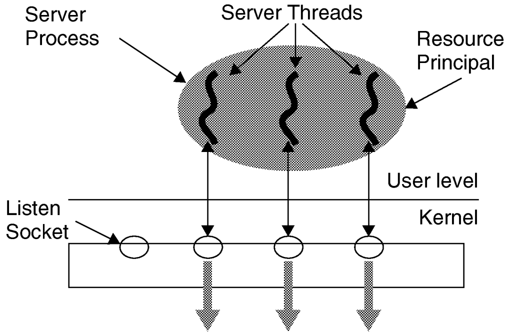
- Resources have to be accounted exactly in a client-server environment

2 Client-Server Accounting

- Event-driven server:
  - ◆ A single process/thread accomplishes different tasks
- Multi-threaded server:
  - ◆ Several threads accomplish different tasks.
- Multitasking server:
  - ◆ Multiple processes cooperate to accomplish a single task




Application Threads
Single Independent Activity
User Level
Kernel
Application Process (Protection Domain + Resource Principal)
Application Process (Protection Domain + Resource Principal)



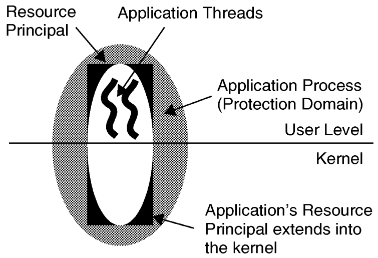
2 Accounting in a Server Environment

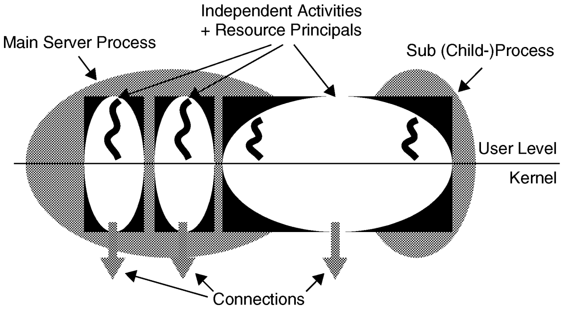
- Resources used by a server on user- and kernel-level should be accounted for
  - fulfilling specific tasks
  - serving requests of specific (classes of) clients
- The execution/protection domain is the wrong abstraction for resource accounting

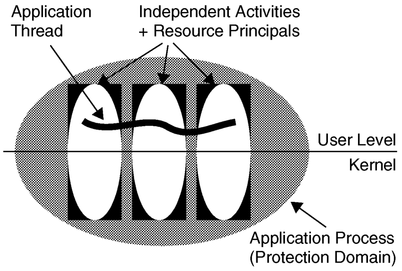
  ➔*Resource Containers* as a novel OS abstraction
  - Separation of protections domains and resource accounting
  - Accounting of any resource consumption to a “resource principal”

3 Resource Containers

- Accounting of CPU time in user-space and kernel mode
- Accounting of kernel objects (sockets, network buffers)
- Differentiation of network connections with different clients
- Use of resource accounting information in the scheduler


Resource Principal
Application Threads
Application Process (Protection Domain)
User Level
Kernel
Application's Resource Principal extends into the kernel

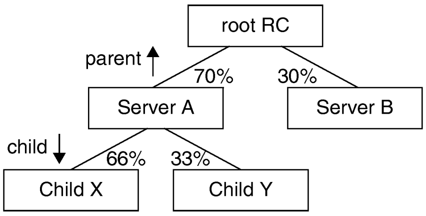



Application Thread
Independent Activities + Resource Principals
User Level
Kernel
Application Process (Protection Domain)

3 Resource Containers

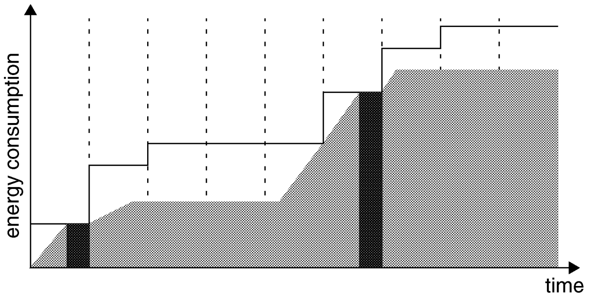
- Implementation
  - ◆ RCs are referenced via file descriptors.
  - ◆ Attributes of a RCs: resource consumption, scheduling parameters, resource limits (e.g., min. x, min. y%, max z%), access rights
  - ◆ *Scheduling Binding*:
    - A single thread can be bound to several RCs (thread structure holds references to RCs).
    - Scheduling decisions depend on resources of all bound RCs.
  - ◆ Dynamic binding of a thread to RCs
    - Explicit binding via system call interface
    - Implicit binding depending on descriptor binding (e.g., pipe, socket)
  - ◆ Resource Container Hierarchy
    - Resource assignment of a child depends on the remaining resources of the parents.

Reference:[BDM99]



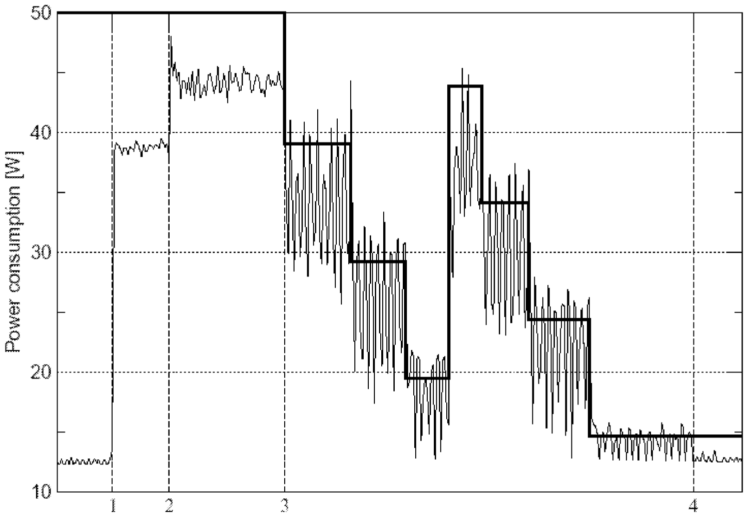
4 Energy Containers

- Resource containers for managing energy as a first class resource
- Energy is accounted with event-driven energy estimation.
- Energy containers provide the information for power management policies.
- Example: Limitation of average power consumption (e.g., solar power)
  - ◆ Periodic refreshing of energy limits (e.g., of the root container)
  - ◆ Only child containers consuming energy are refreshed.
  - ◆ Threads exceeding the limits of their associated containers are blocked,



4 Energy Containers

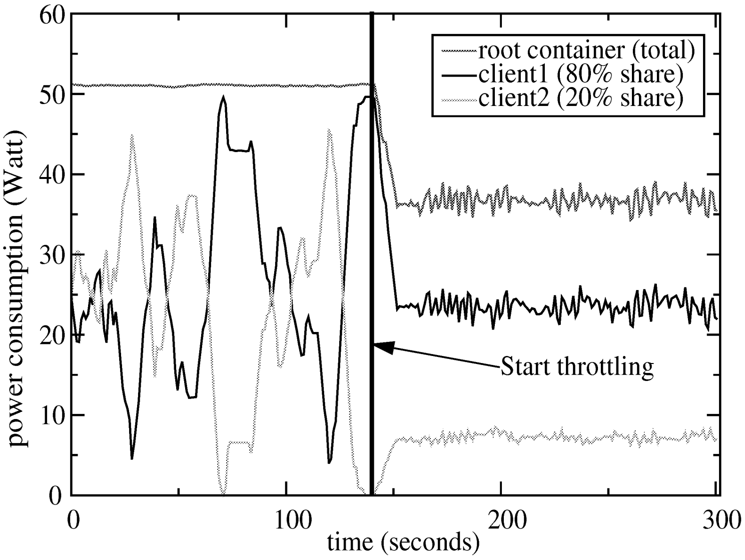
- Case study:
  Pentium 4@2GHz throttled to 40W, 30W, 20W, 45W, 35W, 25W and 15W



Reference:[Wai03]

4 Energy Containers

- Case study:
  Throttling two server applications with different client shares



Reference:[Kel03]

5 Energy Inversion Problem

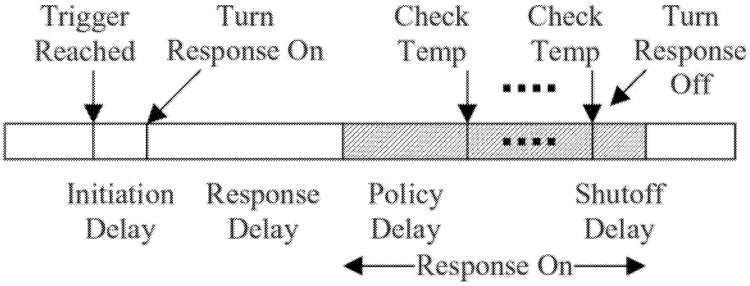
- Client-server environment
  - ◆ A clients requests a service from a server.
  - ◆ The server uses the client’s resource binding (e.g., implicit binding to the client’s energy container by reading from a socket descriptor).
  - ◆ The server exceeds the client’s energy limit while holding a critical resource:
    - ➜ The server is blocked holding another resource until energy is refreshed
    - ➜ Server cannot work on client requests with an energy budget.
    - ➜ “Energy Inversion” problem

E OS Directed Dynamic Thermal Management

E.1 Motivation

- Reduce costs for cooling
- Increased reliability of devices without fans
- Safe operation in case of cooling failure

E.2 Principle of Dynamic Thermal Management (DTM)



Reference:[BM01]

E.3 Trigger Mechanisms

1 Temperature sensors

- A single sensor approximates the average chip temperature if placed far away from hot spots.
- Multiple sensors are required to cover all hot spots.
- There is a delay between temperature reading and actual temperature. (maximum thermal ramp rate of the P4 processor: 50° C/s!) [Int02]
- High overhead for reading a thermal diode (e.g., reading the P4 thermal diode via SMBus can take 8.5 ms)
- Temperature reading can hardly be correlated with active tasks to identify an energy/temperature principal.

2 Basics of Temperature Estimation

- Temperature is estimated by exploiting energy estimation information
  - ➔The accuracy of the temperature estimation depends on the accuracy of the energy estimation and the thermal model.
- The thermal model has to be calibrated for the specific combination of chip, chip case, interface material, heat spreader, and heat sink
- Assignment of an energy/temperature principal is possible.
  - ◆ The reliability of the thermal trigger depends on the reliability of the software (sufficient to indicate catastrophic thermal failure?)

3 Application-Specific Temperature Estimation

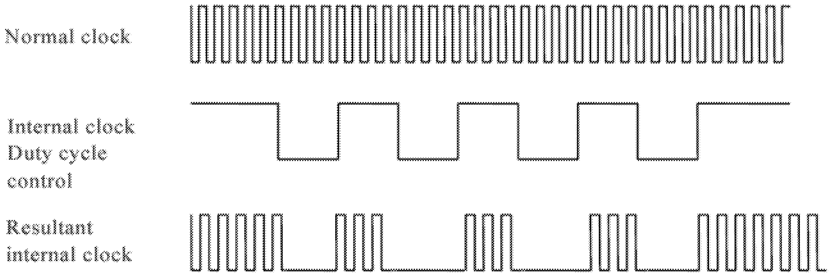
- Application-specific energy characteristics and time-based accounting
  - ◆ Only applicable for simple applications with predictable characteristics
  - ◆ Characteristics are determined with
    - application measurements
    - compile-time analysis
    - simulation
    - ➔ Characterization for each application for each hardware configuration
  - ◆ Hardware support is not required.
  - ◆ Overhead for energy estimation is marginal.

4 Event-Based Temperature Estimation

- On-chip activity counters
  - ◆ Counters can (theoretically) cover all hot spots
    - ➜ How many counters are sufficient?
  - ◆ Hysteresis depends on the scheduling architecture of the OS.
    - ➜ Energy and temperature can be determined for arbitrarily short periods.
  - ◆ Activity counters evaluated by software
    - Flexible accounting procedures
    - Software support for all micro architectural features required
  - ◆ Activity counters evaluated by hardware (co-processor)
    - Marginal overhead
    - Unified OS interface for all HW-components could be possible
    - Co-processor could also initiate thermal emergency procedures

E.4 Response Mechanisms

1 Dynamic clock modulation



◆ Hardware support required (internally or with chip-set support for asserting STPCLK#)

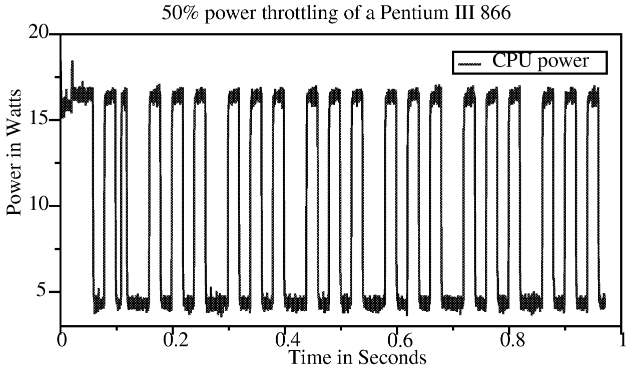
◆ Pentium 4: stop cycles are in the range of < 3 µs.

◆ Pentium 4: response delay: 1 µs

Reference: [Int02]

2 HLT Cycles

- Halting the CPU with the HLT instruction up to the next interrupt



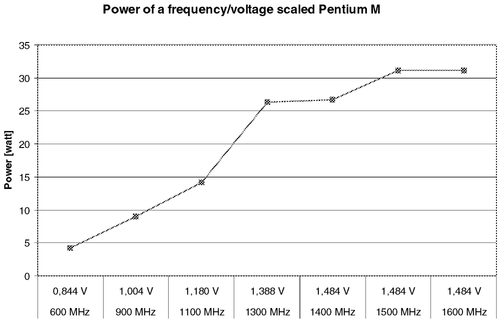
  - ◆ Short response delay, HLT instruction is part of the scheduler code
  - ◆ Maximum HLT period = timer interrupt period (typically 1-10 ms)
  - ◆ Precise power throttling is impossible without feedback from energy accounting.
  - ◆ Coarse grained throttling compared to dynamic clock modulation

3 Dynamic Frequency Scaling

- Changing the clock frequency of the processor
  - ◆ Increased CPU energy efficiency for memory intensive applications because of reduced number of memory stall cycles
  - ◆ Short response delay (e.g. 3.1 µs for an Intel XScale 80200)
  - ◆ Optional combination with voltage scaling (increased response delay due to voltage adjustment):

$$E \propto N_{OPS} \cdot C \cdot V^2$$

$$f_{max} \propto \frac{(V - c)}{V}$$



Reference: [Int00, Int03]

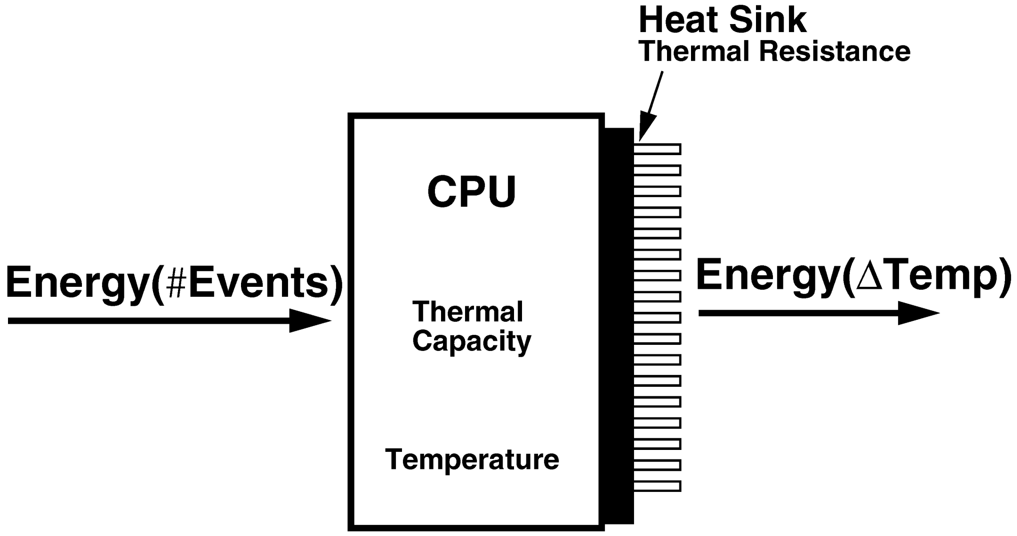
4 Instruction Decode Throttling

- ◆ Throttling the forwarding of instructions from the L1-I-cache to the instruction buffer.
- ◆ Multiple throttling levels
  Throttling level is set by writing a value in a control register.
- ◆ Core, caches and memory interface are clocked at a constant frequency
  - ➜ short response delay (a few cycles)
  - ➜ easy to implement in hardware
  - ➜ no opportunity for voltage scaling

Reference:[SKO+97]

E.5 Event-Driven Temperature Control

- ■ Thermal model:



1 Thermal Model

- The heat sink’s energy input consists only of the energy consumed by the processor, and can be formulated as

$$cm\Delta T = \Delta Q = \int_{t_1}^{t_1+\Delta t} P(t)dt$$

c :	constant
m :	mass of heat sink
ΔT :	heat sink's temperature increase
ΔQ :	difference of inner energy
P :	CPU power consumption
Δt :	elapsed time

which is transformed into

$$dT = \frac{1}{cm}Pdt = c_1Pdt$$

- The energy output of the heat sink is primarily due to convection and can be formulated as

$$\Delta Q = \frac{\alpha}{r} \cdot (T - T_0) \cdot t = cm\Delta T$$

r :	thermal resistance
α :	constant
T_0 :	ambient temperature

which is transformed into

$$dT = -c_2(T - T_0)dt \quad .$$

1 Thermal Model

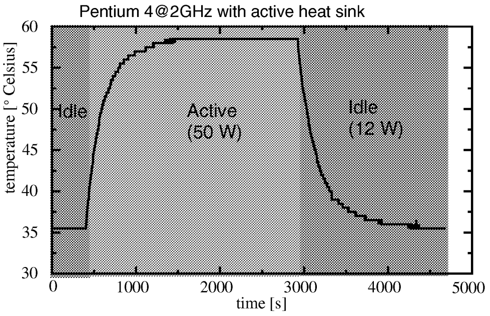
- The heat sink’s energy output by heat radiation does not have to be considered because the temperature is quit low (<60º celsius) and the aluminium surface has a low radiation emitting factor.
- The two formulas are used as an approach to estimate the processor temperature:

$$dT = [c_1P - c_2(T - T_0)]dt$$

Solving this differential equation yields [Kel03]

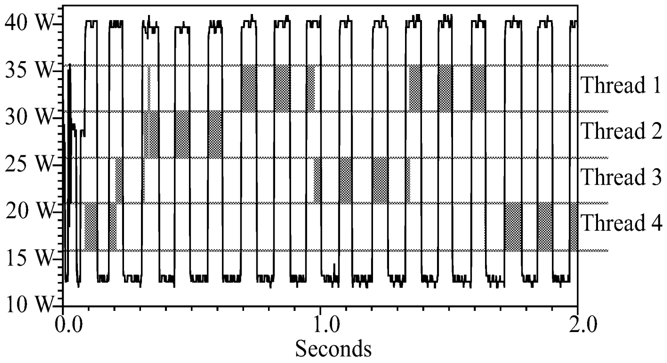
$$T(t) = \frac{-\tilde{c}}{c_2} \cdot e^{-c_2t} + \frac{c_1}{c_2} \cdot P + T_0$$

- The values for c_1 , c_2 and T_0 are found with measurements of the processor temperature on a sudden constant power consumption and a sudden power reduction to HLT power.



