Vertigo: Automatic Performance-Setting for Linux

OSDI `02

Sang-Heon Kim
Min-su Kim

Background(`02)

- **Power management** : from embedded system to server
  - Small size form
  - Battery-based system
  - Ex) Mobile device or PDA

- **Convergence of multiple devices to an integrated device**
  - mp3, mobile, PDA, PMP, camera, Etc…

Problems

- **Low-power processors is needed for battery-operated devices**
- **Power management issue**
  - Variable performance requirements of tasks
    - High performance : Video player
    - Low performance : MP3 audio
  - Dynamic power-performance mode
  - How to calculate performance level accurately in real time?
    - Performance-setting algorithm with model
- **Intel's Sidestep (Usage model)**
  - Plugged in : full-active performance
  - On Battery : efficient performance

Purpose

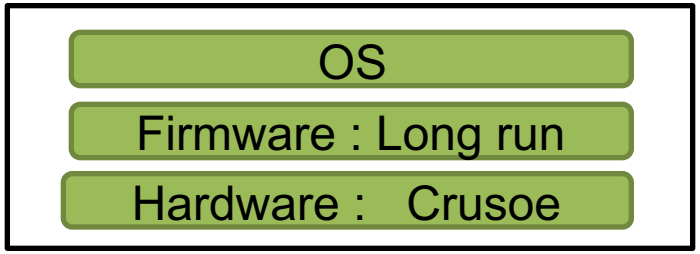
- **Performance–settings based on CPU demand**
  - Using DVFS(Dynamic Voltage Frequency Scaling) technique
  - Dynamic power allocation

Least power consumption,
while users feel **no performance degradation**

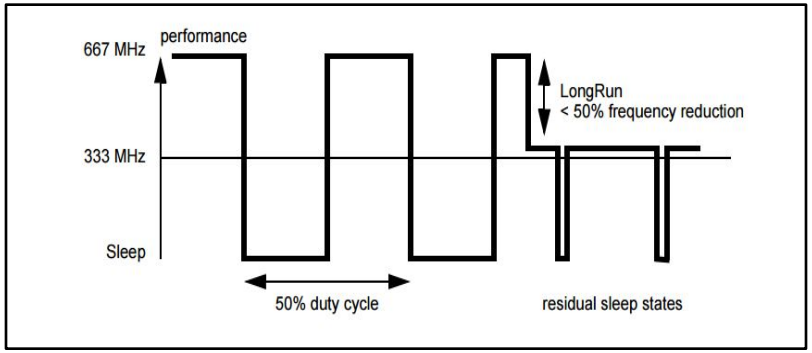
  - Implementation – Where?
    - Kernel (Vertigo) vs Hardware (Long run)

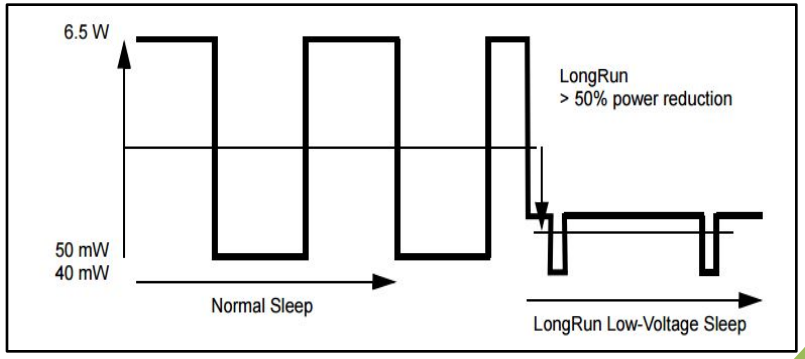
LongRun

- **Hardware & firmware level performance decision**
  - Kernel-independent Power Management



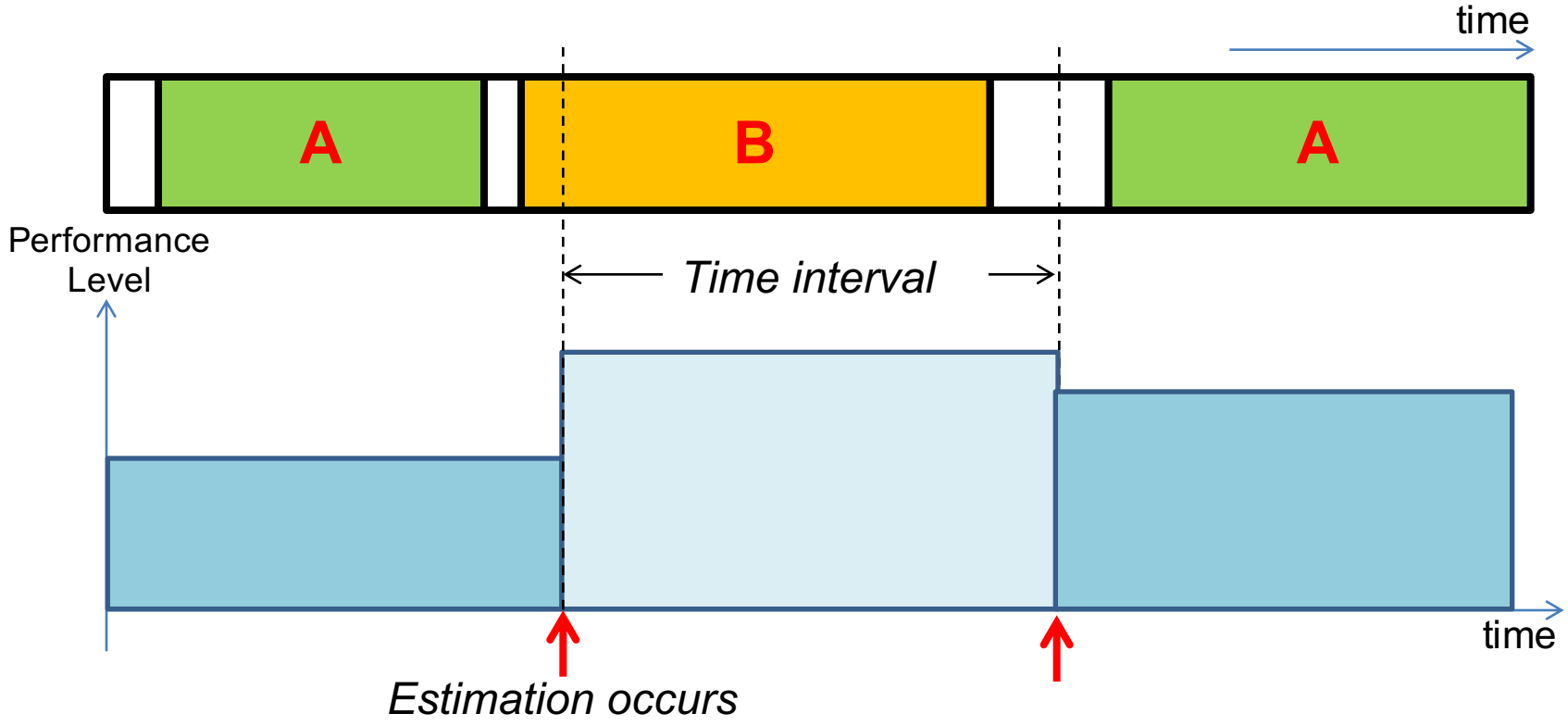
- **Interval based performance management**
  - CPU utilization(duty cycle)





LongRun

- **Interval-based utilization estimate**



➢ *CPU utilization high ⇒ Speed-up*
low ⇒ Speed-down

LongRun

- **Problems**
  - Non-aware to kernel information : application, scenario
    - Cannot optimize to task characteristics
    - Lack of response to task switching
    - Difficult to deal with certain kinds of run-time situations

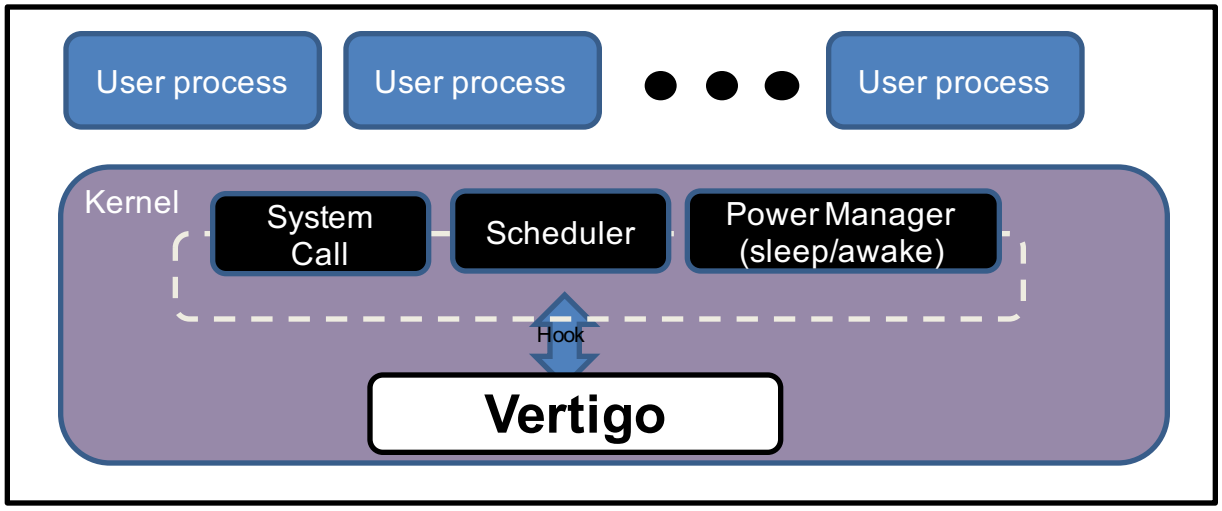
      (e.g - mouse moves, interactive applications)
  - Fixed monitoring interval
    - How long?
      - Too short : oscillated performance level
      - Too long : hard to address performance transition (interactive)