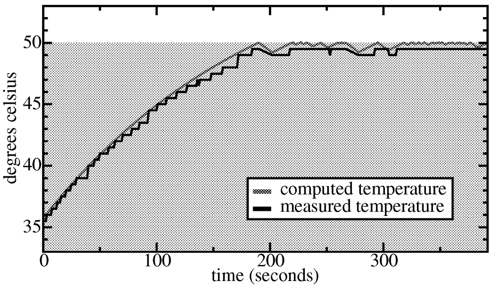
2 Power Throttling using Energy Containers

- Estimation of temperature and target energy consumption in epochs
- Limitation of power consumption
  - ◆ Periodic update of energy limits (e.g, 128 ms)
  - ◆ Refreshing of containers consuming energy in the last epoch
  - ◆ Throttling/halting the CPU, when energy exceeds threshold
  - ◆ Example: 4 compute intensive threads throttled to 50% peak power.

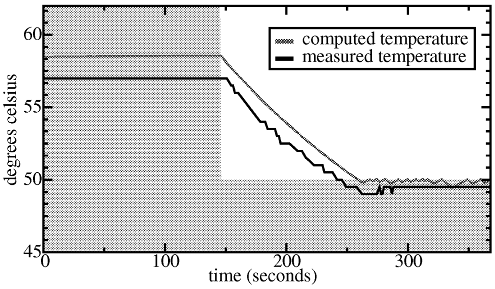


2 Event-Driven Temperature Control

- ◆ Example: Throttling when reaching a limit at 50º



- ◆ Example: Throttling and cool-down when setting a limit at 50º



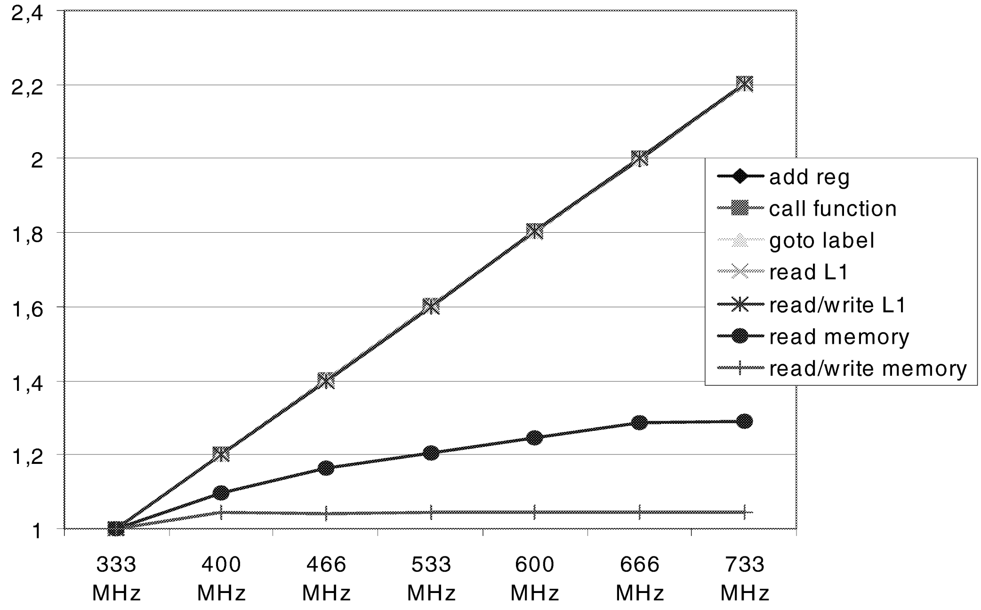
F Clock Scaling

F.1 Power/Performance Trade-Offs: Open Questions

- What kind of savings are possible for a specific thread?
- What is the expected loss in performance?
- How can the power/performance characteristics of an application be determined without
  - ◆ on-line power measurements?
  - ◆ hints from the application?
- How fast can the clock frequency/voltage be adjusted to the changing characteristics of an application?
- How much overhead is acceptable?

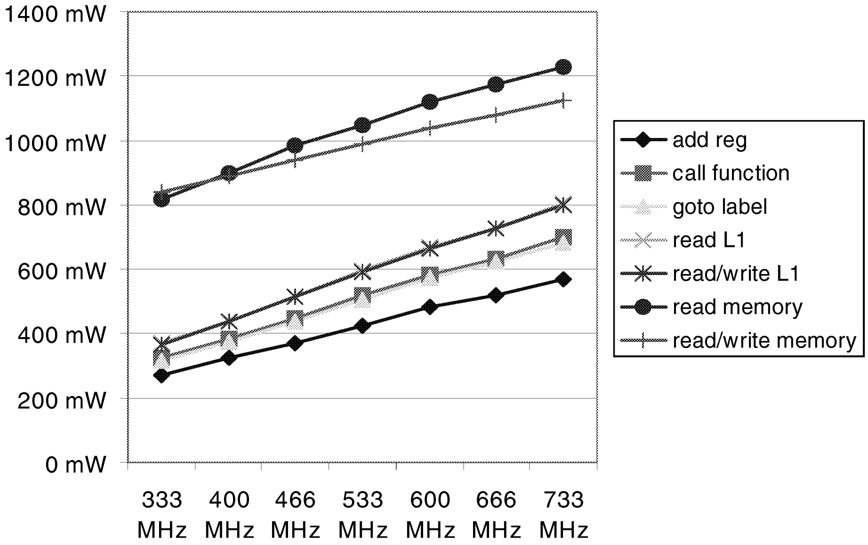
1 Performance Characterization

- Performance vs. clock frequency



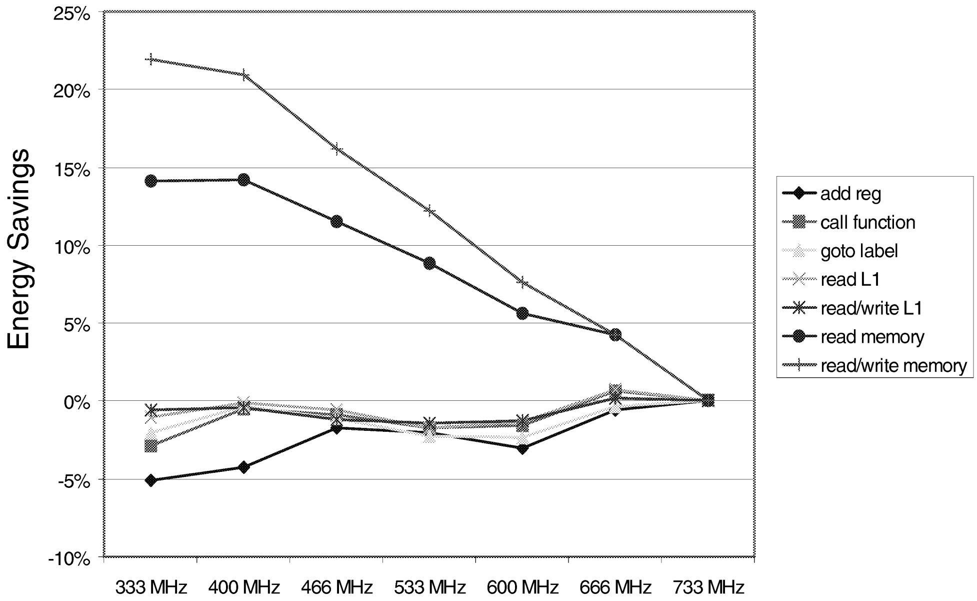
2 Power Characterization

- Power consumption vs. clock frequency:



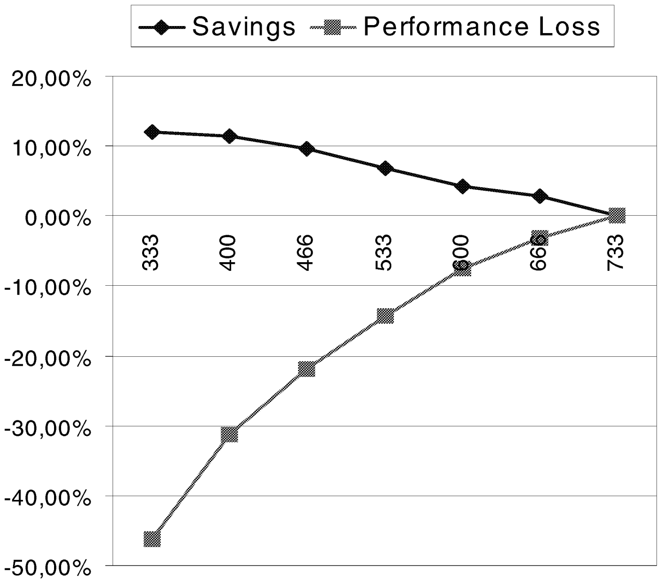
3 Efficiency Characterization

- Energy savings vs. clock frequency



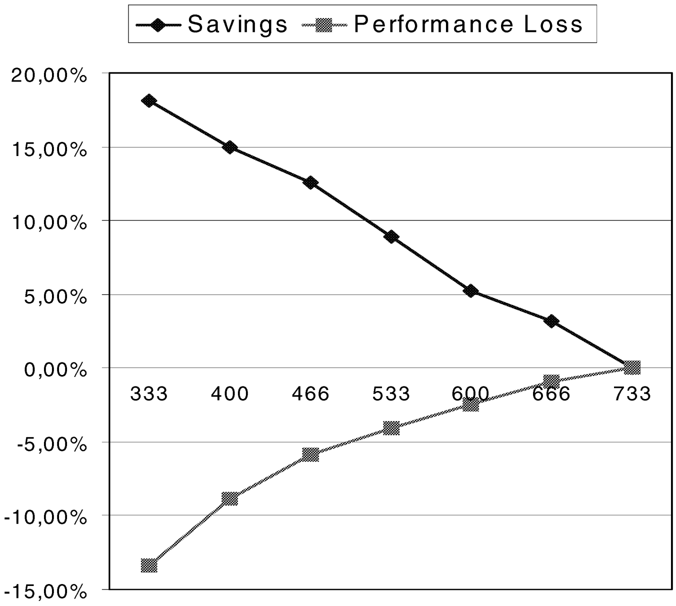
4 Energy Efficiency vs. Performance

- Energy savings vs. performance loss for "`factor`"



4 Clock Scaling: Energy Efficiency vs. Performance (2)

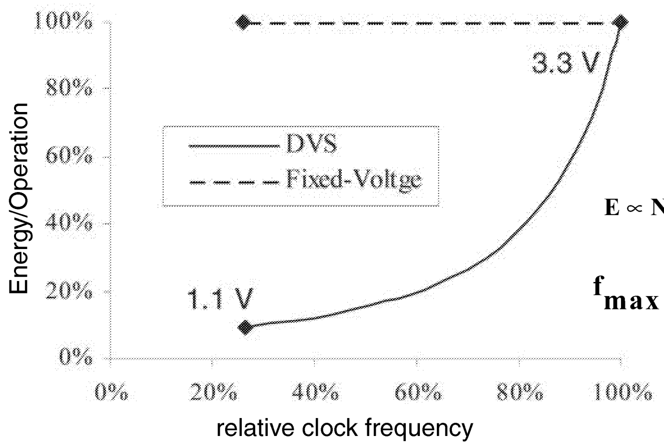
- Energy savings vs. performance loss for "`grep`"



5 Voltage Scaling

- Clock frequency – voltage – energy efficiency (Berkeley lpARM processor):

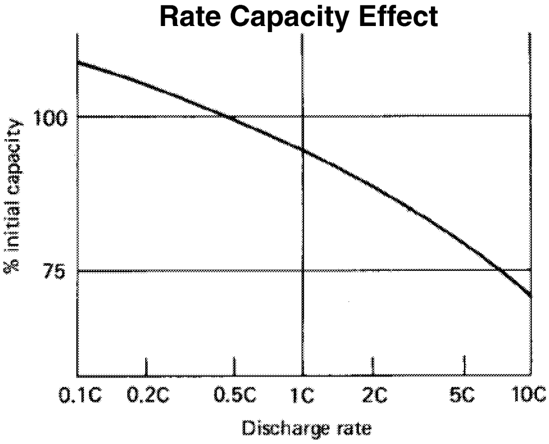
Reference:[PBB00]

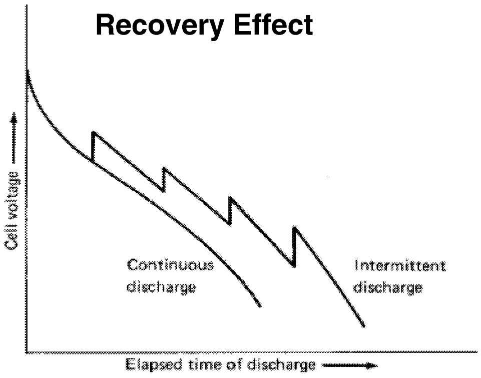


$$E \propto N_{OPS} \cdot C \cdot V^2$$

$$f_{max} \propto \frac{(V - c)}{V}$$

6 Non-Linear Properties of Li+ Batteries

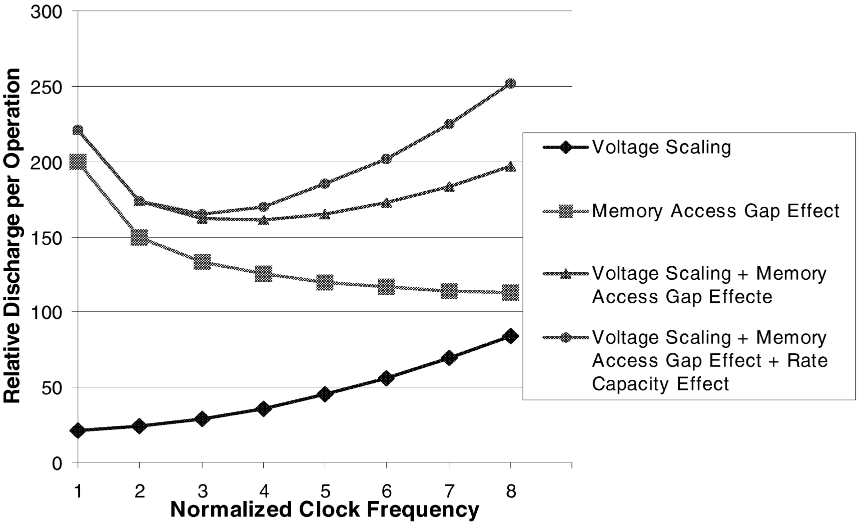




Reference:[MS99]

7 Battery- & Memory-Aware DVS

- Power depends on speed/voltage & application [PLS01, FEL02]
- Respecting the rate capacity effect in memory-conscious DVS.



- Related work: solar power aware DVS

F.2 Speed Setting Policies

1 Static frequency scaling

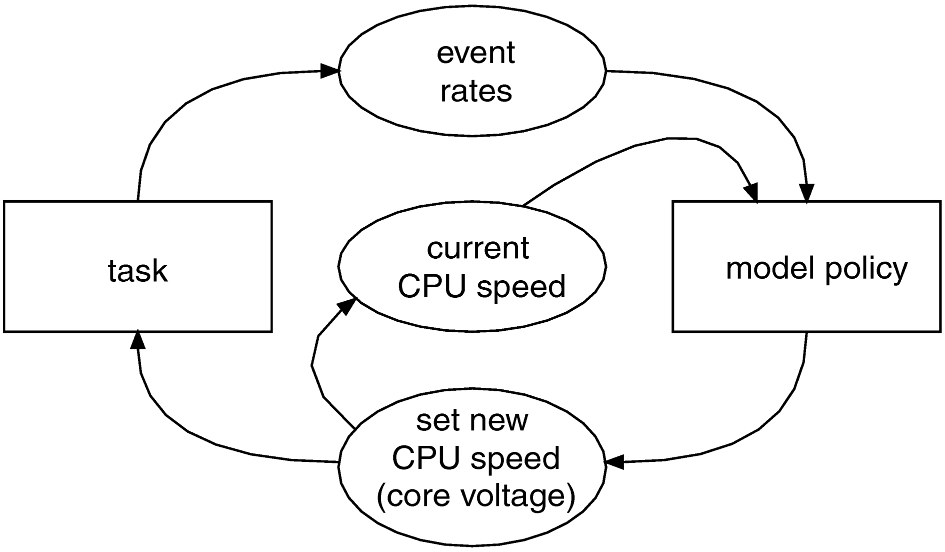
- ◆ Clock frequency for each application is determined according to off-line power/performance analysis.
- ◆ Clock frequency is a static value of the process context
  - Value is part of the binary
  - OS provides system call interface for speed settings.

2 Dynamic frequency/voltage scaling (DVS)

- ◆ Clock frequency is periodically set for the complete system
  - PAST [WWDS94, GCW95], Vertigo [FM02]
- ◆ Clock frequency for each application is dynamically adjusted according to
  - Run-time requirements (deadlines, response times, performance)
  - Run-time characteristics (execution speed depends on input data, inter-process communication, I/O use, energy consumption)
  - Run-time behavior of the system (load, battery capacity, temperature)

3 Process Cruise Control

- Principles of operation
  - ◆ The rates at which some events happen correspond to a specific performance/energy-efficiency profile.



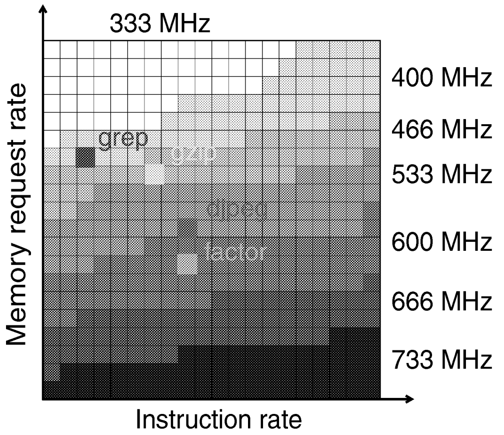
Reference:[WB02]

3 Process Cruise Control

- Event-based characterization
  - ◆ Challenge:
    - • Selection of the appropriate events
      - – Clock cycles, retired instructions, L1 cache access, memory/bus access...
    - • Finding the correlation between event rates and clock speed for predefined power/performance demands.
  - ◆ Methodology of event-based power/performance characterization:
    - • Countable events are triggered by synthetic training applications.
    - • Energy at various clock speeds is measured by a DAQ system.
    - • The rate of various events is determined.
    - • The performance of the training application is evaluated.
    - • For each training application the optimal clock speed is determined for a predefined performance penalty.