Vertigo

- **Key Contribution**
  - Implemented in OS kernel
    - Gives access to a richer set of data for prediction
    - Capability of response to performance requirement
  - Multiple Performance-setting Algorithms
    - Guarantee deadline, especially interactive applications
    - Per task performance prediction Algorithm

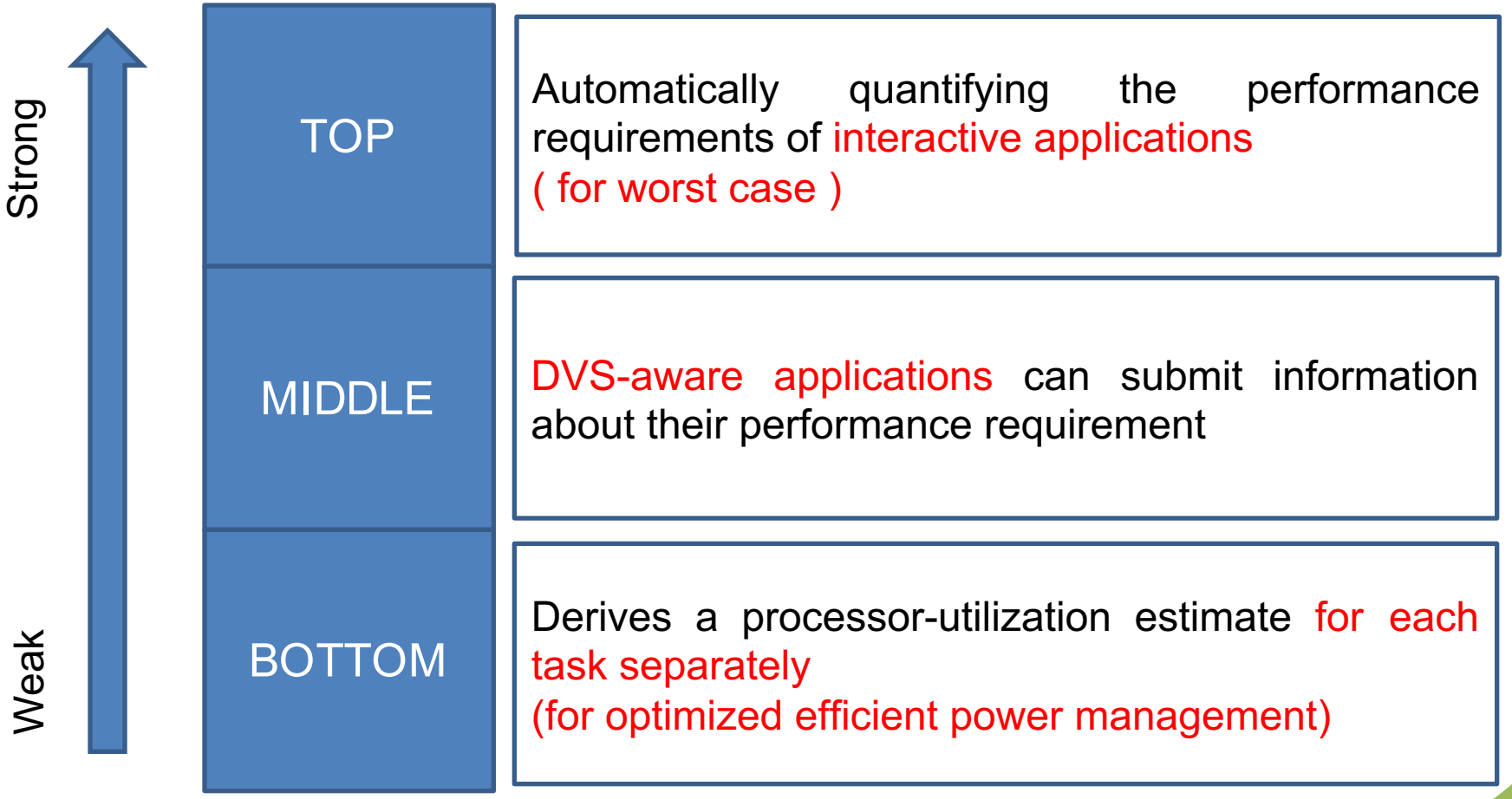
Vertigo

- **Architecture**
  - Vertigo hooks previous Linux kernel
  - Vertigo can access process information
    - System Call : task scenario
    - Scheduler : task identification
    - Power Manager : CPU utilization



Vertigo

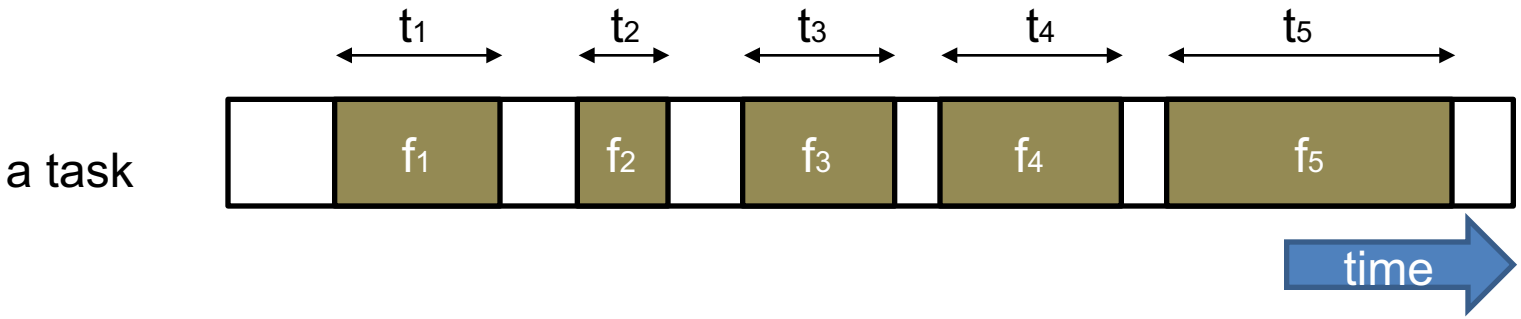
- **Multiple Performance-setting Algorithm**



Vertigo

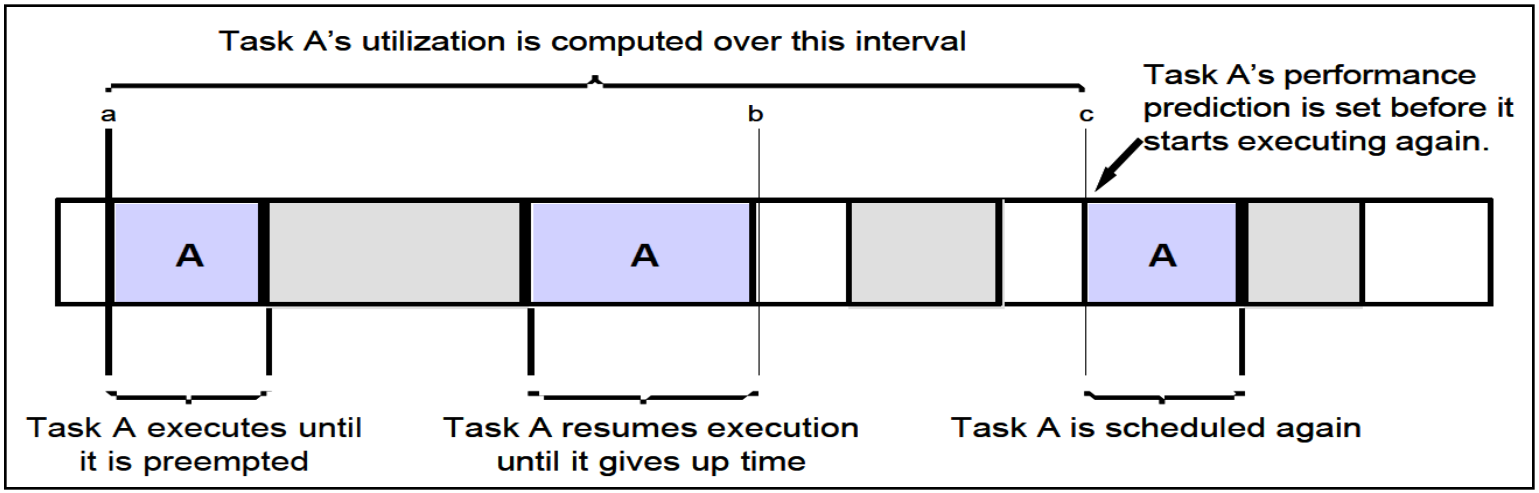
- **Workload model**
  - Full-speed equivalent work = # of cycles

$$Work_{fse} = \sum_{i=1}^{n} t_i f_i$$



Vertigo

- **Per-task workload monitor**
  - When a task starts execution, the per-task data structures are initialized with four pieces of information
    - Work time counter
    - Idle time counter
    - The current time
    - A *run bit* indicating that the task has started running
  - Interval ends with quantum expires or system calls



Vertigo

- **Bottom level performance-setting algorithm**

  : A perspectives-based algorithm

  - Derives a utilization estimate per each task separately
  - No fixed interval → event-driven interval (quantum expires or system call)
  - Workload accumulated by exponentially decaying averages

    - Workload estimation

$$WorkEst_{new} = \frac{k \times WorkEst_{old} + Work_{fse}}{k+1}$$

    - Deadline

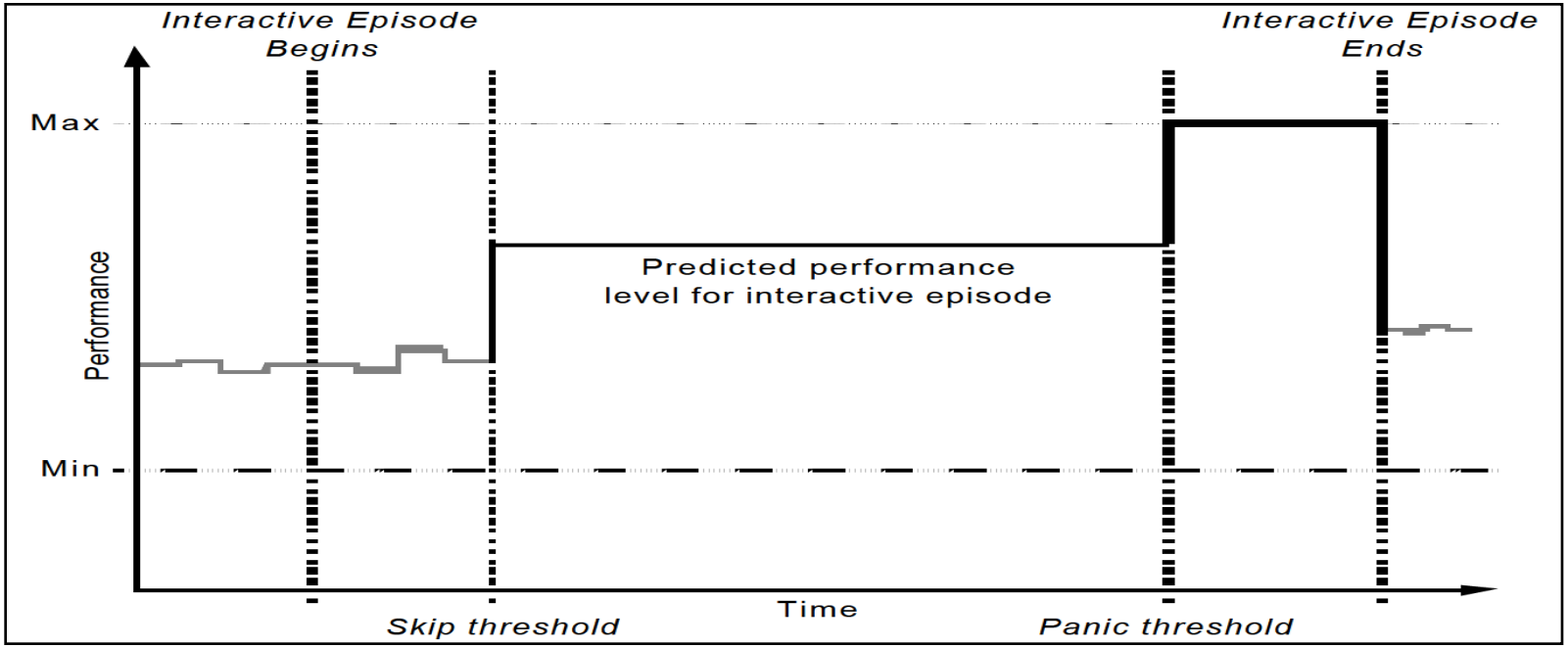
$$Deadline_{new} = \frac{k \times Deadline_{old} + Work_{fse} + Idle}{k+1}$$

    - Require performance

$$Perf = \frac{WorkEst}{Deadline}$$

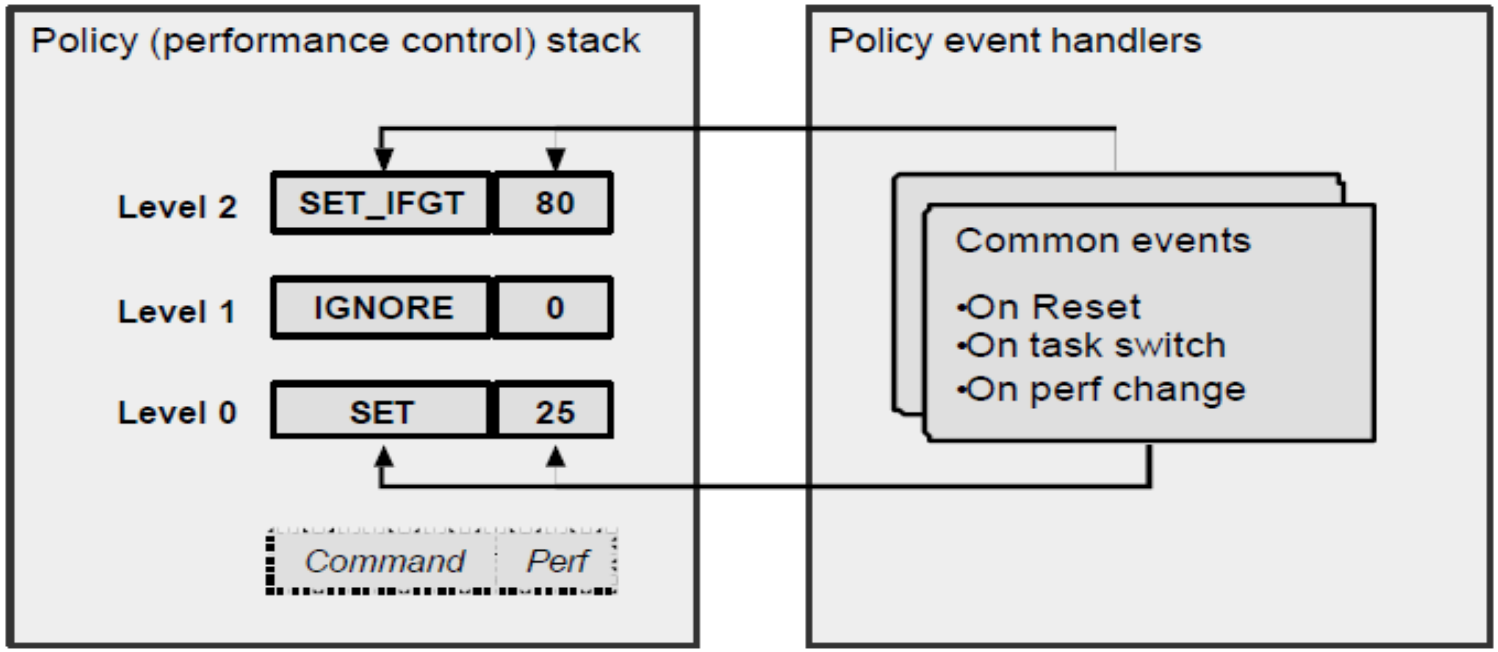
Vertigo

- **Top level performance stetting algorithm**
  - By monitoring the system calls, Vertigo can detect interactive episodes.
    - Mobile target : end-user response time is important
  - Be able to guarantee deadlines



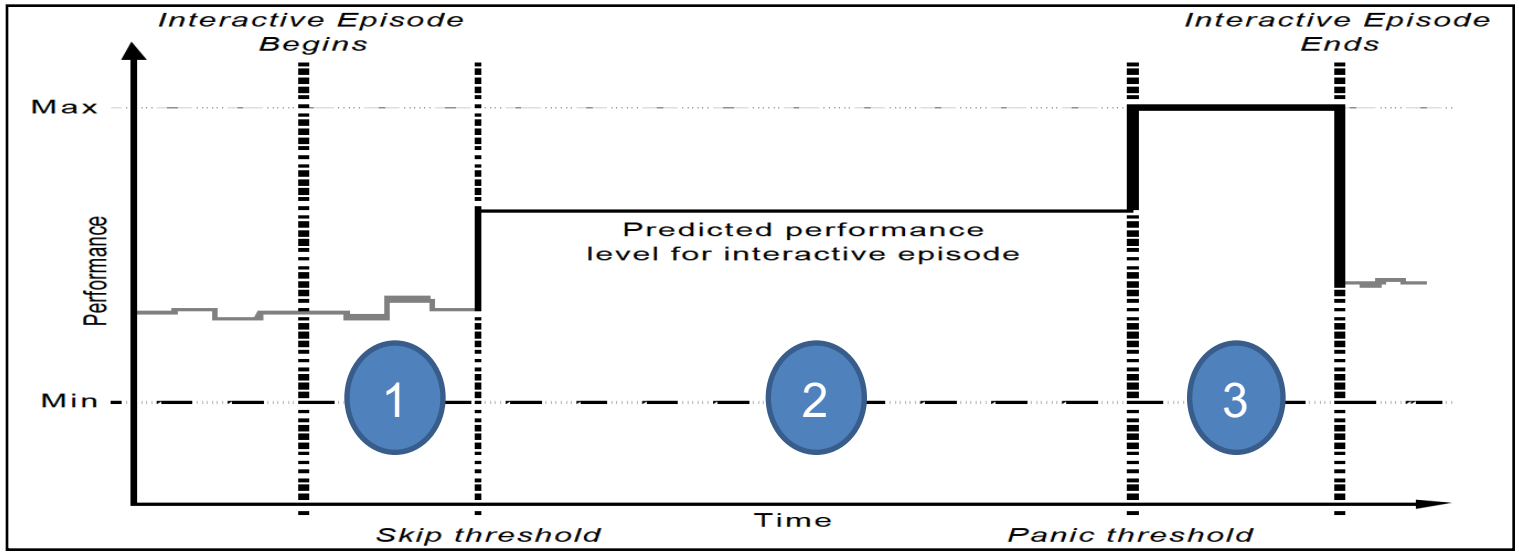
Vertigo

- **Policy stack implementation**
  - Can override lower algorithm policy
  - Kernel event-aware performance-setting