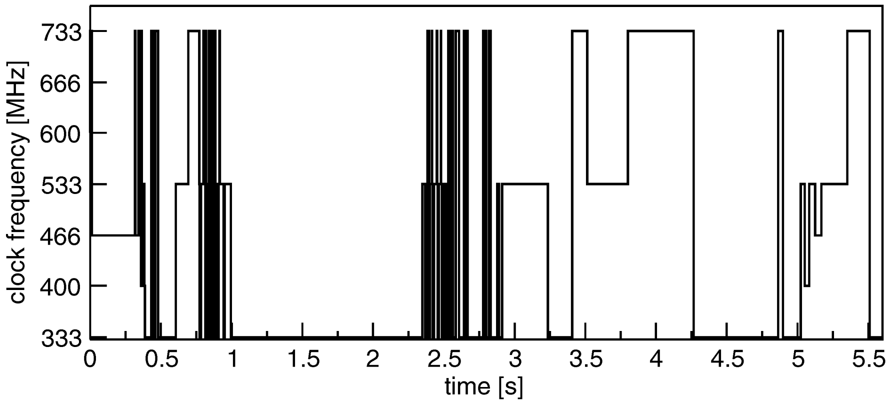
3 Process Cruise Control

- Frequency domains for IQ80310
  - ◆ Selection of events:
    - Instruction rate is an indicator for performance loss.
    - Memory request rate is an indicator for energy efficiency gains.
  - ◆ Example: Partitioning of the event space into frequency domains for a 10% loss in computational performance



3 Process Cruise Control: Measurements

Application	optimal speed	Process Cruise Control: clock scaling	Process Cruise Control: energy savings
grep	400 MHz	400 MHz	15%
gzip	466 MHz	466 MHz	10%
djpeg	600 MHz	533 MHz	8%
factor	600 MHz	600 MHz	4%
ghostscript		dynamic	< 5%



4 Implementation Issues

- Policy implementation
  - Scheduler-hooks or inlining?
  - TLB effects of loadable modules
  - Task-specific policies
  - User-level policies
  - Floating point operations
- Response delays
  - Raise of voltage before raise of frequency
  - Many system clocks are derived from CPU clock frequency
    - ➜ pre-scaling, post-scaling driver calls
- Kernel interface
  - sysfs: exporting kernel data structures and attributes to user space

G Conclusion

- Event-driven power/performance characterization can be accomplished on the
  - System level
  - Task level
- Events cover all energy-related activities caused by the
  - application
  - interaction of applications
  - interaction of application and operating system
- Events-driven power management is independent of specific applications or workloads

➜ Event-driven power/performance characterization is the basis of energy-aware systems.

References

[ABD+97] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S.-T. Leung, R. Sites, M. Vandervoorde, C. Waldspurger, and W. Weihl. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems*, 15(4), Nov 1997.

[BDM99] Gaurav Banga, Peter Druschel, and Jeffrey Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation OSDI'99*, Feb 1999.

[Bel00] Frank Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*, Sep 2000.

[Bel01] F. Bellosa. The case for event-driven energy accounting. Technical Report TR-I4-01-07, University of Erlangen, Department of Computer Science, June 2001.

[BM01] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings Of The Seventh International Symposium On High-Performance Computer Architecture (HPCA'01)*, Jan 2001.

[BTM00] D. Brooks, Vivek Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings Of The 27th Annual International Symposium on Computer Architecture ISCA-27*, June 2000.

[FEL02] Xiaobo Fan, Carla Ellis, and Alvin Lebeck. Interaction of power-aware memory systems and processor voltage scaling. In *Submitted for Publication*, October 2002.

[FM02] Krisztian Flautner and Trevor Mudge. Vertigo: Automatic performance-setting for linux. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation OSDI'2002*, Dec 2002.

[GBCH01] S. H. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. *Intel Technology Journal*, (1), 2001.

[GCW95] K. Govil, E. Chan, and H. Wassermann. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *Proceedings of the first Conference on Mobile Computing and Networking MOBICOM'95*, Mar 1995. also as technical report TR-95-017, ICSI Berkeley, Apr. 1995.

[GSI+02] S. Gurumurthi, A. Sivasubramaniam, M. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. John. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of The Seventh International Symposium On High-Performance Computer Architecture (HPCA'02)*, Feb 2002.

[HKS+03] J. Haid, G. Kaefer, Ch. Steger, R. Weiss, W. Schgler, and M. Manninger. Run-time energy estimation in system-on-a-chip designs. In *Proceedings of the 8th Asia and South Pasific Design Automation Conference and 15th International Conference on VLSI Design (VLSI Design / ASPDAC '03*, Jan 2003.

[Int00] Intel. *Intel 80200 Processor based on Intel XScale Microarchitecture Developer's Manual*, Nov 2000.

[Int02] Intel. *Intel Pentium 4 Processor with 512-KB L2 Cache on 0.13 Micron Process Thermal Design Guidelines Design Guide*, Nov 2002.

[Int03] Intel. *Intel Pentium M Processor Datasheet*, March 2003.

[JM01] Russ Joseph and M. Martonosi. Run-time power estimation in high-performance microprocessors. In *The International Symposium on Low Power Electronics and Design ISLPED'01*, August 2001.

[Kel03] Simon Kellner. Event-driven temperature-control in operating systems. Department of Computer Science, student thesis SA-I4-2003-02, April 2003.

[MS99] Thomas L. Martin and Daniel P. Siewiorek. Non-ideal battery properties and low power operation in wearable computing. In *Proceedings of the Third International Symposium on Wearable Computers*, pages 101–106, October 1999.

[PBB00] T. Pering, T. Burd, and R. Broderson. Voltage scheduling in the lparm microprocessor system. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'00*, July 2000.

[PLS01] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM 2001)*, July 2001.

[SKO+97] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez. Thermal management system for high performance powerpc microprocessors. In *Proceedings of IEEE Compcon'97 Digest of Papers*, Feb 1997.

[SMH01] Phillip Stanley-Marbell and Michael Hsiao. Fast, flexible, cycle-accurate energy estimation. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*, August 2001.

[Wai03] Martin Waitz. Accounting and control of power consumption in energy-aware operating systems. Department of Computer Science, diploma thesis SA-I4-2002-14, January 2003.

[WB02] Andreas Weissel and Frank Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems CASES'02*, Oct 2002.

[WWDS94] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the First Symposium on Operating System Design and Implementation OSDI'94*, Nov 1994.

Energy Management at Upper Levels of System Design

Carla Schlatter Ellis

Department of Computer Science
Duke University
carla@cs.duke.edu

Energy Management at Upper Levels of System Design

Carla Schlatter Ellis

Milly Watt Project

Milly Watt Project

Energy for computing is an important problem (& not just for mobile computing)

- Reducing heat production and fan noise
- Extending battery life for mobile/wireless devices
- Conserving energy resources (lessen environmental impact, save on electricity costs)

How does software interact with or exploit low-power hardware?

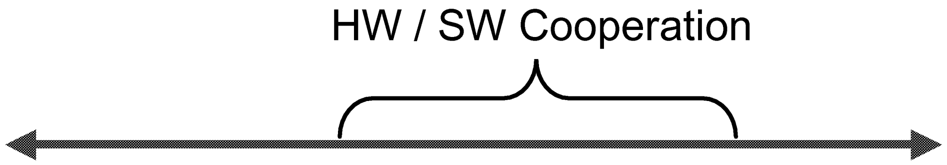
Milly Watt Project

Energy should be a “first class” resource at upper levels of system design

- Focus on Architecture, OS, Networking, Applications

HW / SW cooperation to achieve energy goals

The Energy Saving Spectrum



Hardware

- Low level
- Fine grain
- Low-power Circuits

- Voltage Scaling
- Clock gating
- Power modes: Turning off HW blocks

Software

- High level
- Coarse grain
- OS, compiler or application

- Re-examine interactions between HW and SW, particularly within the resource management functions of the Operating System

Lessons: Performance –> Energy

Latency techniques

- Substitute a lower-latency component
  - Memory Hierarchy -Caching.
- Reduce overhead on the critical path.
  - Maintenance daemons (time shift)
  - No-copy (eliminate waste)
- Exploit Overlap for *Hiding* Latency (HW parallelism)
  - Scheduling policies

Analogous techniques for Energy

- Substitute a lower-power component or mode.
  - Voltage scaling
- Amortize transitions in & out of high power states
  - Time shifting
- Reduce (not just hide) latency
  - Caching
  - No-copy (eliminate waste)

Plan of Lectures

- Wednesday morning: Explicitly managing energy via the OS (ASPLOS02, USENIX03)
- Wednesday afternoon: Power-Aware memory (ASPLOS00, ISLPED01, PACS02)
- Friday morning: Display power management (FaceOff, HOTOS03)

ECOSystem:
Managing Energy as a First-Class Operating System Resource

Outline

- Motivation / Context
- Design Issues & Challenges
- Mechanisms in the ECOSystem Framework
- Prototype Implementation & Experimental Evaluation
- Exploration of Policies

Energy & the OS

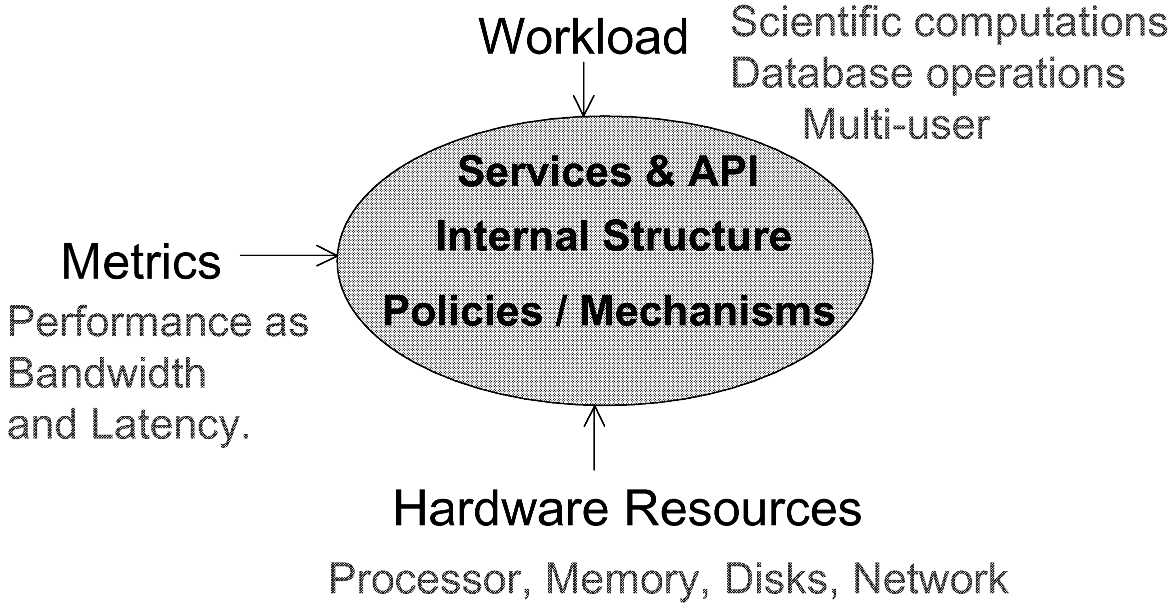
Traditionally, the system-wide view of resources and workload demands resides with the OS

- Explicitly managing energy will require coordination with typical resource management

Energy is not *just another* resource

- Energy has a impact on every other resource of a computing system – it is central.
- A focus on energy provides an opportunity to rethink OS design

Traditional Influences in OS Design



Workload

Scientific computations
Database operations
Multi-user

Services & API
Internal Structure
Policies / Mechanisms

Metrics

Performance as
Bandwidth
and Latency.

Hardware Resources

Processor, Memory, Disks, Network

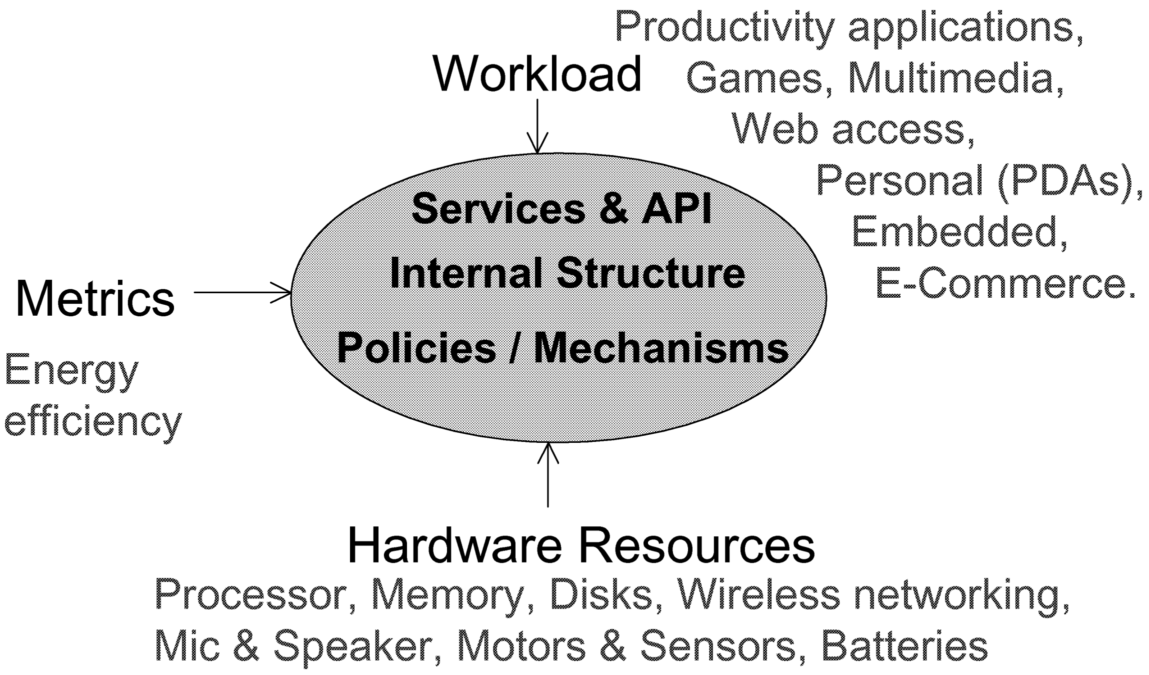
Rethinking OS Design

What is the impact of changing the primary goal of the OS to energy-efficiency rather than (speed-based) performance?

Affects every aspect of OS services and structure:

- Interfaces needed by applications that want to affect power consumption
- Internal organization and algorithms
- Resource management policies and mechanisms

Rethinking OS Design



Workload

Productivity applications, Games, Multimedia, Web access, Personal (PDAs), Embedded, E-Commerce.

Services & API

Internal Structure

Policies / Mechanisms

Metrics

Energy efficiency

Hardware Resources

Processor, Memory, Disks, Wireless networking, Mic & Speaker, Motors & Sensors, Batteries

Initial Research Question

What can be done to achieve energy-related goals

- by the OS
- without *requiring* applications to change
- with a whole-system perspective

⇒ Energy Centric Operating System

Related Work

Energy-unaware OS

- Low-power hardware, energy-aware compilers, algorithm development

  Services (Chase: MUSE)

Energy-aware OS with Unaware applications

- Per-device solutions (disk spindown, DVS)

Energy-aware OS with cooperating Energy-aware Applications

- Flinn: Odyssey (fidelity-based), Bellosa: Coop I/O, Nemesis OS

Outline

- Motivation / Context
- Design Issues & Challenges
- Mechanisms in the ECOSystem Framework
- Prototype Implementation & Experimental Evaluation
- Exploration of Policies

ECOSystem Themes

1. Energy can serve as a unifying concept for resource management over a diverse set of devices.
2. We provide a framework for explicit management of energy use:
   - Energy accounting
   - Energy allocation
   - Scheduling of energy use

Challenges

How to

- Represent the energy resource to capture its global impact on system?
- Precisely define energy goals?
- Formulate strategies that achieve those goals?

⇒ Develop a new energy abstraction:
currentcy

Challenges

How to

- Monitor system-wide energy behavior?
- Attribute energy use to correct task?
- Break down power costs by device?

⇒ Perform meaningful energy accounting:
Framework based on currentcy model and resource containers

Challenges

How to

- Manage the tradeoffs among devices?
- Handle the competition among multiple tasks?
- Reduce demand when the energy resource is limited (when applications don't know how)?

⇒ Devise and enforce energy allocation and energy-aware scheduling policies

exploiting the mechanisms in our framework.

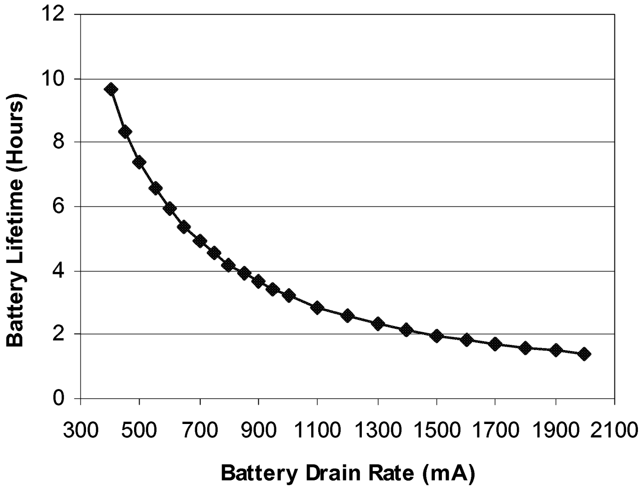
A Concrete Energy Goal: Battery Lifetime

1. Explicitly manage energy use to achieve a target battery lifetime.
   - Coast-to-coast flight with your laptop
   - Sensors that need to operate through the night and recharge when the sun comes up
2. If that requires reducing workload demand, use energy in proportion to task's importance.

   Scenario:
   - Revising and rehearsing a PowerPoint presentation
   - Spelling and grammar checking threads
   - Listening to MP3s in background

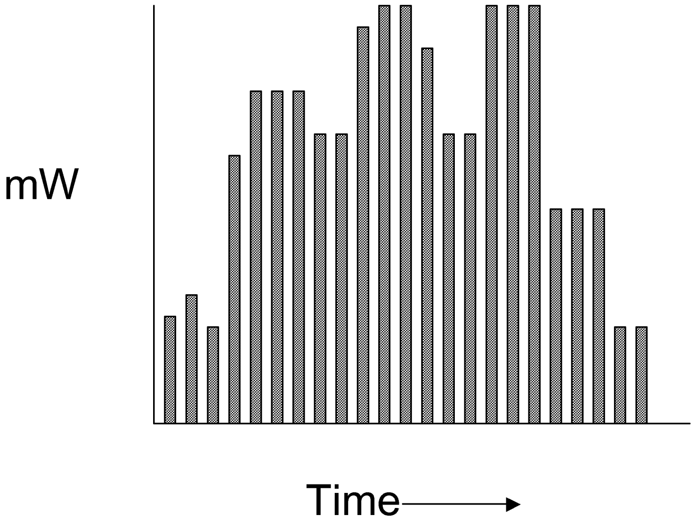
Battery Properties/Models

Battery models provide strategy:
Battery lifetime can be determined by controlling discharge rate.
Limiting availability of currentcy.