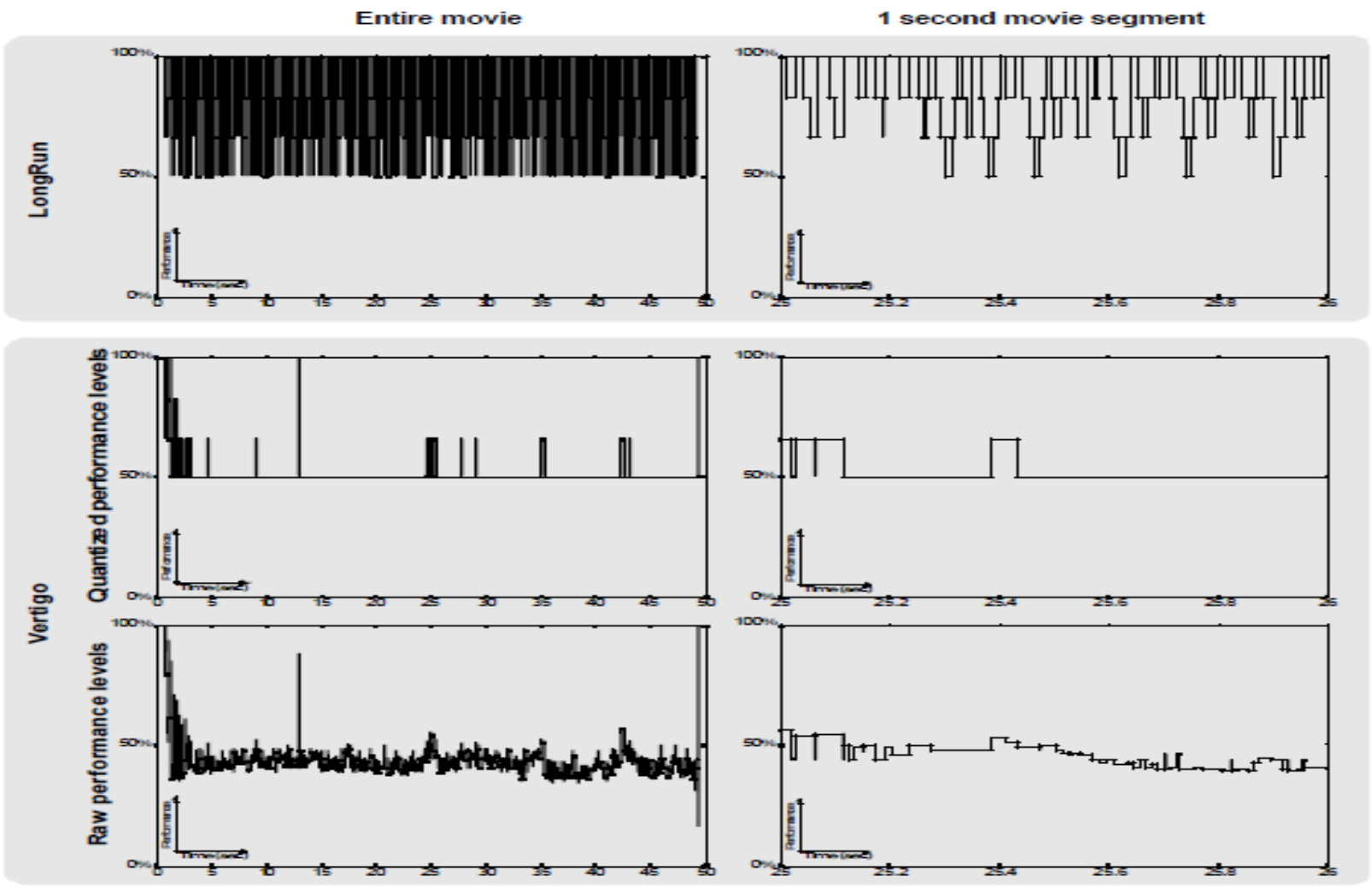
Vertigo

- **Top level performance stetting algorithm**
  1. ***Beginning ~ Skip threshold***
     - Short time routine episodes
  2. ***Skip-threshold ~ Panic threshold***
     - Assign expected performance level by cumulated history
  3. ***Panic threshold***
     - Prediction failure occurs
     - Shift to the maximum performance level
     - Compensate for future triggered event



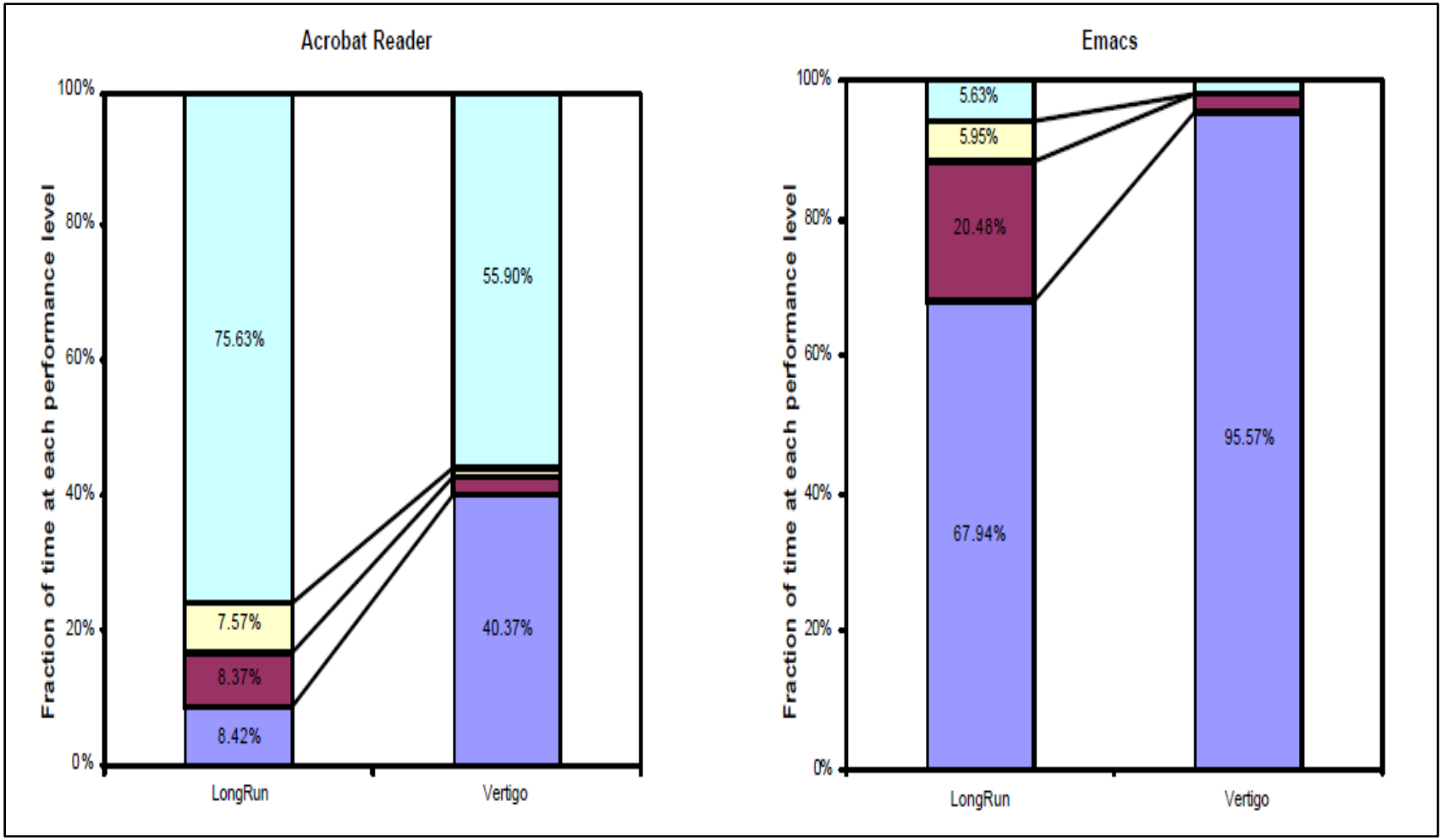
Evaluation

- **MPEG scenario**



Evaluation

- **Interactive applications**



Summary

- **Vertigo**
  - Initial in-kernel level trial to control DVFS
  - Per task performance-setting algorithm
  - Guarantee deadlines for interactive application

- **Power management for Mobile target device**
  - Responsibility : user-interactive application

- **Impact of Vertigo on present OS's power managements**
  - Difficult to implement Vertigo's full functions
  - Vertigo's top level algorithms is useful only for applications that occur interactive episodes frequently
  - Android / Linux ?
    - Aggressive power management is only active when application requires. (interactive episodes)
    - Use "Wakelock" API for power control in Android

Power containers: an OS facility for fine-grained power and energy management on multicore servers

ASPLOS `13

Sang-Heon Kim
Min-su Kim

Background

- **New generation computing systems appearance**
  - Data center / Server systems
  - Online applications :
    - Client-directed applications
    - Rely on clients to supply content
  - High throughput capability is important
  - Quality of Service (Guarantee performance per client's policy)

- **Power management is more important**
  - Core utilization / Shared resource
  - Heterogeneous platform

Problems

- **Problems of Multicore / Server systems**
  - Work load diversity
    - Large power fluctuation
    - Hardware resource sharing
  - Previous approach : Using CPU utilization history
  - Uncore component (cache, memory interconnect)
    - Cause “power viruses”
    - Concurrent execution