Discussion: other energy-related metrics?

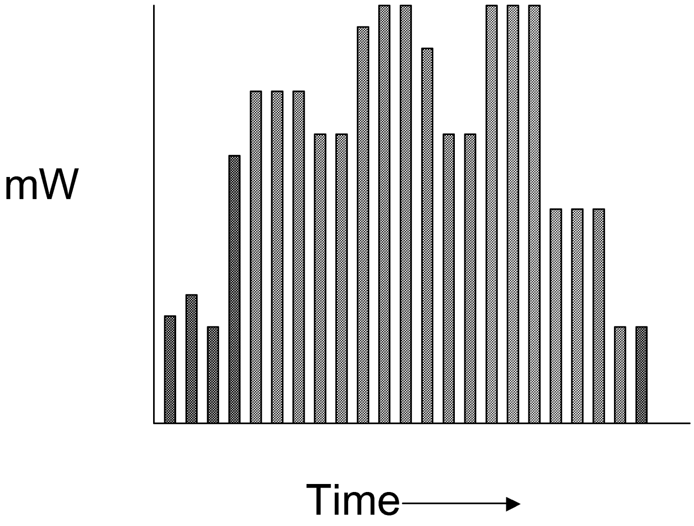
Accounting Challenges



Assume fine grain battery information – What would we see?

Smart Battery Interface – OK for coarse grain measurements

Accounting Challenges

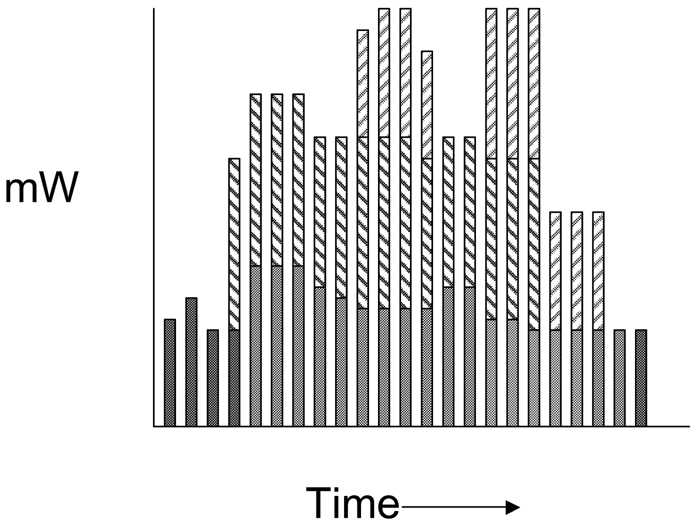


Associating observed mW to current program counter (sampling technique).

3 tasks: blue, green, and pink

Great for energy debugging single task

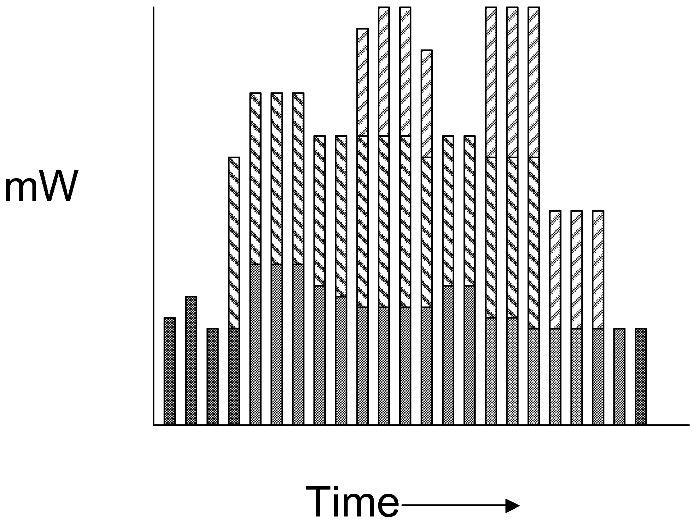
Accounting Challenges



What are the various hardware components contributing?

3 devices:
CPU (solid)
WNIC ▨
disk ▧

Accounting Challenges



How to capture these two dimensions of the accounting problem?

Our choice:
Internal power states model/ event-driven

Requires calibration

Outline

- Motivation / Context
- Design Issues & Challenges
- Mechanisms in the ECOSystem Framework
- Prototype Implementation & Experimental Evaluation
- Exploration of Policies

Unified Currentcy Model

Energy accounting and allocation are expressed in a common *currentcy.*

Mechanism for

1. Characterizing power costs of accessing resources
2. Controlling overall energy consumption
3. Sharing among competing tasks

Mechanisms in the Framework

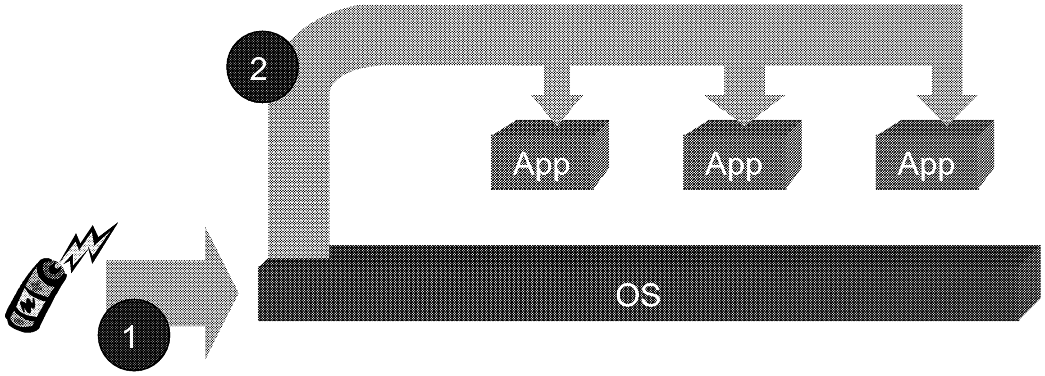
Currentcy Allocation

- Epoch-based allocation – periodically distribute currentcy "allowance"

Currentcy Accounting

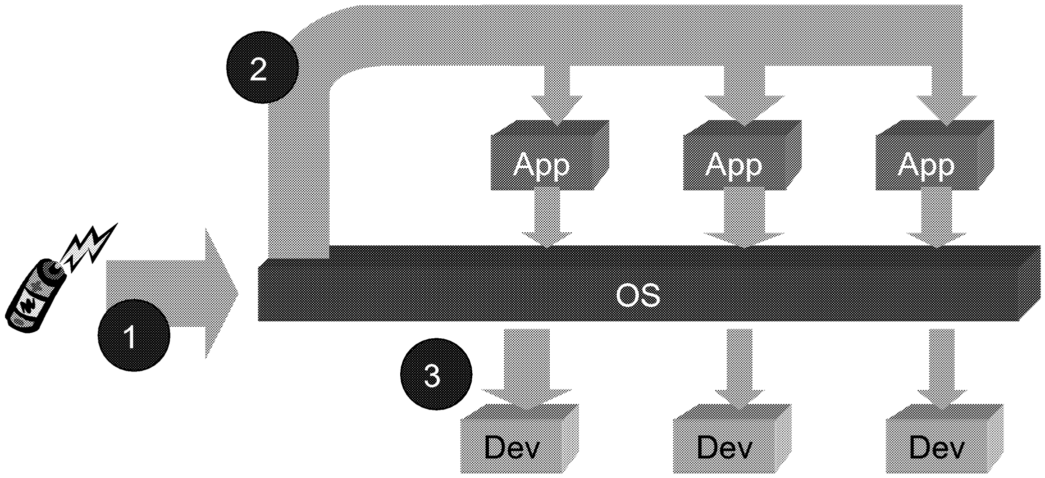
- Basic idea:
  Pay as you go for resource use –
  no more currentcy ➔ no more service.

Currentcy Flow



1. Determine *overall* amount of currentcy available per energy epoch.
2. Distribute available currentcy proportionally among tasks.

Currentcy Flow



3. Deduct currentcy from task's account for resource use.

Outline

- Motivation / Context
- Design Issues & Challenges
- Mechanisms in the ECOSystem Framework
- Prototype Implementation & Experimental Evaluation
- Exploration of Policies

ECOSystem Prototype

- Modifications to Linux on Thinkpad T20
- Initially managing 3 devices: CPU, disk, WNIC
- Embedded power model:
  - Calibrated by measurement
  - Power states of managed devices tracked

ECOSystem Prototype

Modified Linux kernel 2.4.0-test9

- Interface for specifying input parameters (target lifetime, task proportions)
- New kernel thread for currentcy allocation
- Simple implementation of resource containers
- Simple policies for allocation, scheduling, and accounting.

Resource Containers

Banga, Druschel, & Mogul, OSDI'99

Captures

- Kernel activity performed on behalf of task
- Tasks comprised of multiple processes
- Processes serving multiple tasks

ECOSystem Prototype

IBM Thinkpad T20 laptop platform

- Power model – calibrated by measurements
- 650MHz PIII CPU: 15.5W active
- Orinoco 802.11b PC card:
  doze 0.045W, receive 0.925W, transmit 1.425W
- IBM Travelstar hard disk
- Base power consumption of 13W captures everything else and inactive states of above

Hard Disk Power Model

	Cost	Timeout
Access	1.65 mJ	
Idle1	1600 mW	0.5 s
Idle2	650 mW	2 s
Idle3	400 mW	27.5 s
Standby	0 mW	
Spinup	6000 mJ	
Spindown	6000 mJ	

Discussion: How might you account for display usage in the ECOSystem framework?

Device Specific Accounting Policies

- CPU: hybrid of sampling and task switch accounting
- Disk: tasks directly pay for file accesses, sharing of spinup & spindown costs.
- Network: source or destination task pays based on length of data transferred

Experimental Evaluation

- Validate the embedded energy model.
- Can we achieve a target battery lifetime?
- Can we achieve proportional energy usage among multiple tasks?
- Assess the performance impact of limiting energy availability.

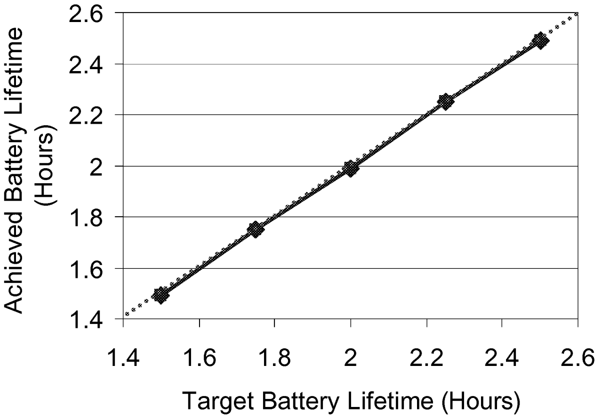
Energy Accounting

Application	Currentcy Model	PC Sampling	Back-of-envelope
Compute	8,236,720 mJ	9,094,922 mJ	8,521,400 mJ (max CPU)
DiskW	339,749 mJ	470 mJ	338,760 mJ (disk alone)
NetRecv	810,409 mJ	286,012 mJ	506,900 mJ (WNIC alone)

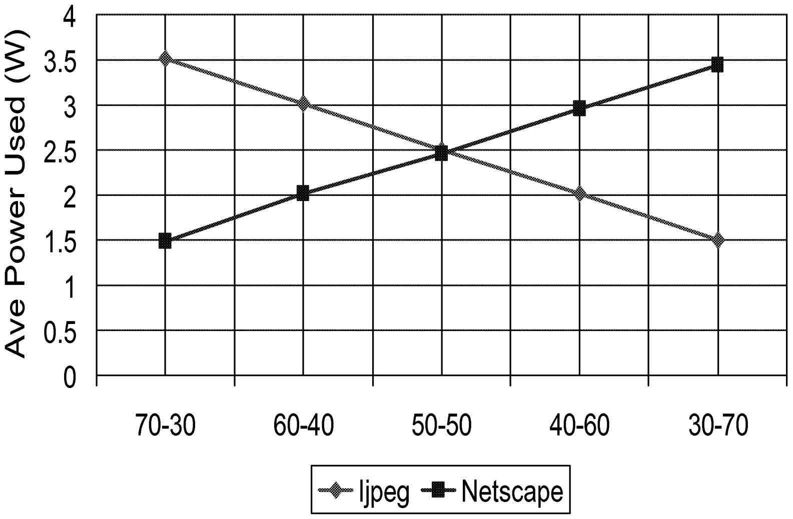
Running three tasks concurrently for 548 seconds

Achieving Target Battery Lifetime

- Using CPU intensive benchmark and varying overall allocation of currentcy, we can achieve target battery lifetime.



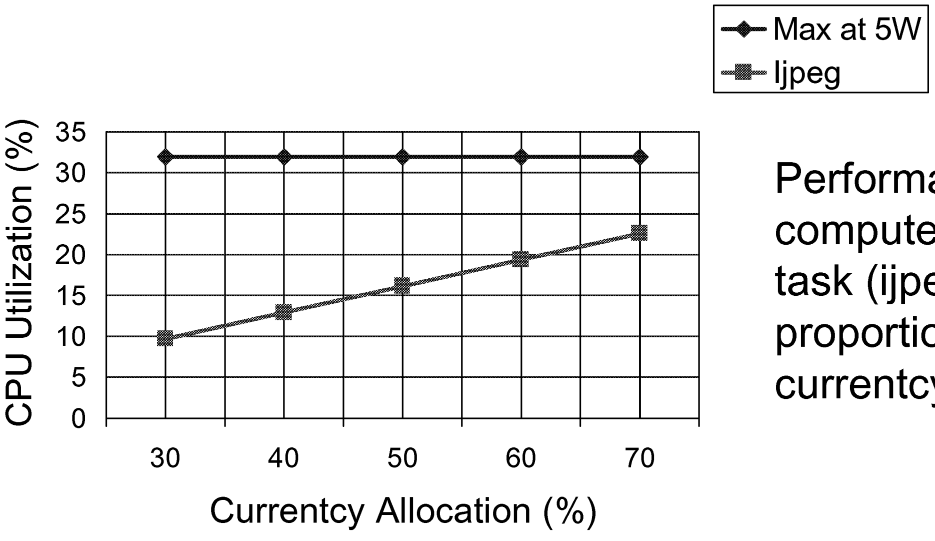
Proportional Energy Allocation



Battery lifetime is set to 2.16 hours (unconstrained would be 1.3 hr)

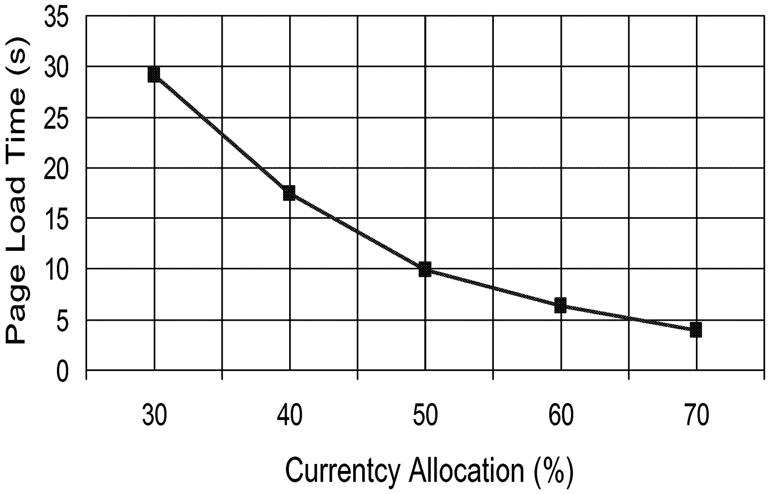
Overall allocation equivalent
to an average power consumption of 5W.

Proportional CPU Utilization



Performance of compute bound task (ijpeg) scales proportionally with currentcy allocation

Netscape Performance Impact



Some applications don’t gracefully degrade with drastically reduced currentcy allocations

Outline

- Motivation / Context
- Design Issues & Challenges
- Mechanisms in the ECOSystem Framework
- Prototype Implementation & Experimental Evaluation
- Exploration of Policies

ECOSystem Themes

1. Energy can serve as a unifying concept for managing a diverse set of resources.
   – We introduced the currentcy abstraction
2. A framework is needed for explicit management of energy. [ASPLOS 02]
   – We developed mechanisms for currentcy accounting, currentcy allocation, and scheduling of currentcy use
3. We need policies to achieve energy goals.

Previous Experiments

- Validated the embedded energy model.
- Demonstrated that we can achieve a target battery lifetime.
- Demonstrated we can achieve proportional energy usage among multiple tasks.

Research Questions

- How useful is the currentcy abstraction in articulating more complex energy goals?
- How effective are the mechanisms in the framework for manipulating currentcy in implementing non-trivial policies?
- What weaknesses are exposed in the currentcy model?

Manipulating Currentcy

- Overall currentcy allocation
  - Static vs. dynamic
  - How often, how much
- Per-task allocation
  - How much
  - Handling of unused currentcy
  - Deficit spending?
- Accounting
  - Pay-as-you-go
  - Bidding & pricing games

Challenges

1. To fully utilize available battery capacity within the desired battery lifetime with little or no leftover (residual) capacity.

⇒ Devise an allocation policy that balances supply and demand among tasks.

Currentcy conserving allocation.

Challenges

2. To produce more robust proportional sharing by ensuring adequate spending opportunities.

⇒ Develop CPU scheduling that considers energy expenditures on non-CPU resources.

Currentcy-aware scheduling.

Challenges

3. To reduce response time variability when energy is limited.

⇒ Design a scheduling policy that controls the pace of currentcy consumption.

Challenges

4. To encourage greater energy efficiency (lower average cost) for I/O accesses on power-managed disks.

⇒ Amortize spinup and spindown costs over multiple disk requests by shaping request patterns.

Buffer management and prefetching strategies.