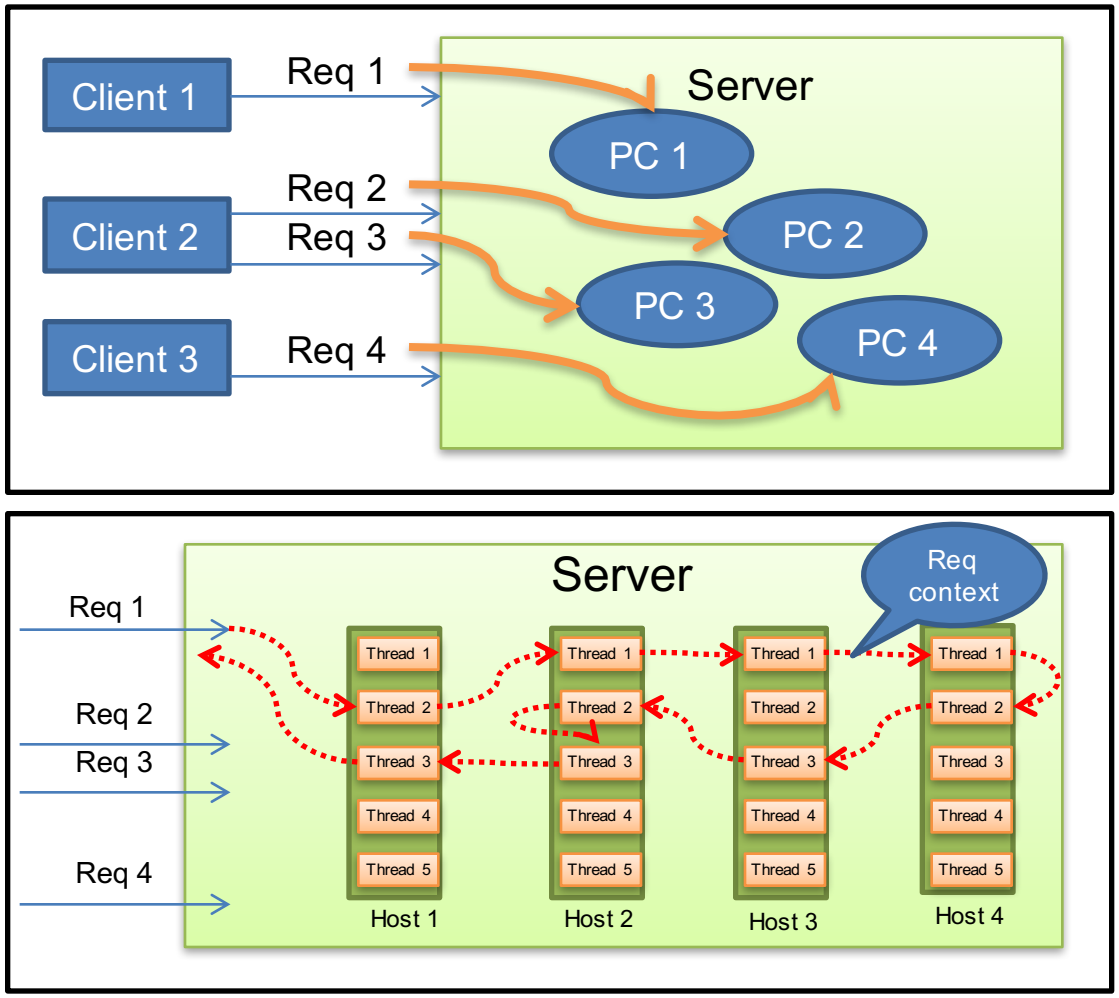
➢ **Per-client/request power management is highly desirable**

➢ **Isolating per-client power attribution**

➢ **Recognizing the energy usage of individual requests**

Power container

- **System overview**



Power container

- **Account for and control the power and energy usage of individual requests in multicore servers**

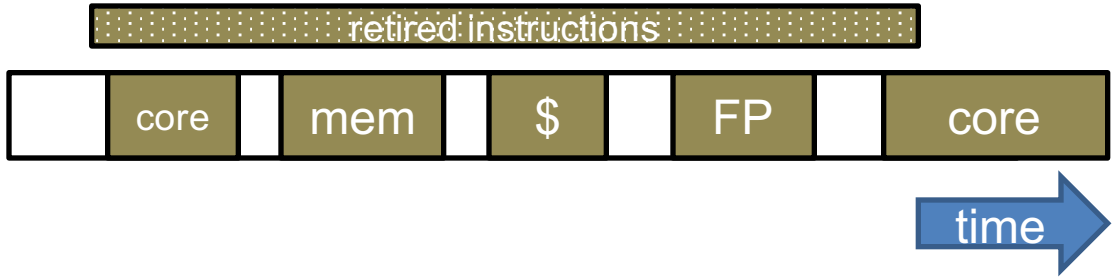
- **Per request power modeling**
  - Aware uncore component's power model
  - For better recalibration, adopt online power measurement

- **Request context-aware power management**
  - Request tracking in multi-stage server

Power Attribution to Tasks

- **Power consumption model**
  - Hardware counter monitor workload per cycle
    - **Core utilization** per elapsed cycles
    - **Retired instructions** per CPU cycle
    - **Floating point operations** per cycle
    - Etc…
  - Event-based power accounting
    - Hardware counter : periodic counter sampling
    - Computing relevant event frequencies
  - Cover uncore component's power consumption
  - Can apply to both **entire system** end **specific tasks**

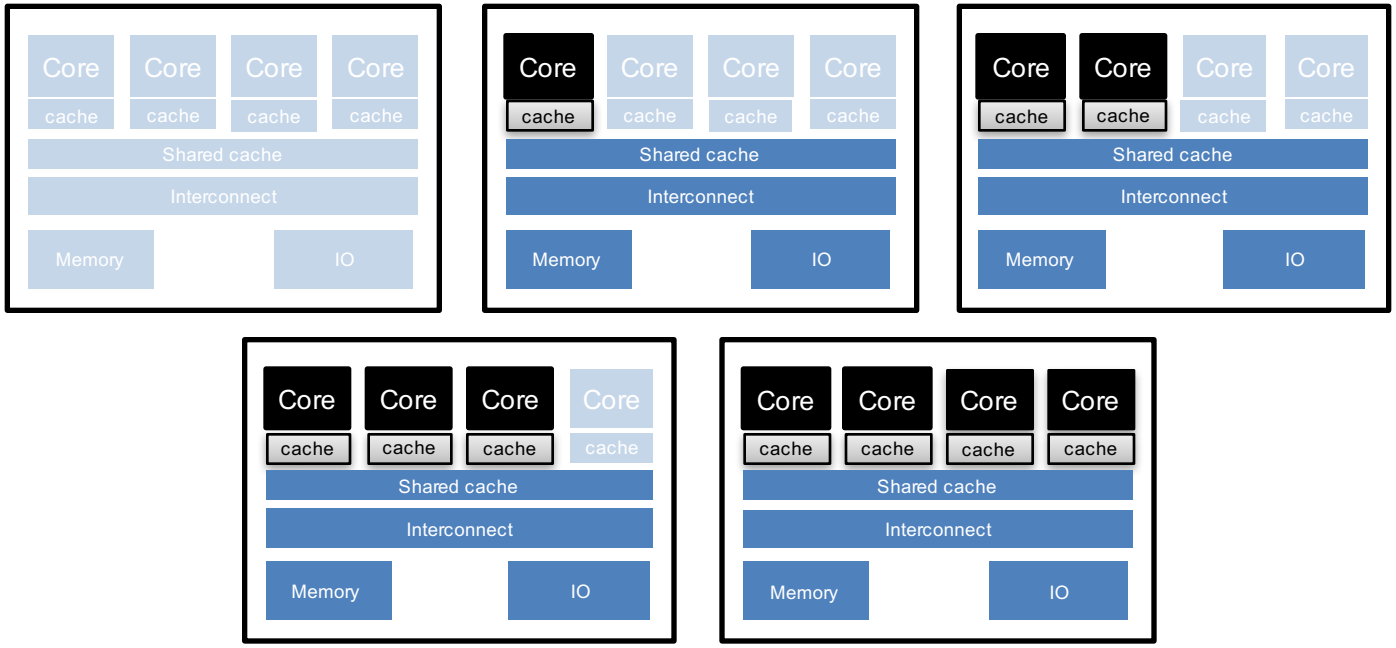
$$P_{active} = C_{core} \cdot M_{core} + C_{fp} \cdot M_{fp} + C_{mem} \cdot M_{mem} + \cdots$$



Power Attribution to Tasks

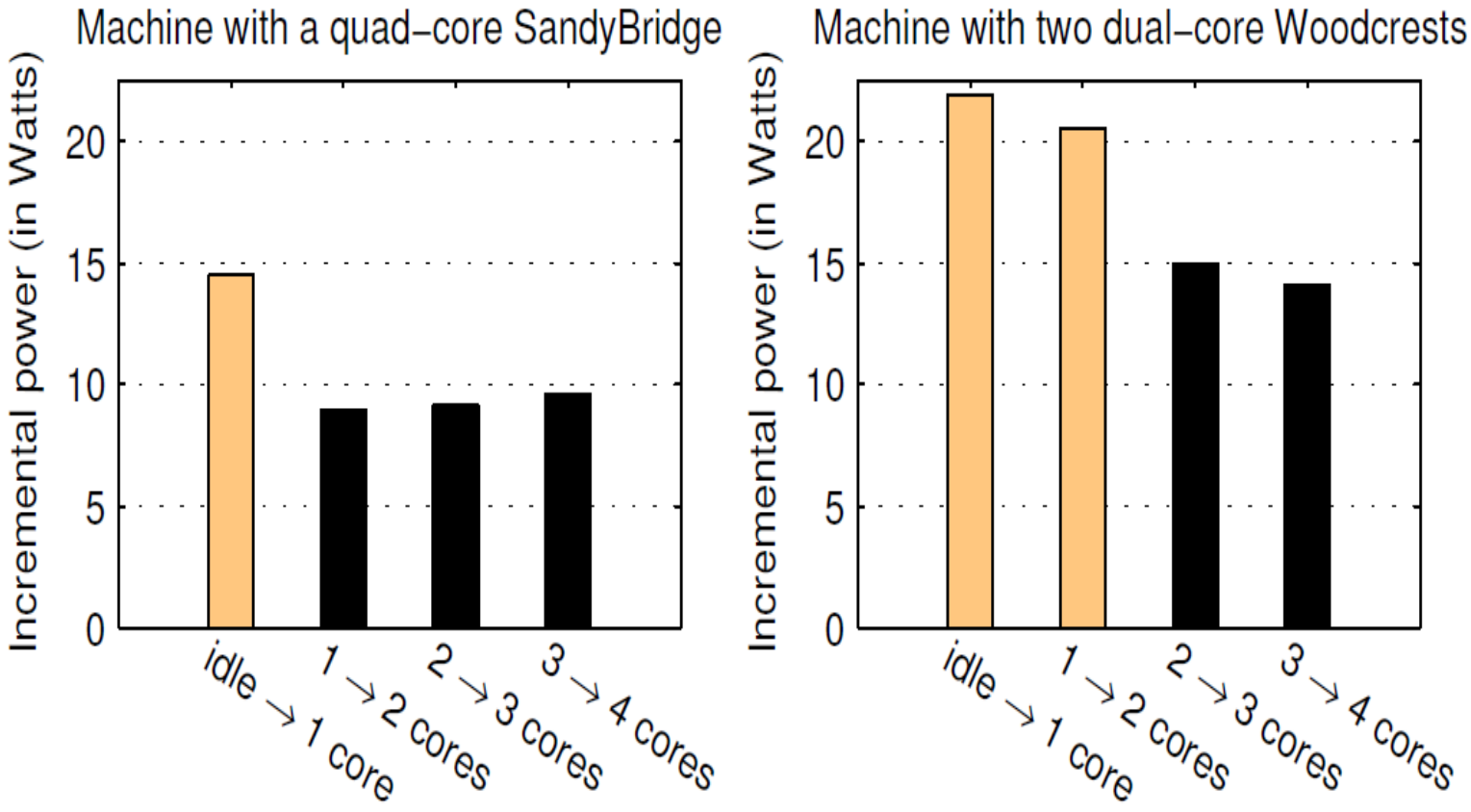
- **Multicores power consumption model**
  - Power consumption : not proportional to # of utilized CPU
  - Shared resource power consumption model