Experience

Identified performance implications of limiting energy availability that motivate this work:

- Mismatches between user-supplied specifications and actual needs of the task
- Other devices causing a form of priority inversion
- Currentcy-starved behaviors (infeasible goals)

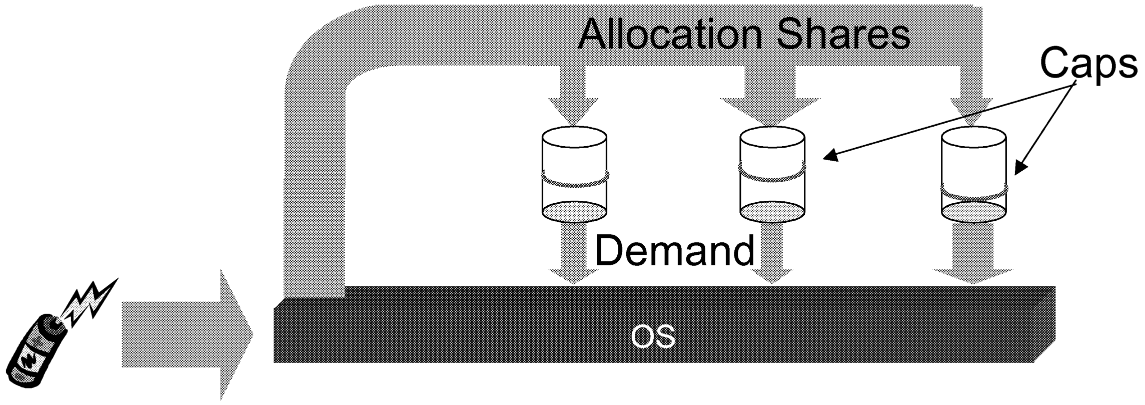
Challenges

1. To fully utilize available battery capacity within the desired battery lifetime with little or no leftover (residual) capacity.

⇒ Devise an allocation policy that balances supply and demand among tasks.

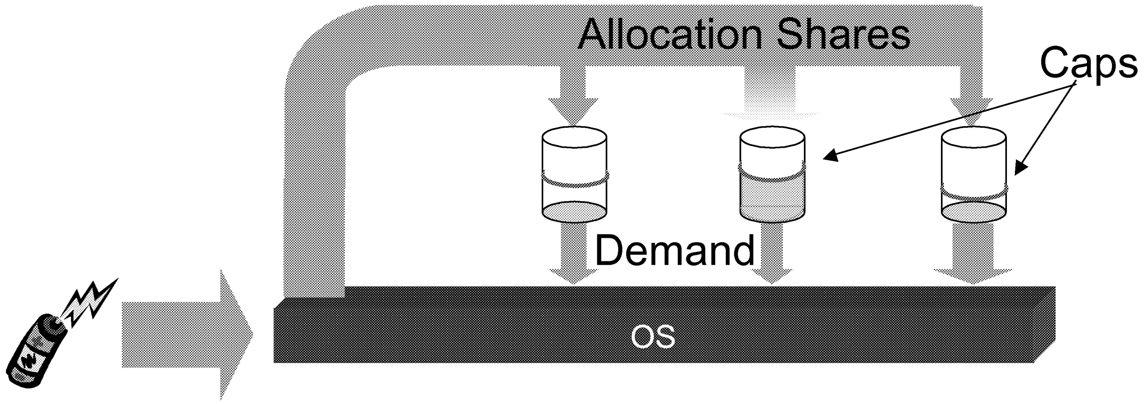
Currentcy conserving allocation.

Problem: Residual Energy



Allocations do not reflect actual consumption needs

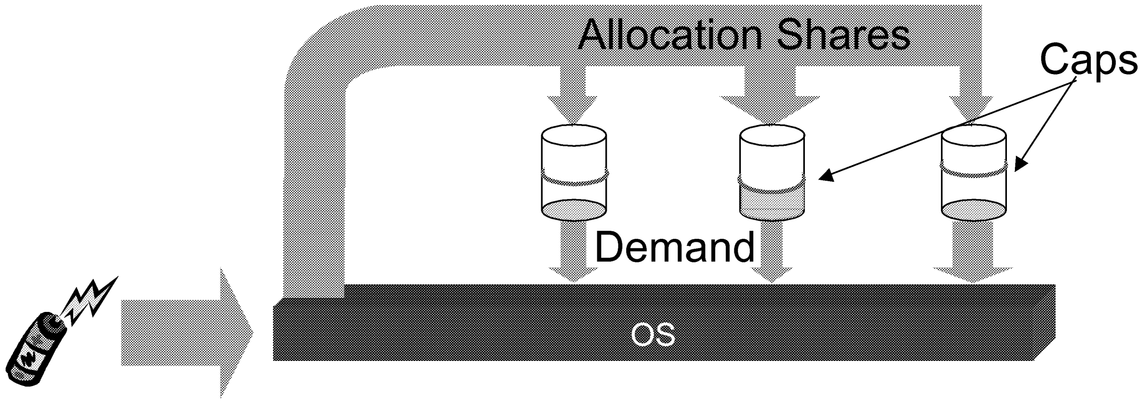
Problem: Residual Energy



A task’s unspent currentcy (above a “cap”) is being thrown away to maintain steady battery discharge.

⇒ Leftover energy capacity at end of lifetime.

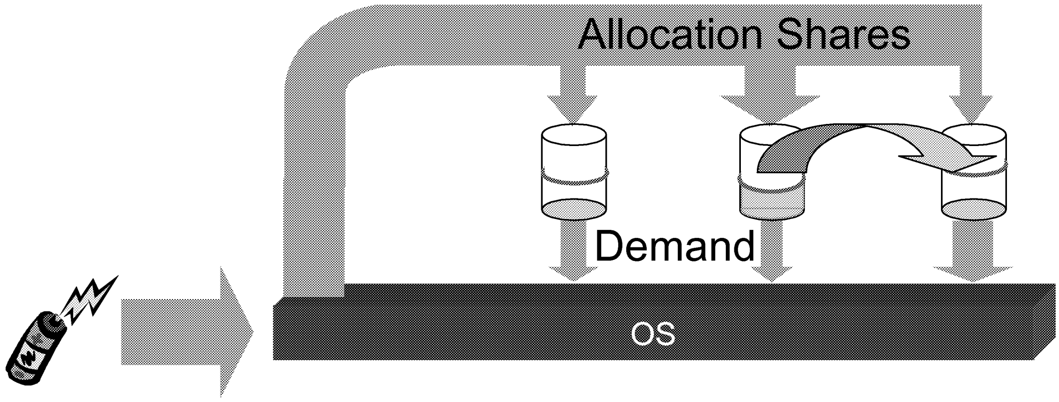
Currentcy Conserving Allocation



Two-step policy. Each epoch:

1. Adjust per-task caps to reflect observed need
   - Weighted average of currentcy used in previous epochs.

Currentcy Conserving Allocation



2. Redistribute overflow currentcy

Currentcy Conserving Allocation Experiment

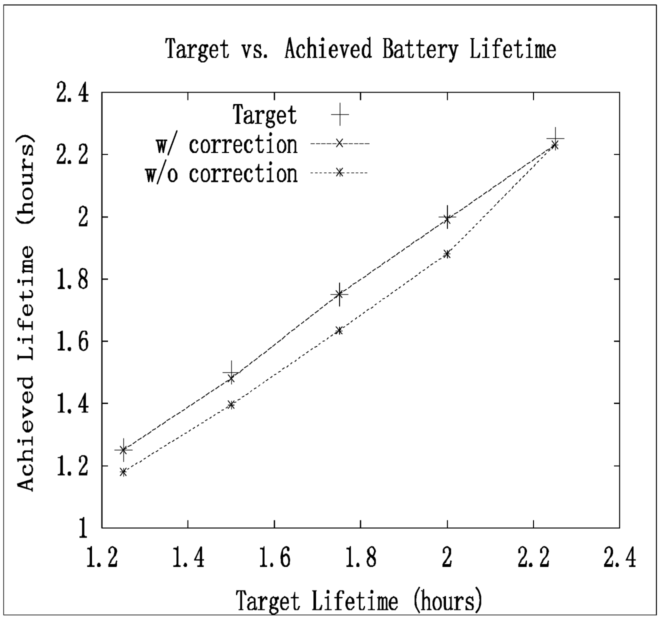
Workload:

- Computationally intensive ijpeg – image encoder
- Image viewer, gqview, with think time of 10 seconds and images from disk
  - Performance levels out at 6500mW allocation.
- Total allocation of 12W, shares of 8W for gqview (too much) and 4W for ijpeg (capable of 15.5W).

Comparing against total allocation "correction" method in original prototype.

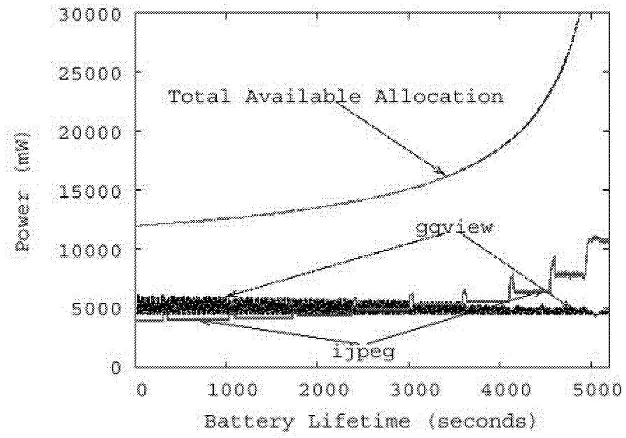
Comparison to Auto-correction

- Query the smart battery periodically
- Make adjustment to the overall currentcy each epoch to correct any error introduced by our models.
- In this test
  - we set the power consumption of CPU to be 14W instead of measured 15.5W
  - Run a CPU intensive benchmark w/ and w/o auto correction



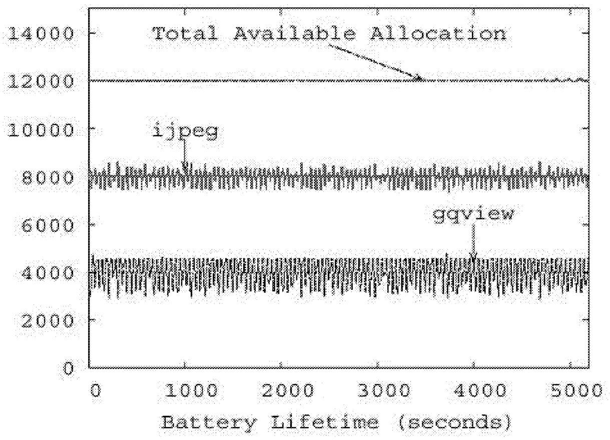
Currentcy Conserving Allocation Results

6.7% remaining



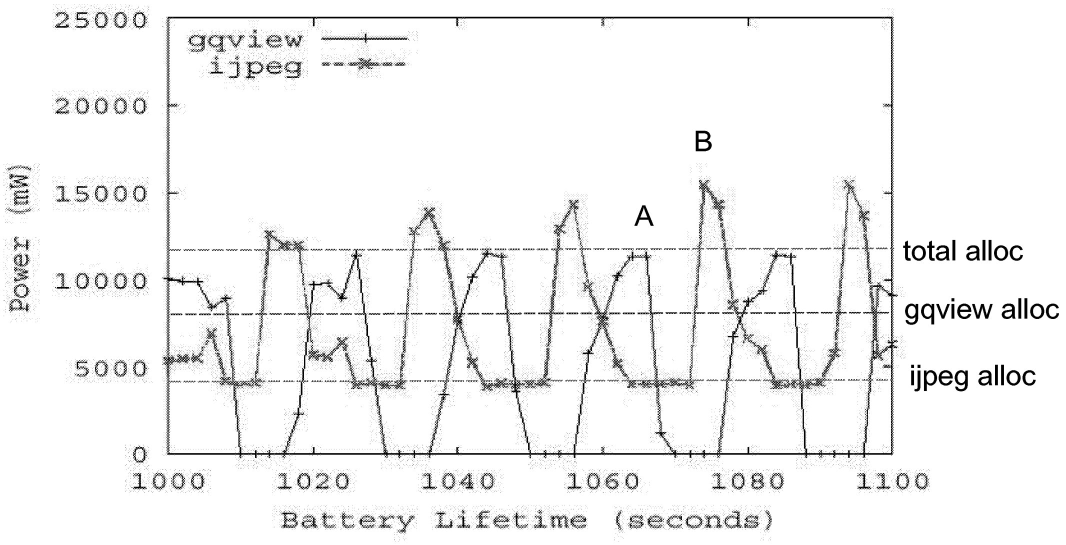
Total allocation correction

<1% remaining



Currentcy Conserving

Currentcy Conserving Allocation Results



<1% remaining capacity

Challenges

2. To produce more robust proportional sharing by ensuring adequate spending opportunities.

⇒ Develop CPU scheduling that considers energy expenditures on non-CPU resources.

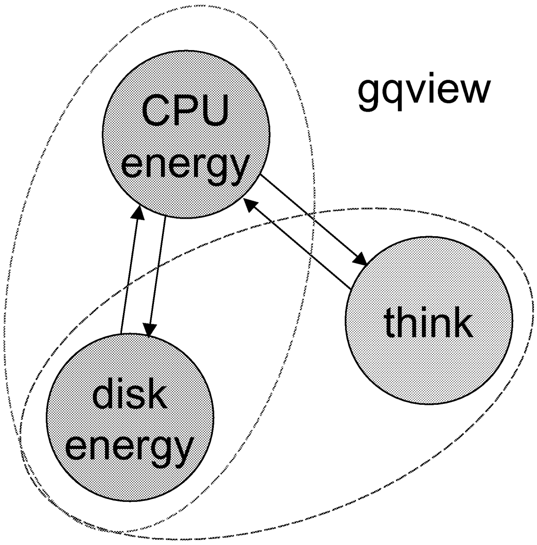
Currentcy-aware scheduling or energy-centric scheduling.

Problem: Scheduling/ Allocation Interactions

- Allocation shares may be appropriately specified and consistent with demand, but the ability to spend depends on scheduling policies that control the opportunities to access resource.
- Priority Inversion – a task with small allocation but large CPU component can dominate a task with larger allocation but demands on other devices.
- Scheduling should be “aware” of currentcy expenditures throughout the system.

Problem: Scheduling/ Allocation Interactions

- Traditional schedulers
  - Explicitly deal with CPU time and processes on ready queue
  - May implicitly compensate for time spent off ready queue
- Energy-aware
  - Deals with energy use outside of CPU
  - Currentcy explicitly captures progress using multiple devices



Energy-Centric Scheduling

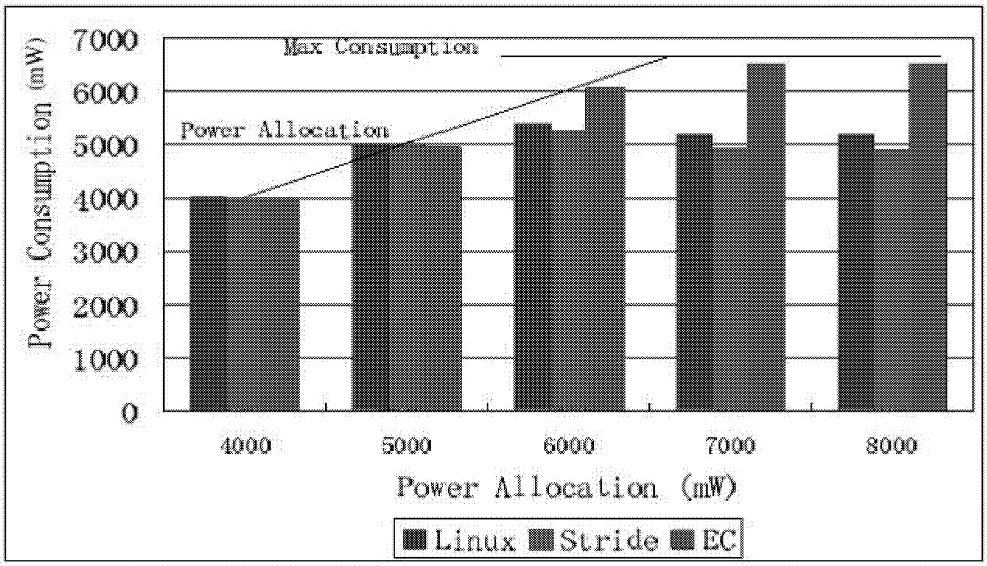
- The next task to be scheduled for CPU is the one with the lowest amount of currentcy spent in this epoch relative to its share
  - Captures currentcy spent on any device.

- Dynamic share – weighted by the task's static share divided by currentcy spent in last epoch.
  - Compensation for previous lack of spending opportunities

Energy-Centric Scheduling Experiment

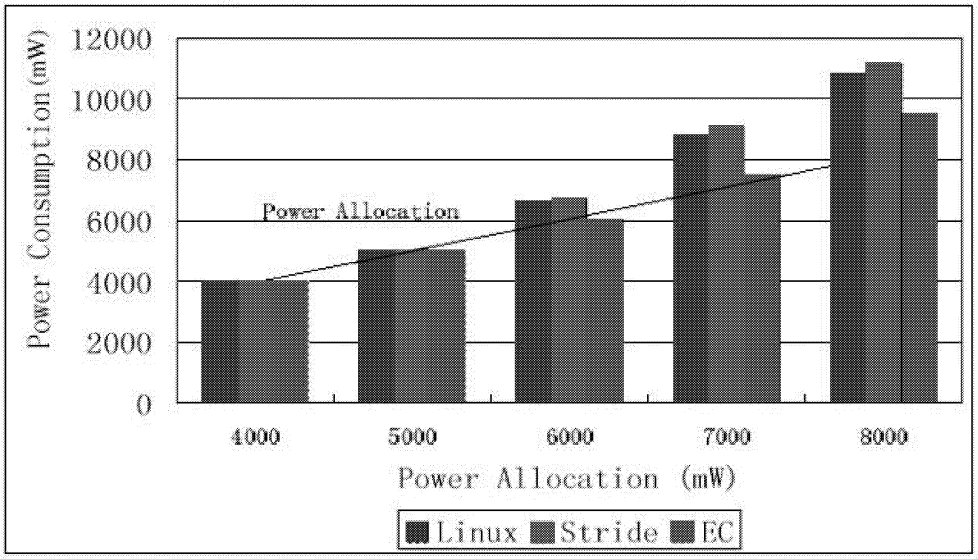
- Workload:
  - Computationally intensive ijpeg
  - Image viewer, gqview, with think time of 10 seconds and disk access (700mW)
    - Performance levels out at 6500mW allocation.
  - Given equal allocation shares, total allocation varied
- Comparing against round-robin and stride based on static share value.