$$P_{active} = P_{single} + C_{\text{chip share}} \cdot M_{\text{chip share}}$$



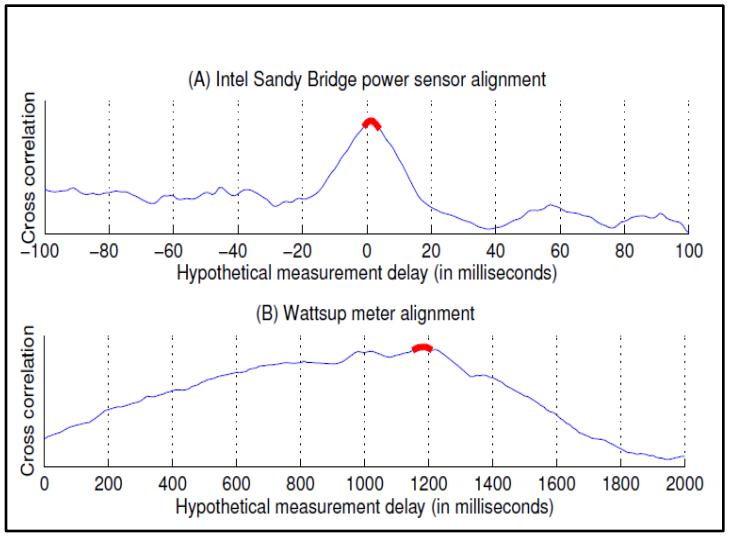
Power Attribution to Tasks

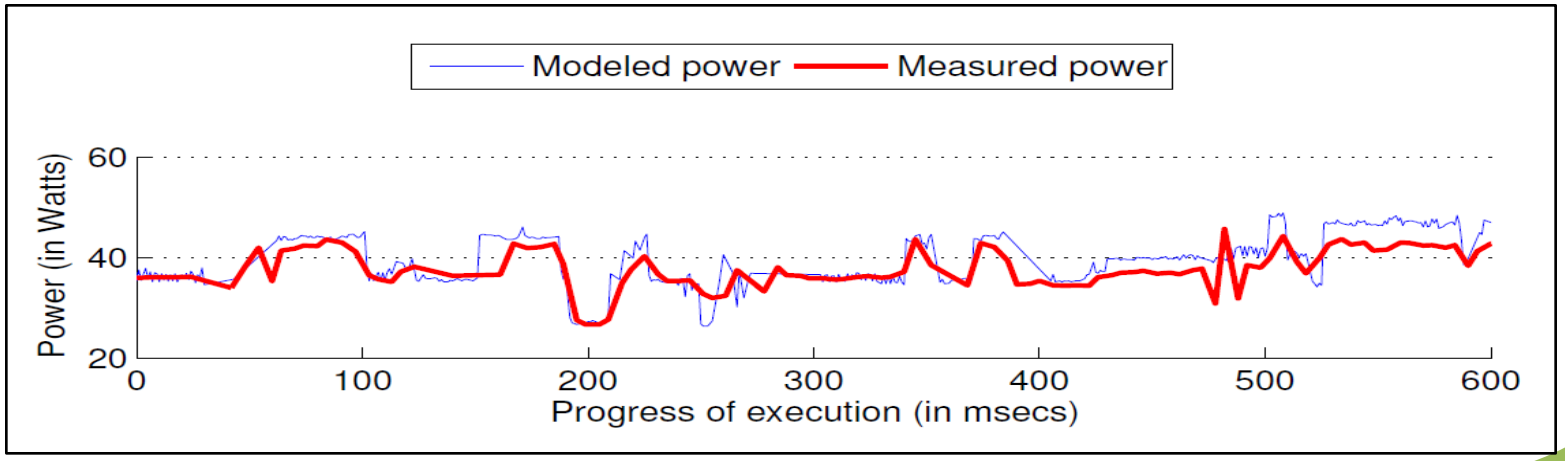
- **Multicores power consumption model**



Recalibration & power measurement

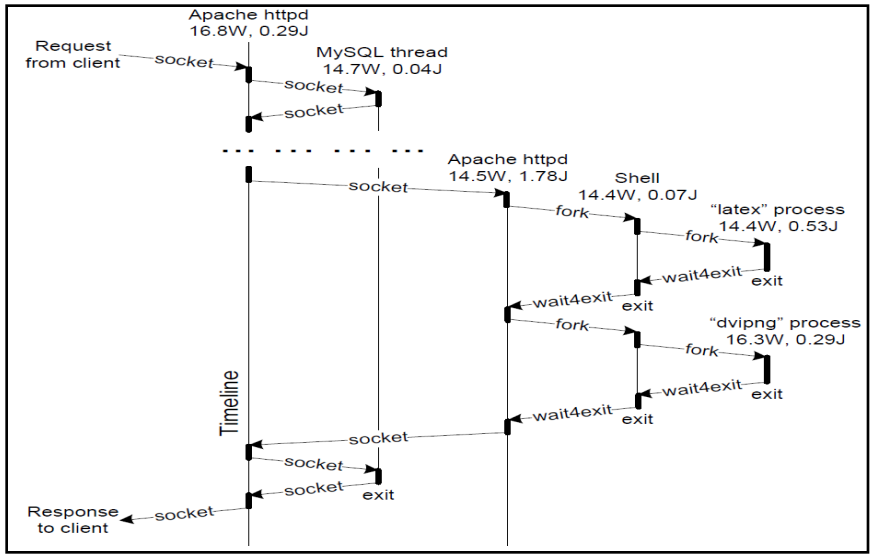
- **Compare power model to measurement**
  - Model
    - Some inaccuracy
    - Good prediction of power transition
    - Can be immediately applied
  - Measurement
    - Lag time : I/O transfer time

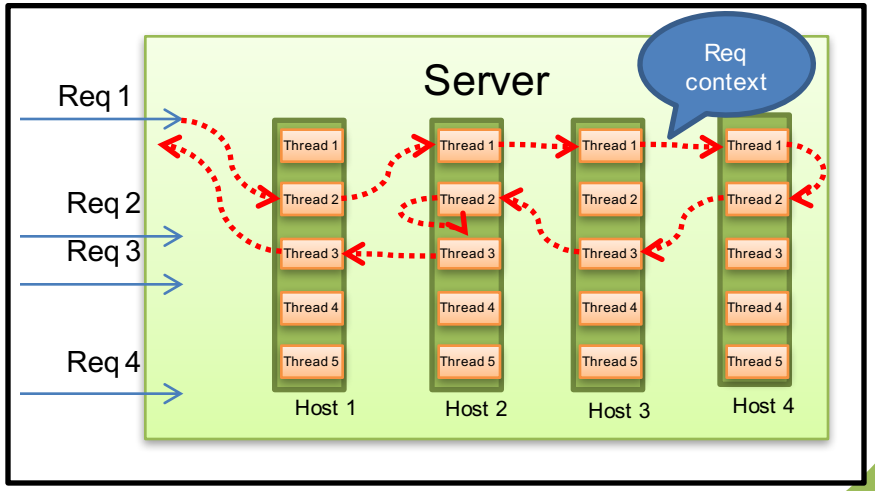




Request tracking

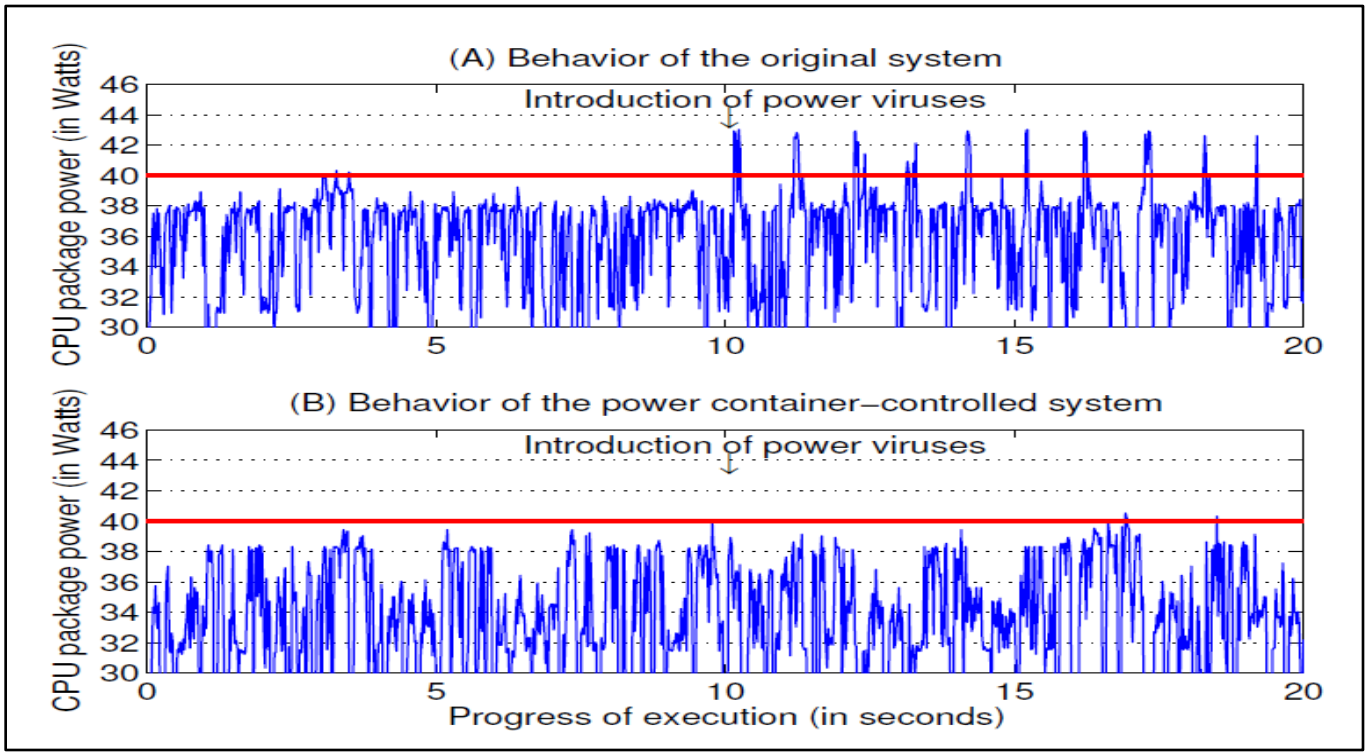
- **Request execution may flow through multiple processes in a multi-stage server**
- **Request context transfer**
  - Event-driven at kernel (sockets,fork….)
  - Application transparency by recognizing key request propagation channel
- **Support request tracking over a persistent socket connection – with request tag**





Container-enabled management

- **Fair request power conditioning**
  - Request power accounting can detect **power spikes(power virus)**
  - Container-specific power control can precisely throttle **execution of power-hungry requests**



Container-enabled management

- **Heterogeneity-aware request distribution**
  - Load placement and distribution on available machines may affect the system energy efficiency
  - Enable the preferential placement of each request on a machine where its relative energy efficiency is high
- **Information about request execution control**
  - Tagging request messages **to next machine**

    → container identifier and control policy settings – application transparency
  - Tagging response messages **to previous machine**

    → cumulative power and energy usage information – for heterogeneity-aware

Overhead

- **Container maintenance operation**
  - Reading the hardware counter values
    - Computing modeled power values,
    - Updating request statistics
  - (quad-core Sandy-Bridge) 0.95 us per (1ms>)
    => (0.1% overhead)

- **Power measurement alignment and model recalibration**
  - 16 us per 10ms

Evaluation

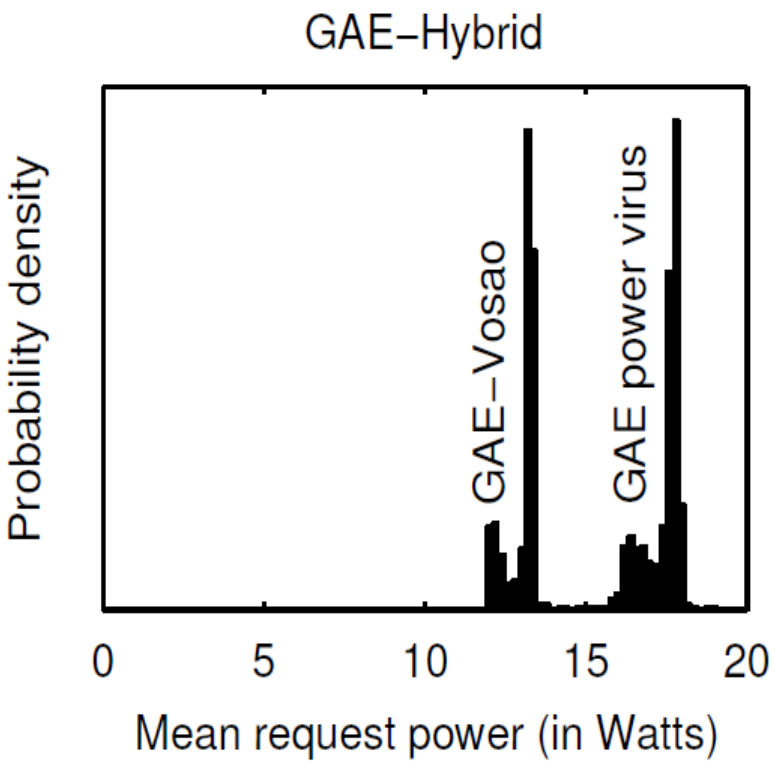
- **Power model calibration**
  - Power model coefficient decision by Benchmark

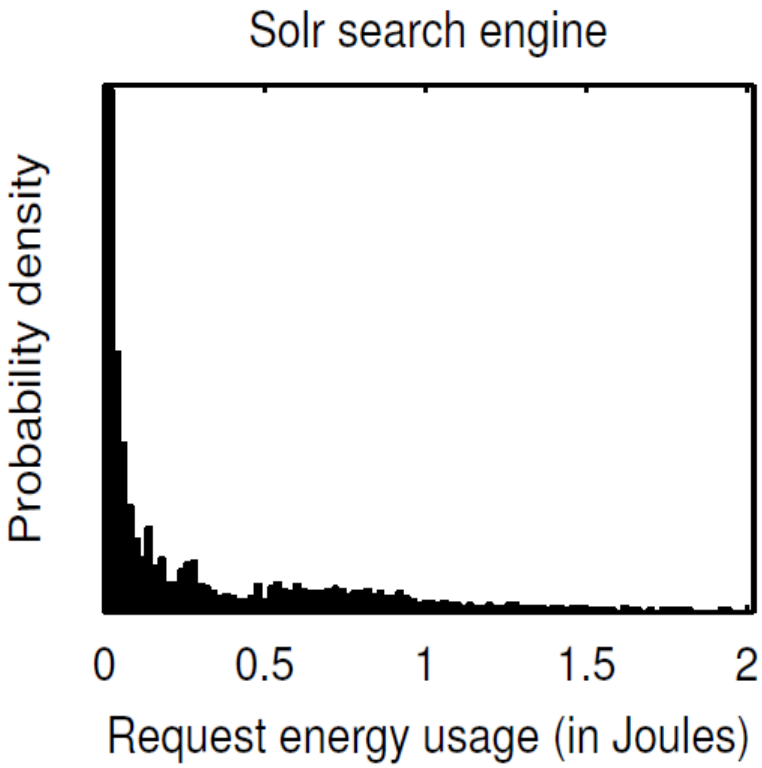
$$
\begin{aligned}
\mathcal{C}_{\text{idle}} &= 26.1\ \text{Watts};\\
\mathcal{C}_{\text{core}} \cdot \mathcal{M}^{\max}_{\text{core}} &= 33.1\ \text{Watts};\\
\mathcal{C}_{\text{ins}} \cdot \mathcal{M}^{\max}_{\text{ins}} &= 12.4\ \text{Watts};\\
\mathcal{C}_{\text{cache}} \cdot \mathcal{M}^{\max}_{\text{cache}} &= 13.9\ \text{Watts};\\
\mathcal{C}_{\text{mem}} \cdot \mathcal{M}^{\max}_{\text{mem}} &= 8.2\ \text{Watts};\\
\mathcal{C}_{\text{chipshare}} \cdot \mathcal{M}^{\max}_{\text{chipshare}} &= 5.6\ \text{Watts};\\
\mathcal{C}_{\text{disk}} \cdot \mathcal{M}^{\max}_{\text{disk}} &= 1.7\ \text{Watts};\\
\mathcal{C}_{\text{net}} \cdot \mathcal{M}^{\max}_{\text{net}} &= 5.8\ \text{Watts}.
\end{aligned}
$$

Uncore component's impact on entire power consumption

Evaluation

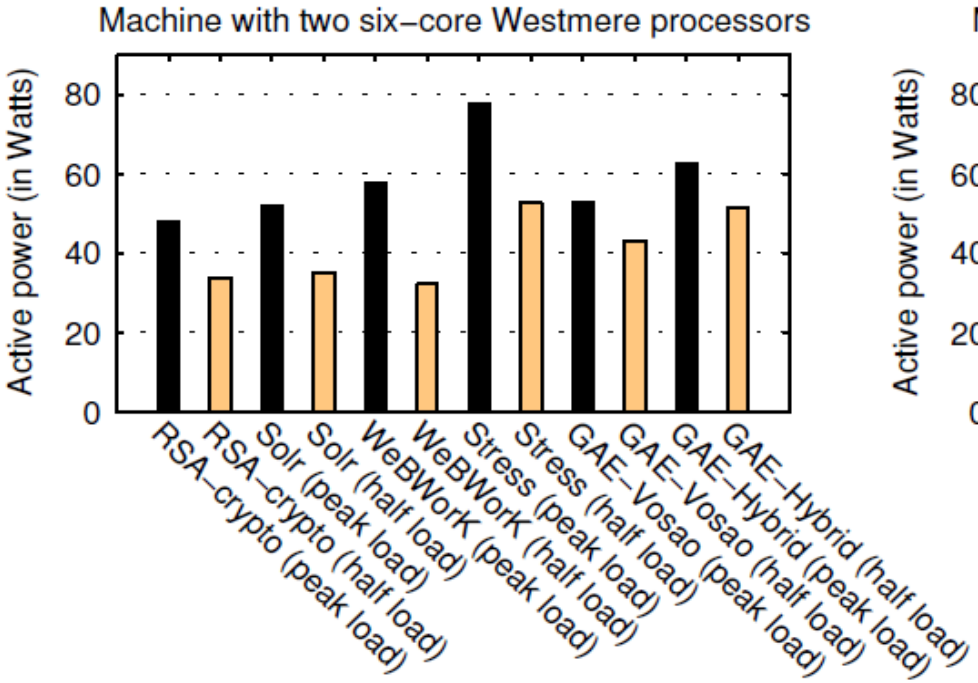
- **Accuracy of power prediction of Power container**