Energy-Centric Scheduling Results



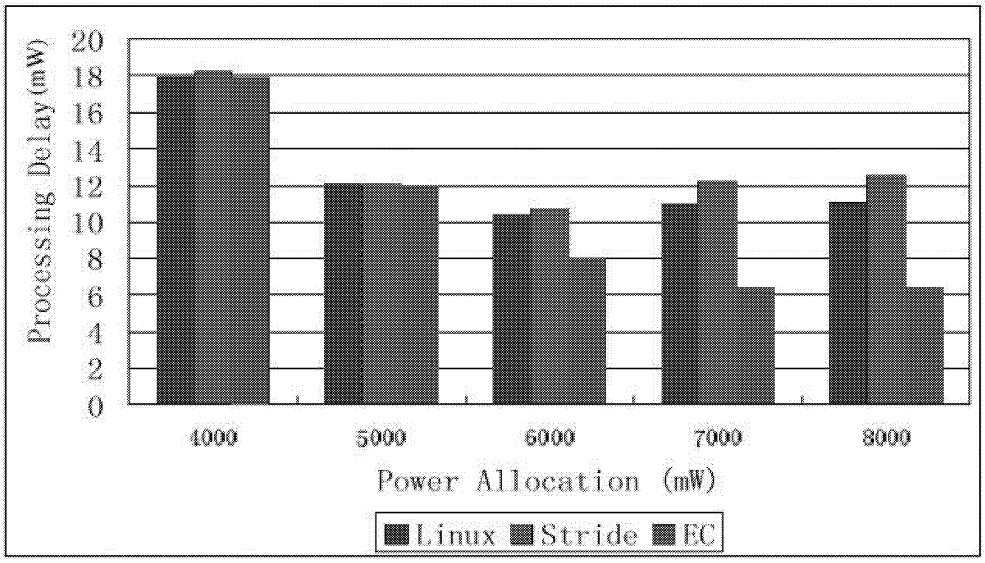
Gqview power consumption

Energy-Centric Scheduling Results



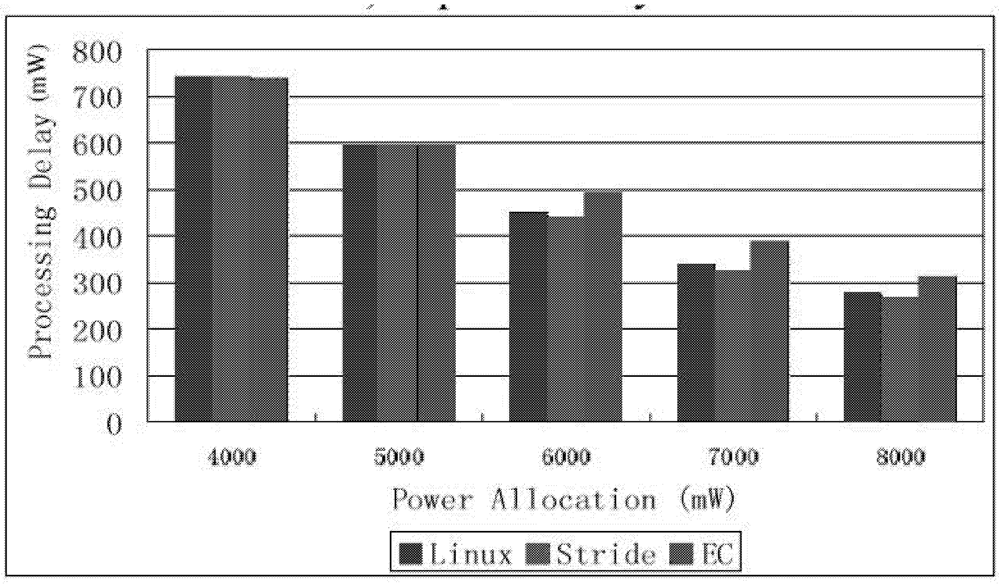
ljpeg power consumption

Energy-Centric Scheduling Results



Gqview delay

Energy-Centric Scheduling Results



ljpeg delay

Challenges

3. To reduce response time variability when energy is limited.

⇒ Design a scheduling policy that controls the *pace* of currentcy consumption.

Response Time Variation

	Power (mW)	Delay - range (s)	Ave. (s)	Std. Dev.
Unconstrained	2197	0.27- 0.68	0.43	0.11
Epoch length	1212	1.0 - 33.8	3.8	5.8
Self-pacing	1199	3.3 - 5.6	4.0	0.6

Challenges

4. To encourage greater energy efficiency (lower average cost) for I/O accesses on power-managed disks.

⇒ Amortize spinup and spindown costs over multiple disk requests by shaping request patterns.

Buffer management and prefetching strategies.

Benefits of Currentcy

Currentcy abstraction

- Provides a concrete representation of energy supply and demand – allowing explicit energy/power management.
- Provides unified view of energy impact of different devices – enabling multi-device, system-wide resource management
  - Comparable, quantifiable, tradeoffs can be expressed
- Encourages analogies to economic models – motivating a rich set of policies.

Future Work/Open Questions with ECOSystem

Using currentcy and ECOSystem to explore

- More of the policy design space: other energy goals, other objectives, other approaches to achieve these objectives
- The power of economic models based on currentcy
- Application assistance by extending API
- Admission control
- Including more components of the system: display, virtual memory,

Power-Aware Memory Management

Outline

- Motivation for memory power/energy management and the opportunity
- Hardware power management
- OS page allocation policies
- Experimental results
- Future work, open questions

Memory: The Unturned Stone

Previous Architecture/OS Energy Studies:

- Disk spindown policies [Douglis, Krishnan, Helmbold, Li]
- Processor voltage and clock scaling [Weiser, Pering, Lorch, Farkas et al]
- Network Interface [Stemm, Kravets]
- Mems-based storage [Nagle et al]
- Application-aware adaptation & API [Flinn&Satya]

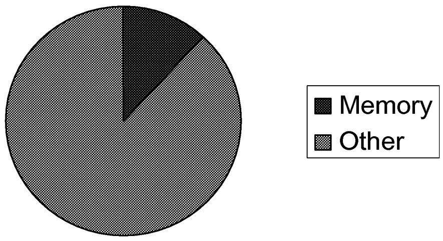
- But where is main memory management?

Power Aware Page Allocation [ASPLOS00]

Memory System Power Consumption

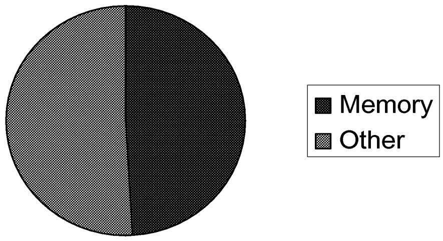
Laptop Power Budget

9 Watt Processor



Handheld Power Budget

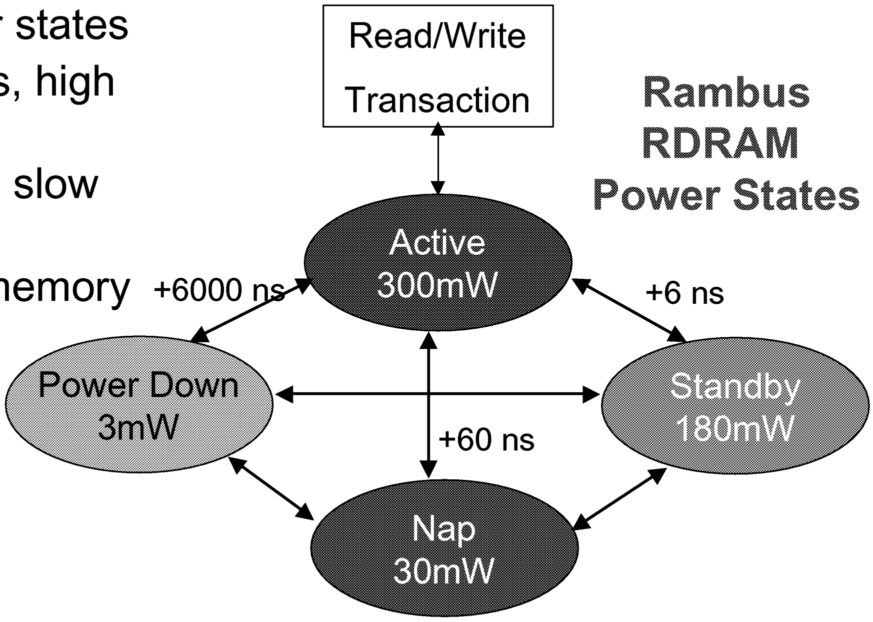
1 Watt Processor



- Laptop: memory is small percentage of total power budget
- Handheld: low power processor, memory is more important

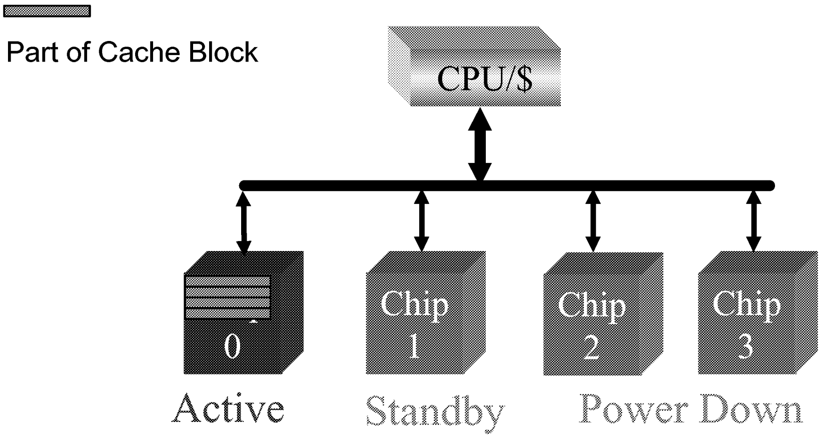
Opportunity: Power Aware DRAM

- Multiple power states
  - Fast access, high power
  - Low power, slow access
- New take on memory hierarchy
- How to exploit opportunity?



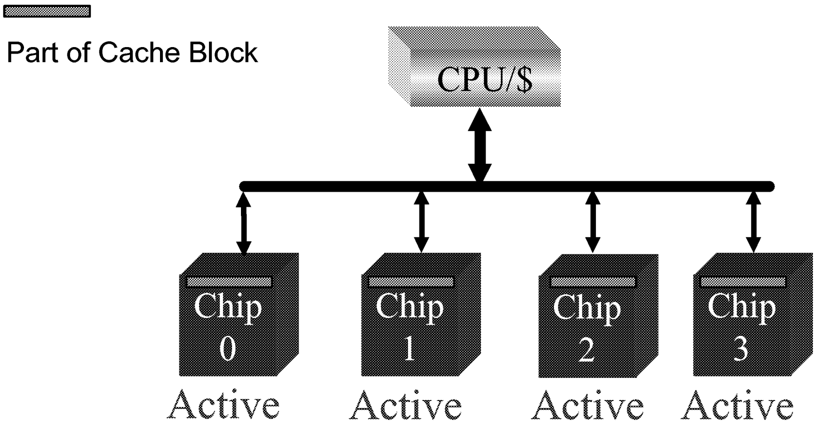
Rambus RDRAM Power States

RAMBUS RDRAM Main Memory Design



- Single RDRAM chip provides high bandwidth per access
  - Novel signaling scheme transfers multiple bits on one wire
  - Many internal banks: many requests to one chip
- Energy implication: Activate only one chip to perform access at same high bandwidth as conventional design

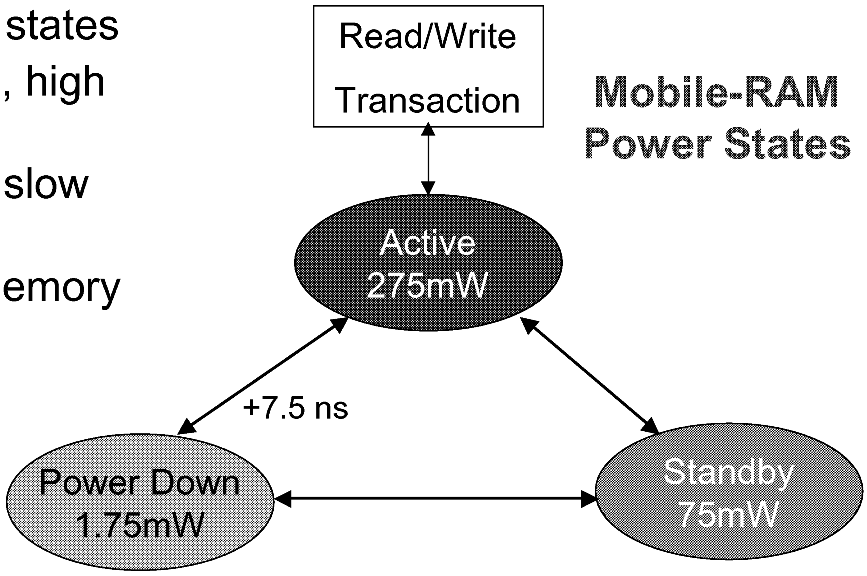
Conventional Main Memory Design



- Multiple DRAM chips provide high bandwidth per access
  - Wide bus to processor
  - Few internal banks
- Energy implication: Must activate all those chips to perform access at high bandwidth

Opportunity: Power Aware DRAM

- Multiple power states
  - Fast access, high power
  - Low power, slow access
- New take on memory hierarchy
- How to exploit opportunity?

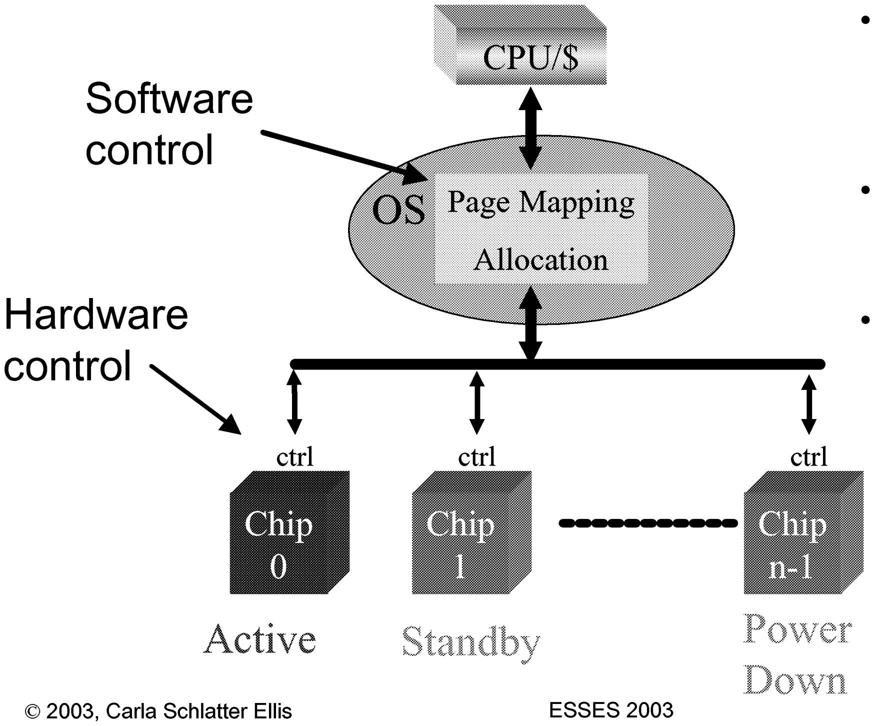


Exploiting the Opportunity

Interaction between power state model and access locality

- How to manage the power state transitions?
  - Memory controller policies
  - Quantify benefits of power states
- What role does software have?
  - Energy impact of allocation of data/text to memory.

Power-Aware DRAM Main Memory Design



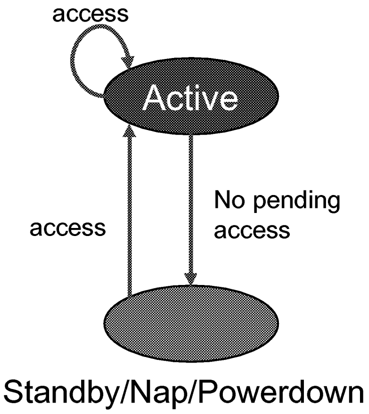
- Properties of RDRAM allow us to access and control each chip individually
- 2 dimensions to affect energy policy: HW controller / OS
- Energy strategy:
  - Cluster accesses to already powered up chips
  - Interaction between power state transitions and data locality

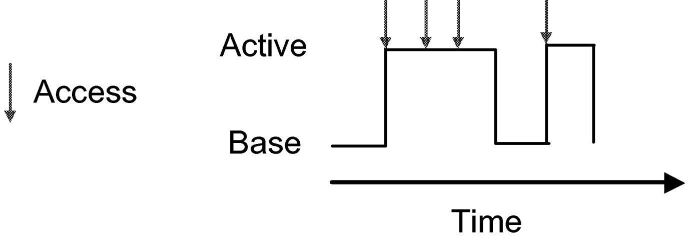
Outline

- Motivation for memory power/energy management and the opportunity
- Hardware power management
- OS page allocation policies
- Experimental results
- Future work, open questions

Dual-state HW Power State Policies

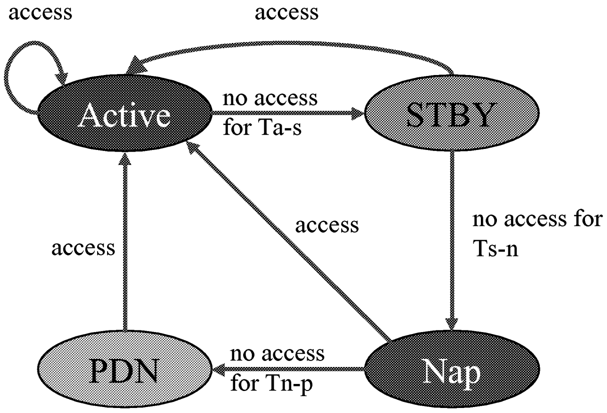
- All chips in one base state
- Individual chip Active while pending requests
- Return to base power state if no pending access

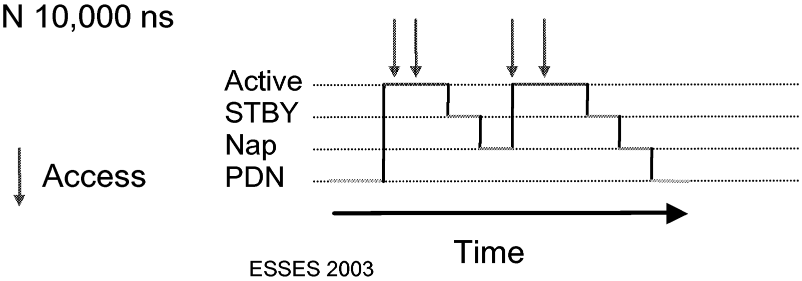




Quad-state HW Policies

- Downgrade state if no access for threshold time
- Independent transitions based on access pattern to each chip
- Competitive Analysis
  - rent-to-buy
  - Active to nap 100’s of ns
  - Nap to PDN 10,000 ns

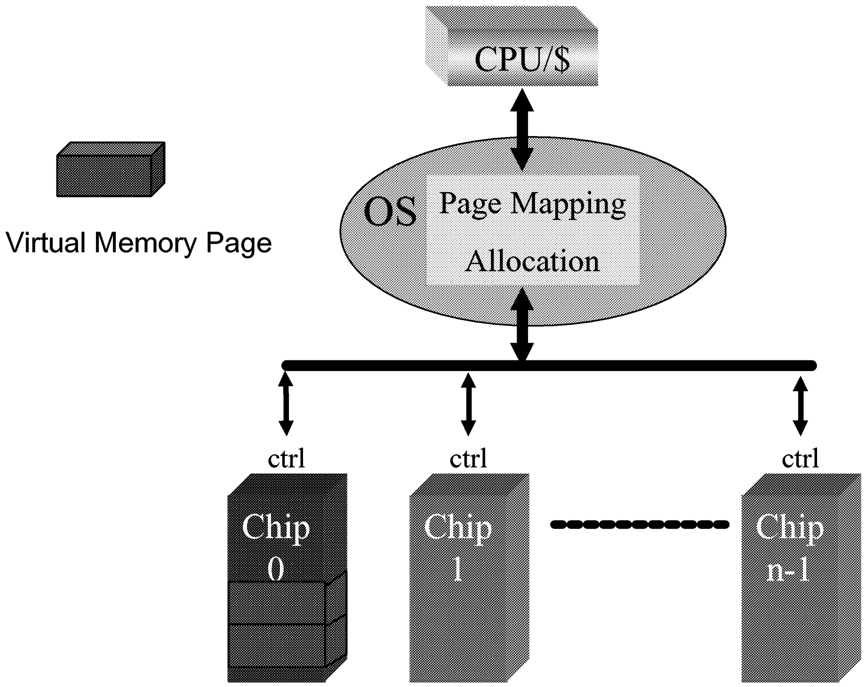




Outline

- Motivation for memory power/energy management and the opportunity
- Hardware power management
- OS page allocation policies
- Experimental results
- Future work, open questions

Page Allocation and Power-Aware DRAM



- Physical address determines which chip is accessed
- Assume non-interleaved memory
  - Addresses 0 to N-1 to chip 0, N to 2N-1 to chip 1, etc.
- Entire virtual memory page in one chip
- Virtual memory page allocation influences chip-level locality

Page Allocation Polices

Virtual to Physical Page Mapping

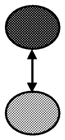
- Random Allocation – baseline policy
  - Pages spread across chips
- Sequential First-Touch Allocation
  - Consolidate pages into minimal number of chips
  - One shot
- Frequency-based Allocation
  - First-touch not always best
  - Allow (limited) movement after first-touch

Outline

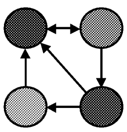
- Motivation for memory power/energy management and the opportunity
- Hardware power management
- OS page allocation policies
- Experimental results
- Future work, open questions

The Design Space

	Random Allocation	Sequential Allocation
Dual-state Hardware	1 Simple HW	2 Can the OS help?
Quad-state Hardware	3 Sophisticated HW	4 Cooperative HW & SW



2 state model



4 state model

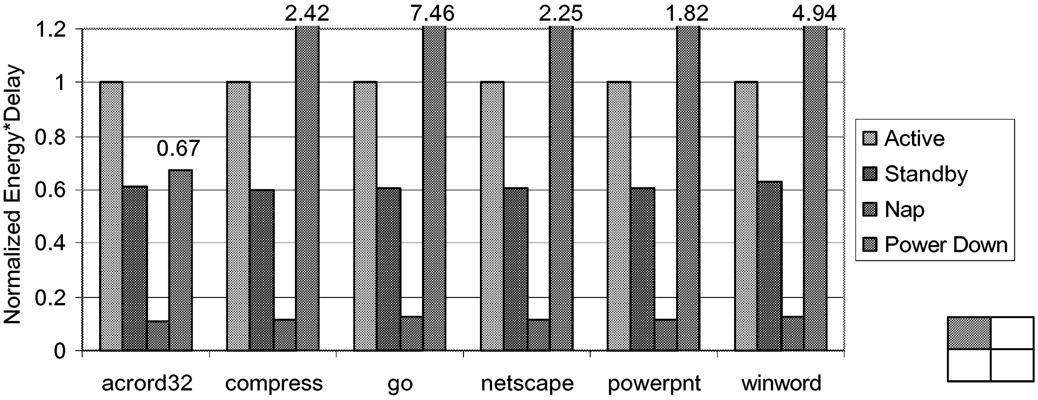
Methodology

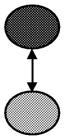
- Metric: Energy*Delay Product
  - Avoid very slow solutions
- Energy Consumption (DRAM only)
  - Processor & Cache affect runtime
  - Runtime doesn’t change much in most cases
- 8KB page size
- L1/L2 non-blocking caches
  - 256KB direct-mapped L2
  - Qualitatively similar to 4-way associative L2
- Average power for transition from lower to higher state
- Trace-driven and Execution-driven simulators

Methodology Continued

- Trace-Driven Simulation
  - Windows NT personal productivity applications (Etch at Washington)
  - Simplified processor and memory model
  - Eight outstanding cache misses
  - Eight 32Mb chips, total 32MB, non-interleaved
- Execution-Driven Simulation
  - SPEC benchmarks (subset of integer)
  - SimpleScalar w/ detailed RDRAM timing and power models
  - Sixteen outstanding cache misses
  - Eight 256Mb chips, total 256MB, non-interleaved

Dual-state + Random Allocation (NT Traces)

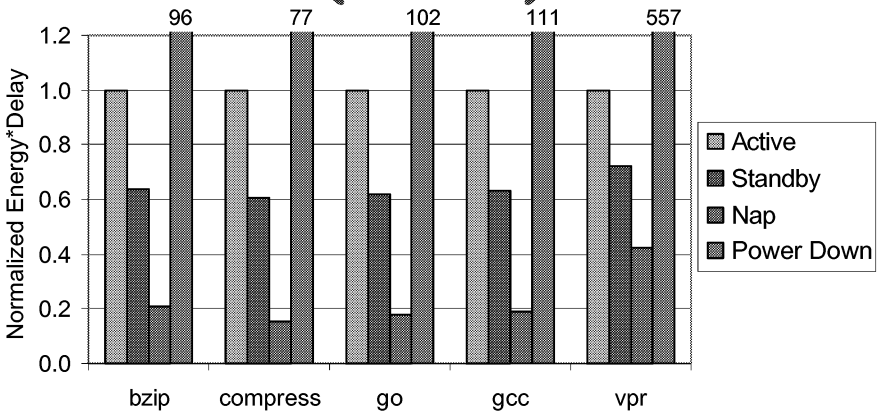




2 state model

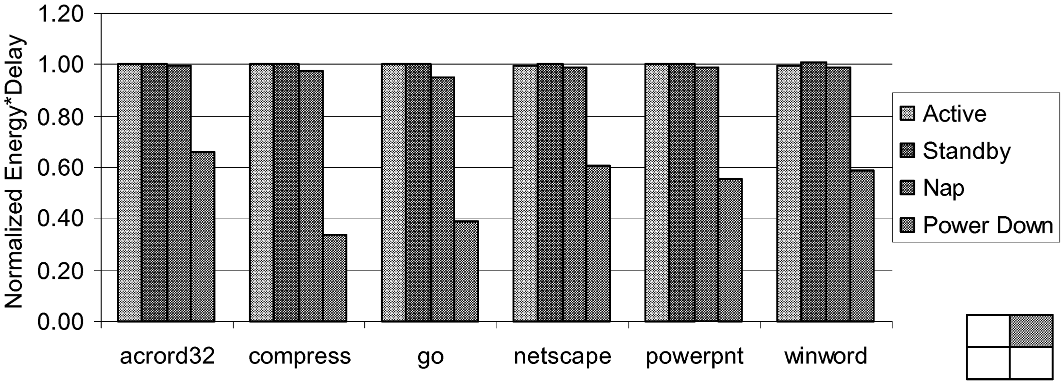
- Active to perform access, return to base state
- Nap is best ~85% reduction in E*D over full power
- Little change in run-time, most gains in energy/power

Dual-state + Random Allocation (SPEC)



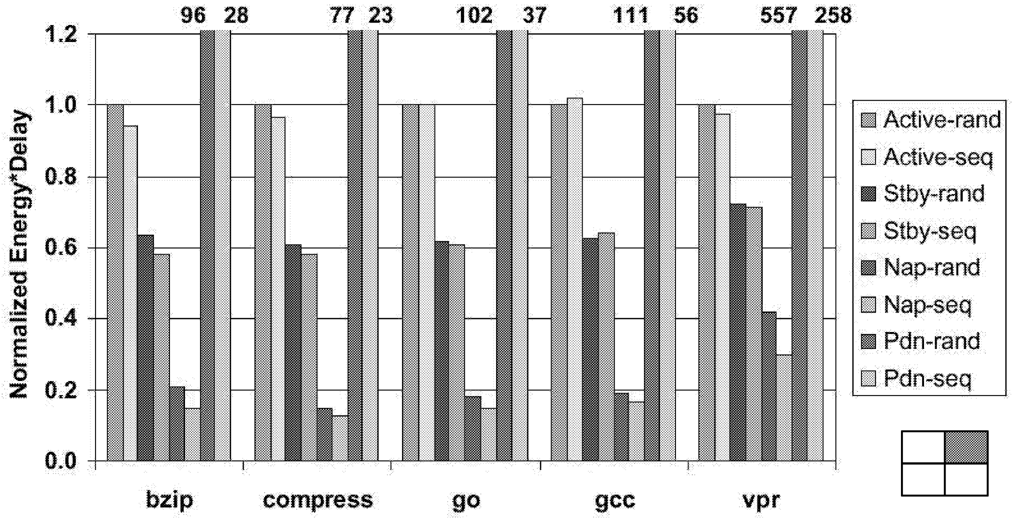
- All chips use same base state
- Nap is best 60% to 85% reduction in E*D over full power
- Simple HW provides good improvement

Benefits of Sequential Allocation (NT Traces)



- Sequential normalized to random for same dual-state policy
- Very little benefit for most modes
  - Helps PowerDown, which is still really bad

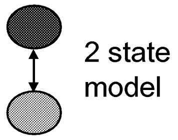
Benefits of Sequential Allocation (SPEC)

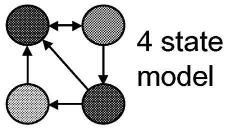


- 10% to 30% additional improvement for dual-state nap
- Some benefits due to cache effects

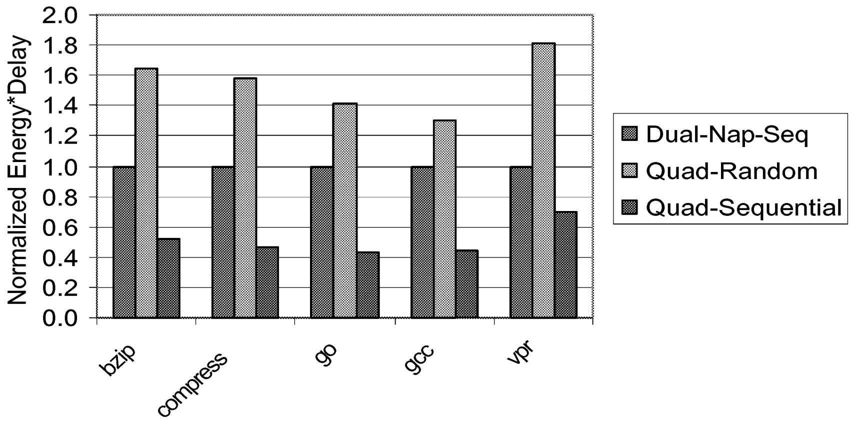
Results (Energy*Delay product)

	Random Allocation	Sequential Allocation
Dual-state Hardware	Nap is best 60%-85% improvement	10% to 30% improvement for nap. Base for future results
Quad-state Hardware	What about smarter HW?	Smart HW and OS support?



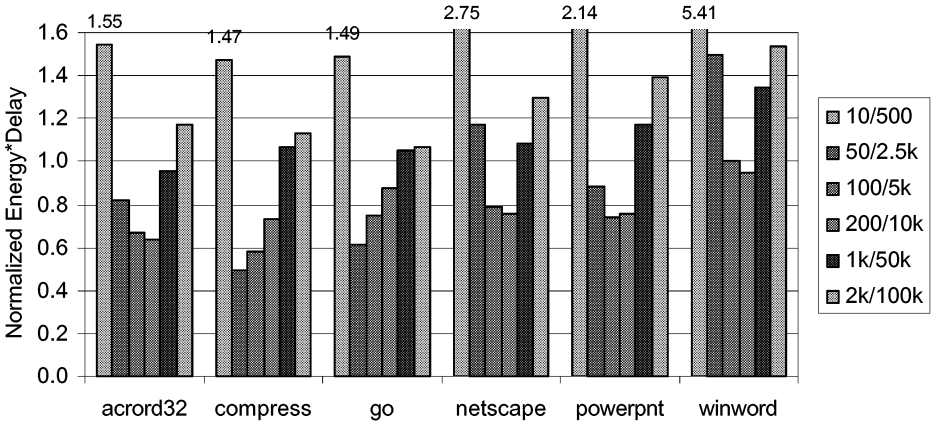


Quad-state HW (SPEC)



- Base: Dual-state Nap Sequential Allocation
- Thresholds: 0ns A->S; 750ns S->N; 375,000 N->P
- Quad-state + Sequential 30% to 55% additional improvement over dual-state nap sequential
- HW / SW Cooperation is important

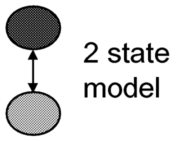
Threshold Sensitivity: Quad-state HW + Sequential Allocation (NT)

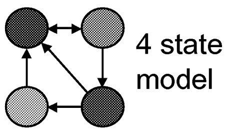


- Quad-state vs. Dual-state nap sequential
- Bars: active->nap / nap ->powerdown threshold values
- Additional 6% to 50% improvement over best dual-state

Results (Energy*Delay product)

	Random Allocation	Sequential Allocation
Dual-state Hardware	Nap is best dual-state policy 60%-85%	Additional 10% to 30% over Nap
Quad-state Hardware	Improvement not obvious, Could be equal to dual-state	Best Approach: 6% to 55% over dual-nap-seq, 80% to 99% over all active.





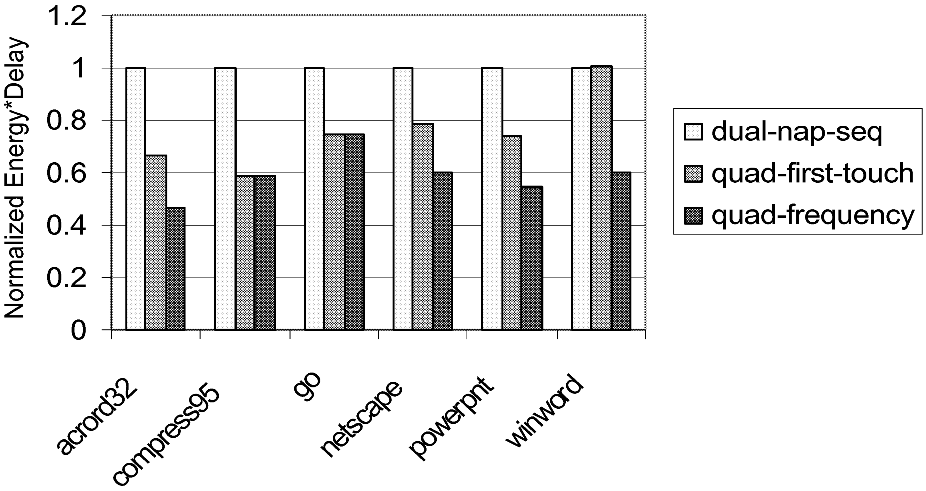
Outline

- Motivation for memory power/energy management and the opportunity
- Hardware power management
- OS page allocation policies
- Experimental results
- Future work, open questions

Better Page Allocation Policies?

- Intuitively, first-touch will not always be best
- Allow movement after first-touch as “corrections”
- Frequency-based allocation
- Preliminary results
  - Offline algorithm: sort by page count
  - Allocate sequentially in decreasing order
  - Packs most frequently accessed pages into first chip
  - Provides insight into potential benefits (if any) of page movement and motivate an on-line algorithm

Frequency vs. First-Touch (NT)

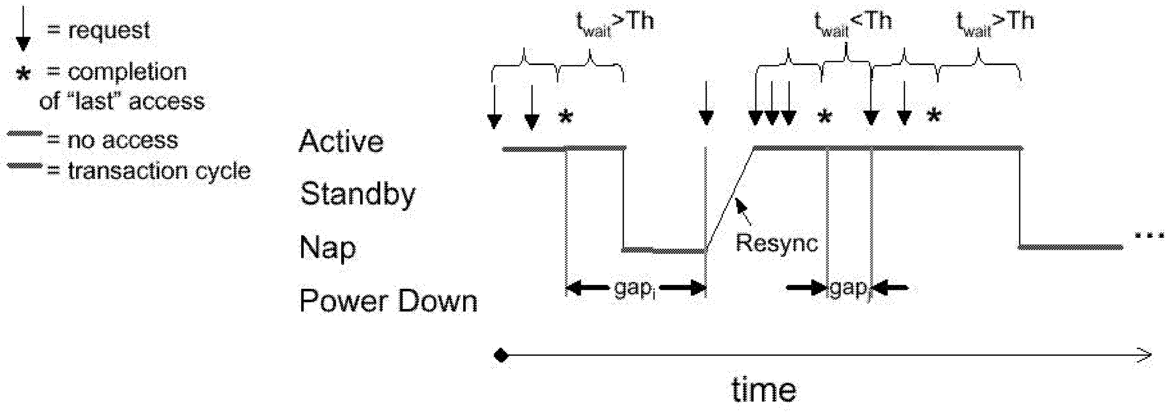


- Base: dual-state nap sequential
- Thresholds: 100 A->N; 5,000 N->PDN
- Opportunity for further improvements beyond first-touch

Hardware Support for Page Movement

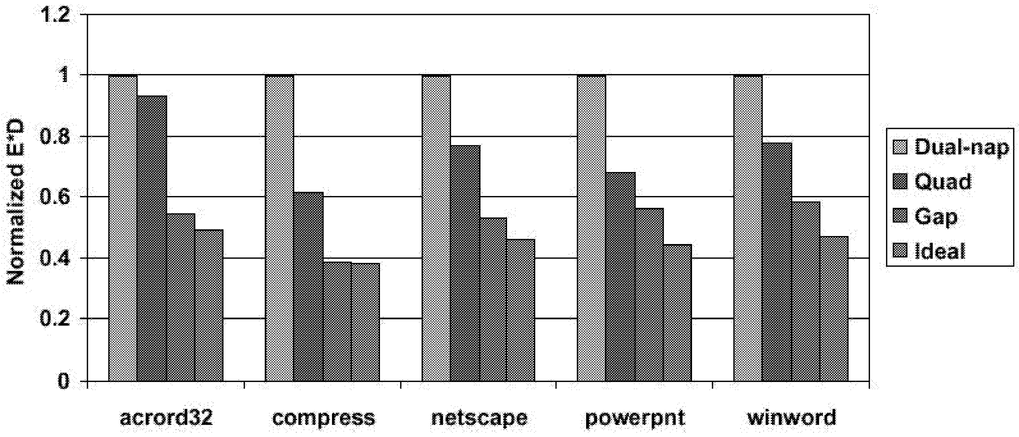
- Data collection hardware
  - Reserve n pages in chip 0 (n=128)
  - 10-bit saturating counter per physical page
- On-line Algorithm
  - Warmup for 100ms, sample accesses for 2ms
  - Sort counts, move 128 most frequent pages to reserved pages in hot chip, repack others into minimum number of chips
- Preliminary experiments and results
  - Use 0.011ms and 0.008mJ for page move
  - 10% improvement for winword
  - Need to consider in execution-driven simulator

Determining Thresholds in Power State Transitions



- If (gap > benefit boundary) threshold = 0 //but gap unknown
- For exponential gap distributions, large average gap, Th = 0 is best //unfortunately, gap distributions are not generally exponential

History-based Prediction in Controller

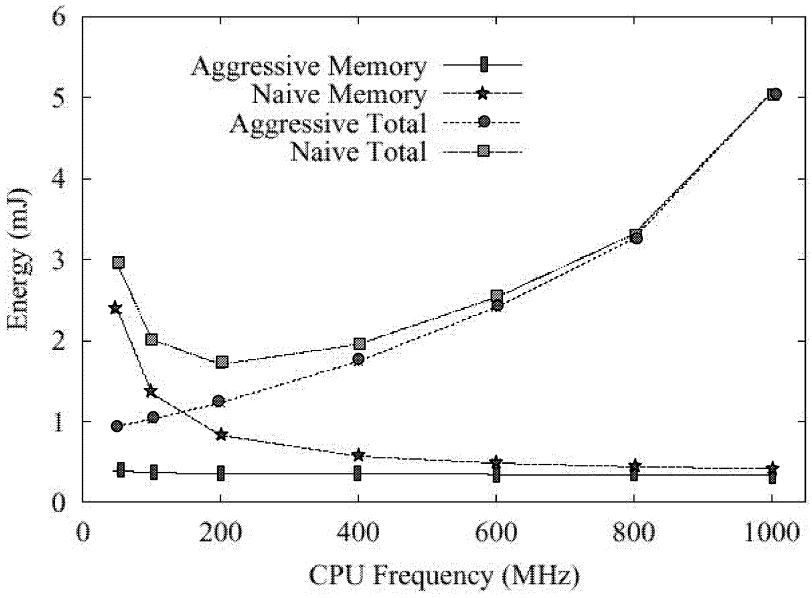


- Sequential page allocation, NT traces
- Ideal: offline policy, delay = all active, minimize power
- Gap policy: History-based prediction
  - If predicted gap > benefit boundary, immediately transition

DVS/PA-Memory Synergy

- With power-aware memory considered, the lowest speed/voltage is not necessarily lowest energy choice.
- Memory access behavior must enter into the speed-setting decision
- The best memory controller policy may depend on the speed setting. Memory controller policy should be adaptive in that case.

Effect of PA Memory on Total Energy with DVS



a) 2% Miss Ratio

Conclusion

- New DRAM technologies provide opportunity
  - Multiple power states
- Simple hardware power mode management is effective
- Effects of operating system page allocation
- Cooperative hardware / software (OS page allocation) solution is best
- Power-aware memory complement DVS

Display Management: Sensing User Intention and Context

Outline

- Motivation and Research Objective
- FaceOff Architecture and Prototype
- Evaluation
  - Best Case Feasibility Study
  - Responsiveness Study
- Future Work

Motivation

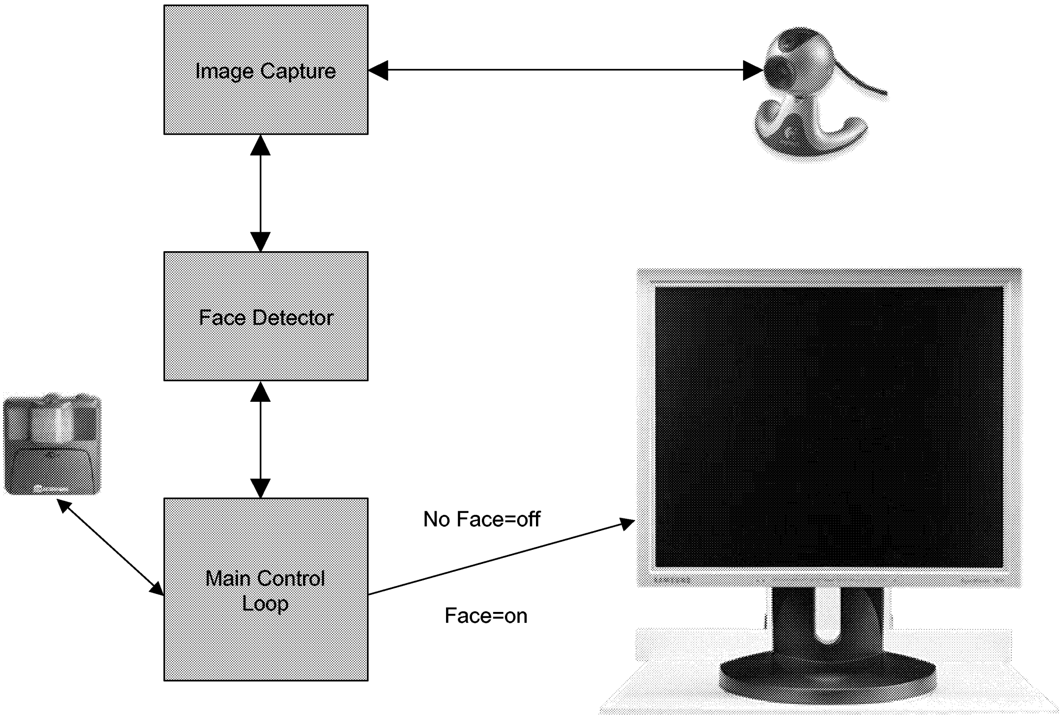
- Current energy management techniques tied to process execution
- Can we use low power sensors to match I/O behavior more directly to user behavior and reduce system energy consumption?

Sensing User Intention and Context for Energy Management

Case Study: FaceOff

- Displays:
  - Typically responsible for large power drain
  - Power State can be controlled by software
  - State transition strategies naïve

A display is only necessary if someone is looking at it.

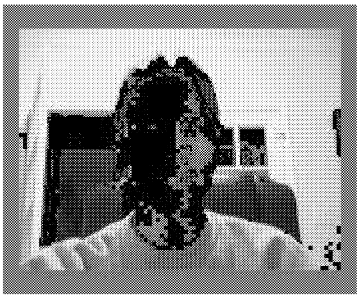


Prototype

- IBM ThinkPad T21 running RedHat Linux
  - Base Power Consumption = 9.6 Watts
  - Max CPU = 8.5 Watts over Base
  - Display = 7.6 Watts
- Logitech QuickCam Web Cam
  - Power Consumption = 1.5 Watts
- Software components:
  - Image capture, face detection, display power state control

Face Detection

- Skin detection used for prototype
- Real time proprietary methods exist



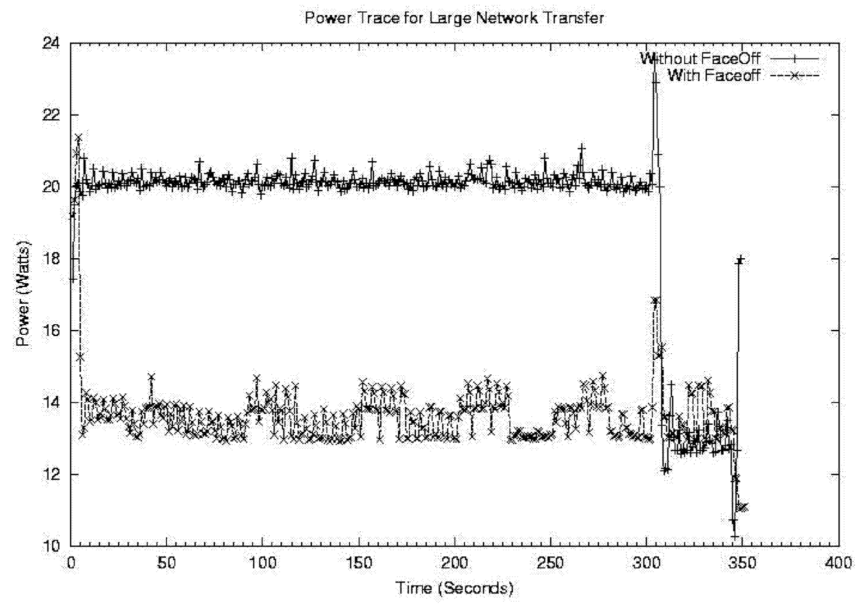
Outline

- Motivation and Research Objective
- FaceOff Architecture and Prototype
- Evaluation
  - Best Case Feasibility Study
  - Responsiveness Study
- Future Work

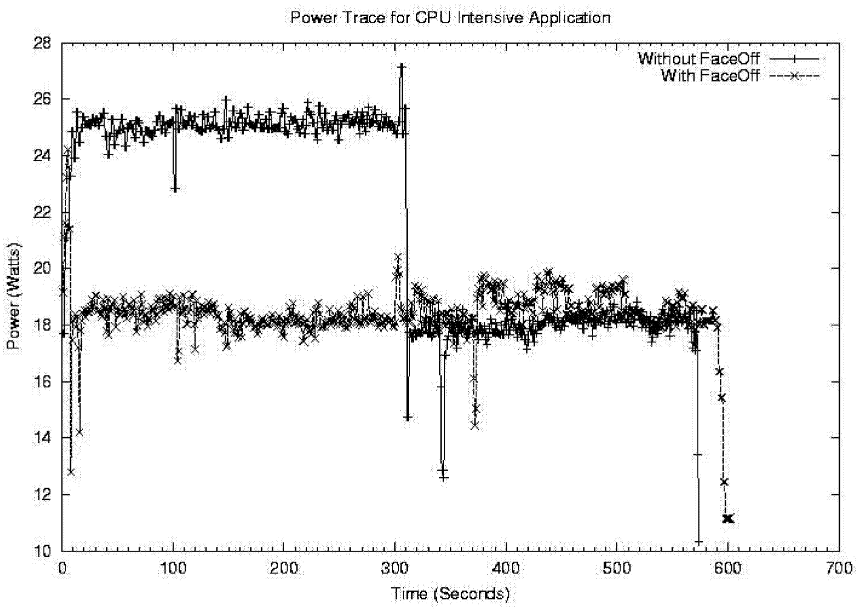
Best Case Feasibility Study

- What is the potential for energy savings?
  - Assume Zero Overhead and Perfect Accuracy
- Tradeoff of energy costs:
  - CPU/Camera vs. Display
- Effect on System Performance
  - Network file transfer (113 MB)
  - CPU intensive process (Linux kernel compile)
  - MP3 Song (no display necessary)

File Transfer Traces



Kernel Compile Traces



Energy and Time Comparisons

Energy (J)	Default	With FaceOff	% Savings
Compile	12506.85	11023.07	11.86
Transfer	6795.42	4791.19	29.49

Time (s)	Default	With FaceOff	% Overhead
Compile	575	603.5	4.96
Transfer	348.6	351.3	0.77

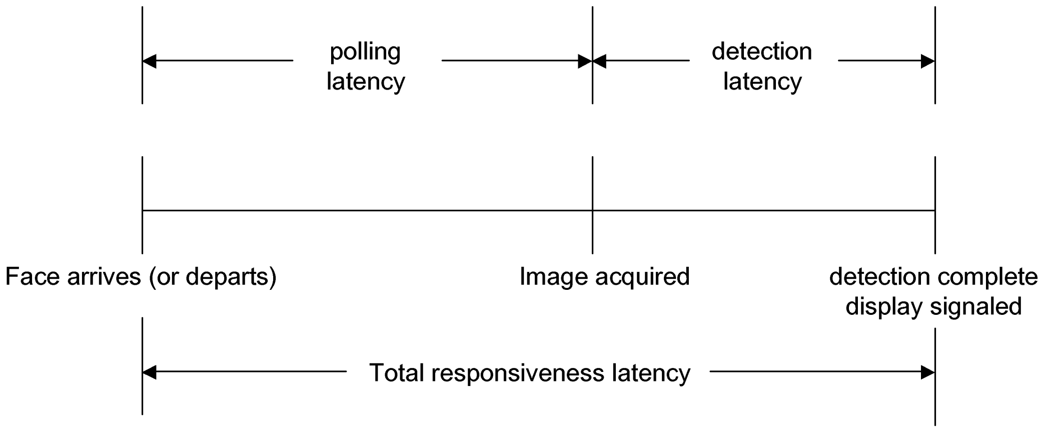
MP3 Application

- Playing an MP3
  - Display not necessary
  - Song completes before default timeout turns off display
- Energy comparison
  - 3,403 J with FaceOff vs. 4,714 J with Default
  - 28% energy savings
- No noticeable effect on playback

Responsiveness Study

- Use full prototype including skin detection
- Establish baseline timing
- Examine Responsiveness
  - varying system load
  - varying polling rate

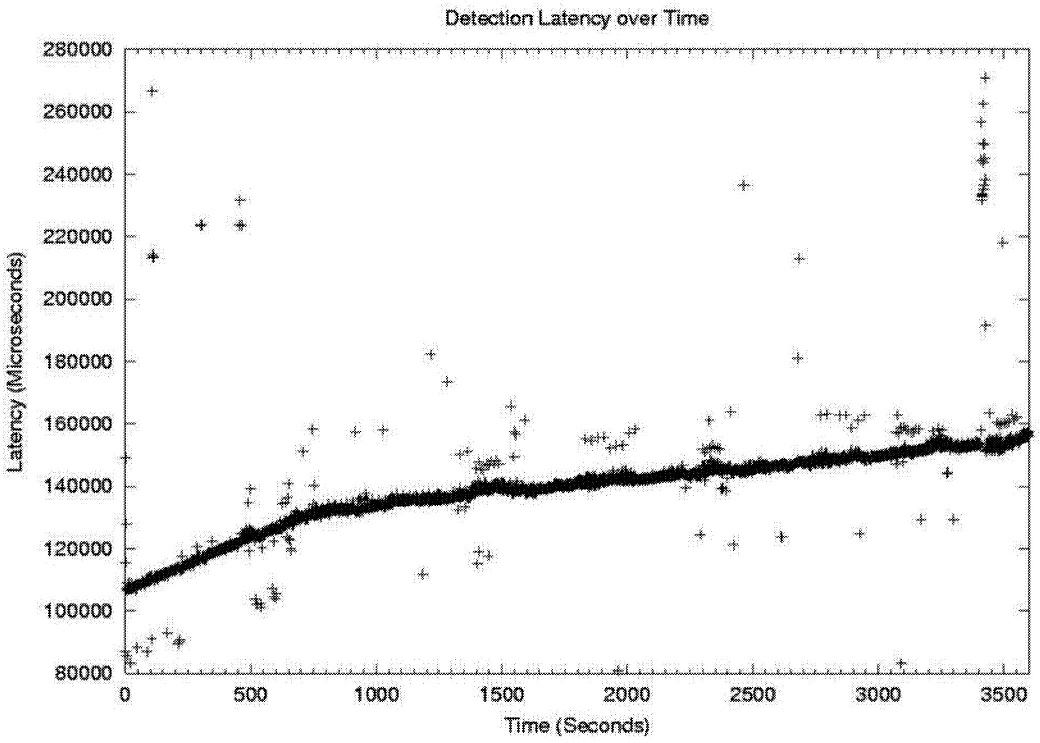
Responsiveness Timing


polling latency
detection latency
Face arrives (or departs)
Image acquired
detection complete display signaled
Total responsiveness latency

Baseline Detection Latency

- Measured over a period of one hour with no programs other than background processes running
- Latency increased over time
  - Started at ~110ms
  - Increased to ~160ms
  - Why?
    - Appears to be an effect of Linux scheduler reducing priority of long running jobs

Detection Latency over Time



Detection Latency Under Load

Workload	Average (99% Confidence)	Maximum	Minimum
Network Transfer	175±7ms	305ms	116ms
Kernel Compile	230±5ms	669ms	51ms
MP3 Song	154±3ms	229ms	84ms

Outline

- Motivation and Research Objective
- FaceOff Architecture and Prototype
- Evaluation
  - Best Case Feasibility Study
  - Responsiveness Study
- Future Work

Varying Polling Rate

- Reduce overhead by reducing polling rate
  - Increases responsiveness latency
- Adaptive polling rate
  - Eliminate polling in presence of UI events
  - Begin polling as duration without UI events increases and face is detected
  - Reduce polling when no face present
    - Similar problem with latency increase upon return

Optimization with Motion Sensor

- Combine adaptive polling & motion sensing
- Meet responsiveness requirements with minimal FaceOff system overhead
- Eliminate image polling when no motion
- Switch display state on immediately when motion detected and restart image polling

Implementation

- Prototype using X10 ActiveHome Wireless Motion Sensor and Receiver
  - Receiver connects to serial port
  - Reading port blocks until sensor triggers
  - Takes up to 10 seconds to recharge
- Promising addition to FaceOff system

More Roles for Sensors

- Touch Sensor
  - Detect picking up of a PDA
- Light, Sound sensors
  - Adjust display brightness (Compaq iPAQ)
  - Adjust speaker volume
- 802.11 Signal Strength sensor
  - Determine possibility of offloading computation

Enhanced Sensors

- "Active Camera"
  - Perform some or all of the face detection
- Color filtering
  - Preprocessing skin color segmentation
- Low Power processor for external sensor control, computation

Discussion: Other ideas for using sensors to save energy?

Related Work

- Display Power Management
  - Industry Specifications
    - APM, ACPI, DPMS
  - Zoned Backlighting
  - Energy-Adaptive Display System Design
- Attentive/Perceptual UIs
  - Smart Kiosk System: Gesture analysis
  - CAMSHIFT: Game control
  - IBM PupilCam: Head gesture recognition

Future Work

- Continue work on optimizing responsiveness
- Comprehensive user study
  - Survey of usability
  - Characterization of usage patterns
    - End-to-end experiment
- Implementation with available very low power camera/motion sensor and prototype for small device (handheld)

Conclusions

- Context information offers promising method of energy management
- FaceOff illustrates feasibility of approach
- Available very low power sensors as well as optimization techniques would improve upon the FaceOff energy savings

Milly Watt Project

People: Angela Dalton, Xiaobo Fan, Alvy Lebeck, Amin Vahdat, Heng Zeng

Emails: carla@cs.duke.edu
{angela,xiaobo,alvy,vahdat,zengh}@cs.duke.edu

Info: http://www.cs.duke.edu/ari/millywatt/

Sponsored by:

ARTES
A network for Real-Time research and graduate Education in Sweden

ARTIST
http://www.artist-embedded.org/

두뇌한국
BK21

CISS
CENTER FOR EMBEDDED SOFTWARE SYSTEMS

MRTC
MÄLARDALEN REAL-TIME
RESEARCH CENTRE