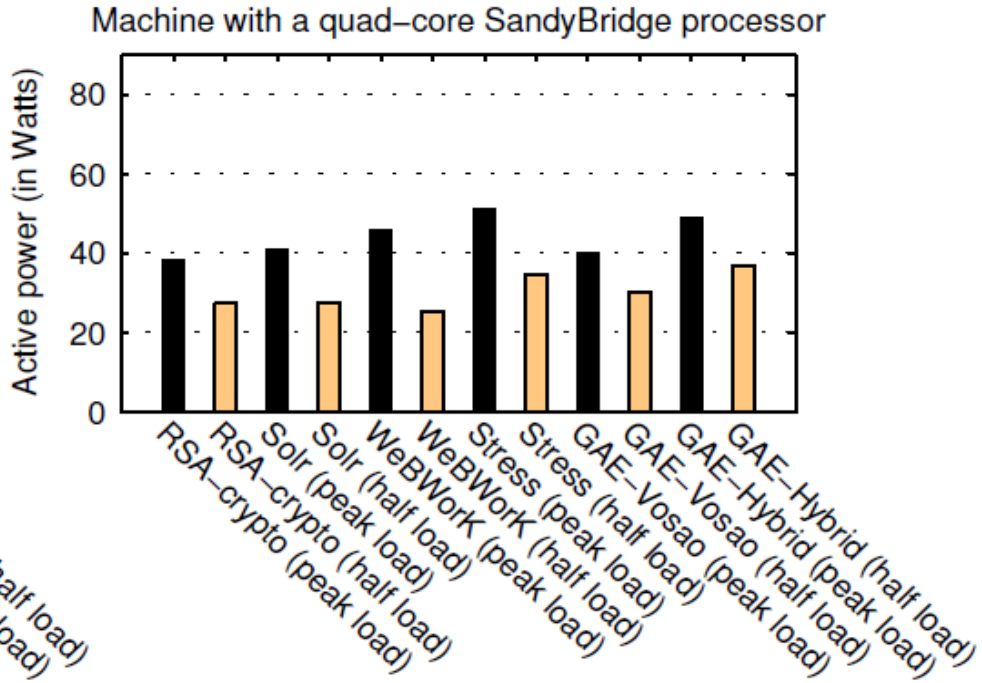




Evaluation

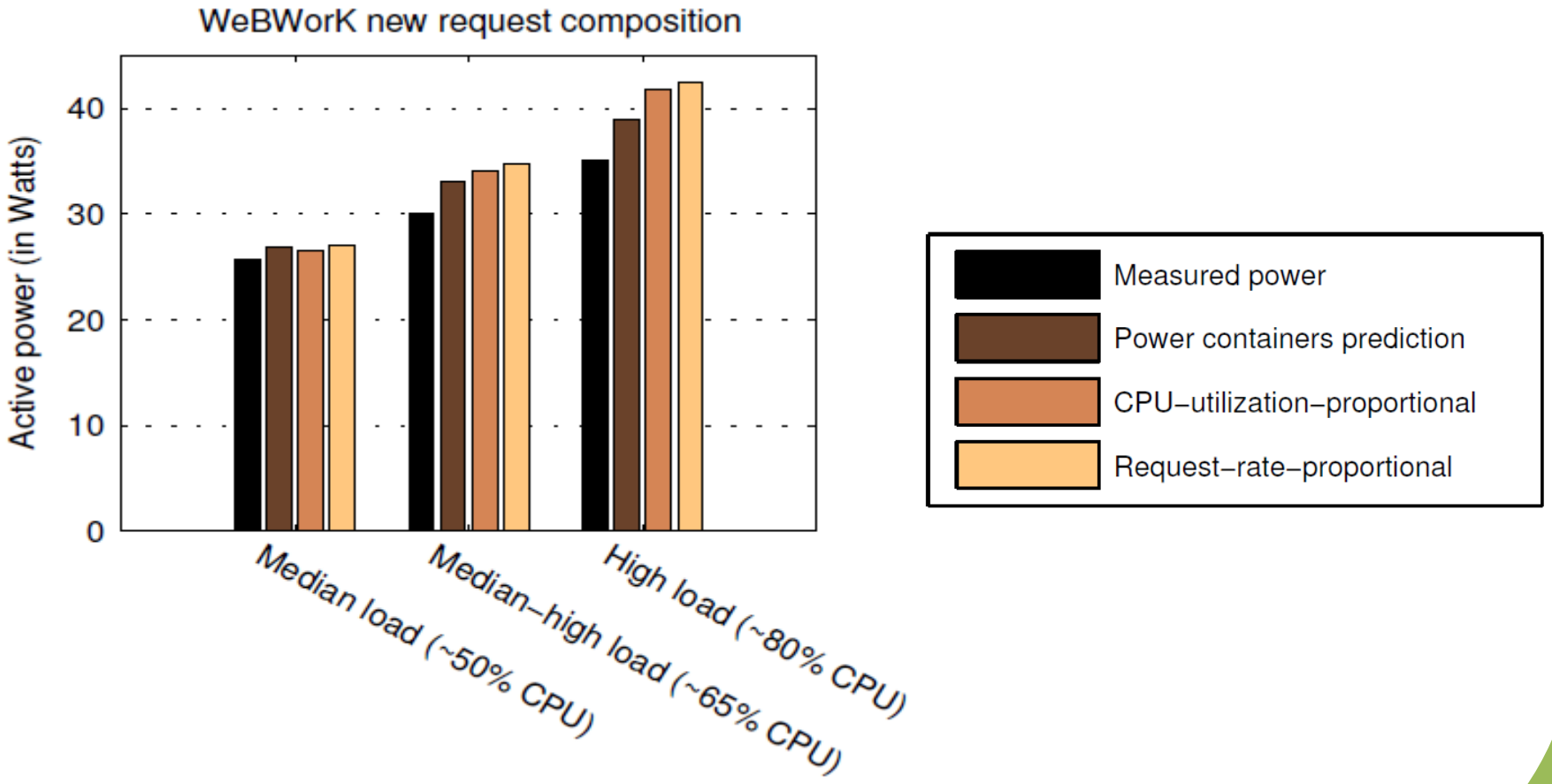
- **Measured active power of application workloads**





Evaluation

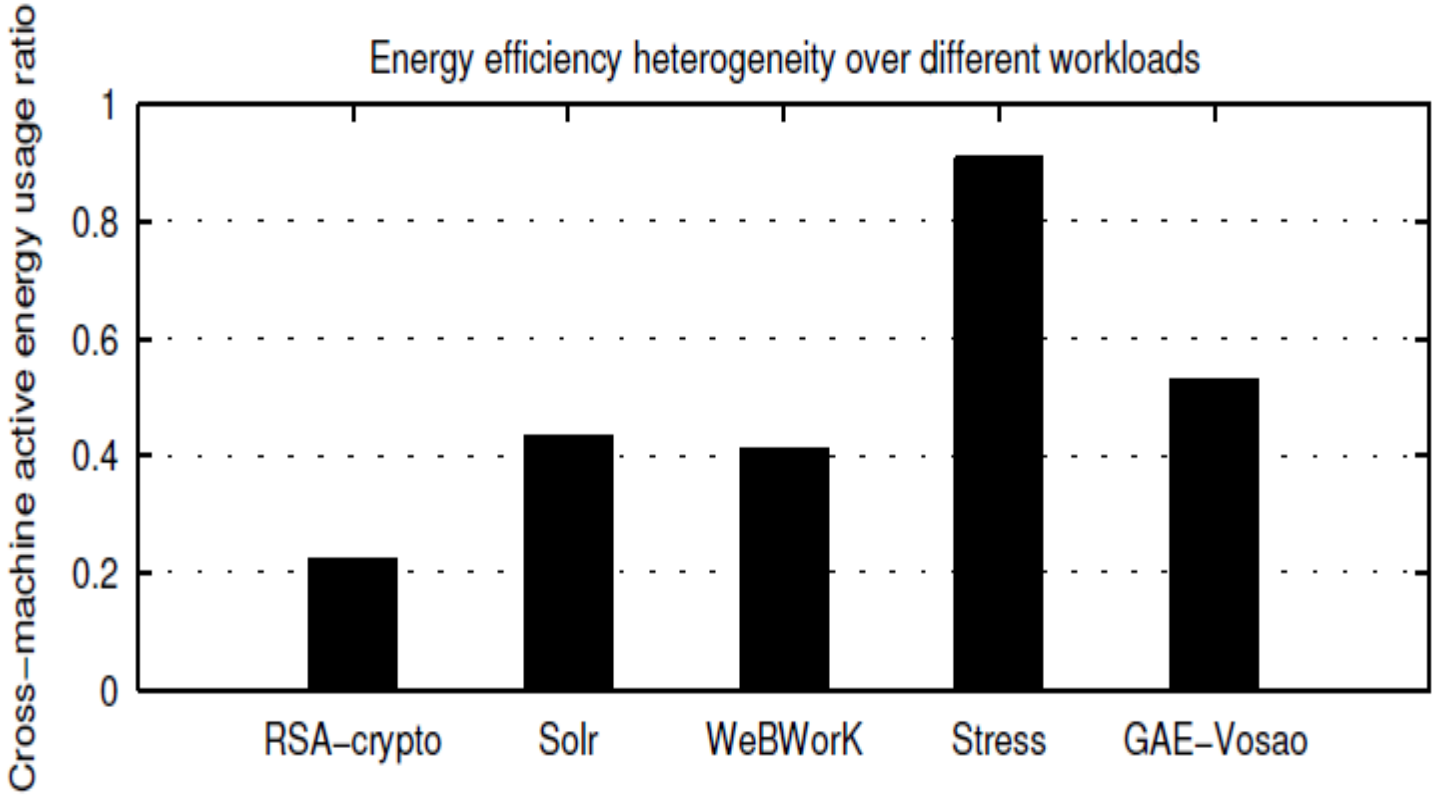
- **Accuracy of power prediction of Power container**



→ Power containers prediction is pretty accurate

Evaluation

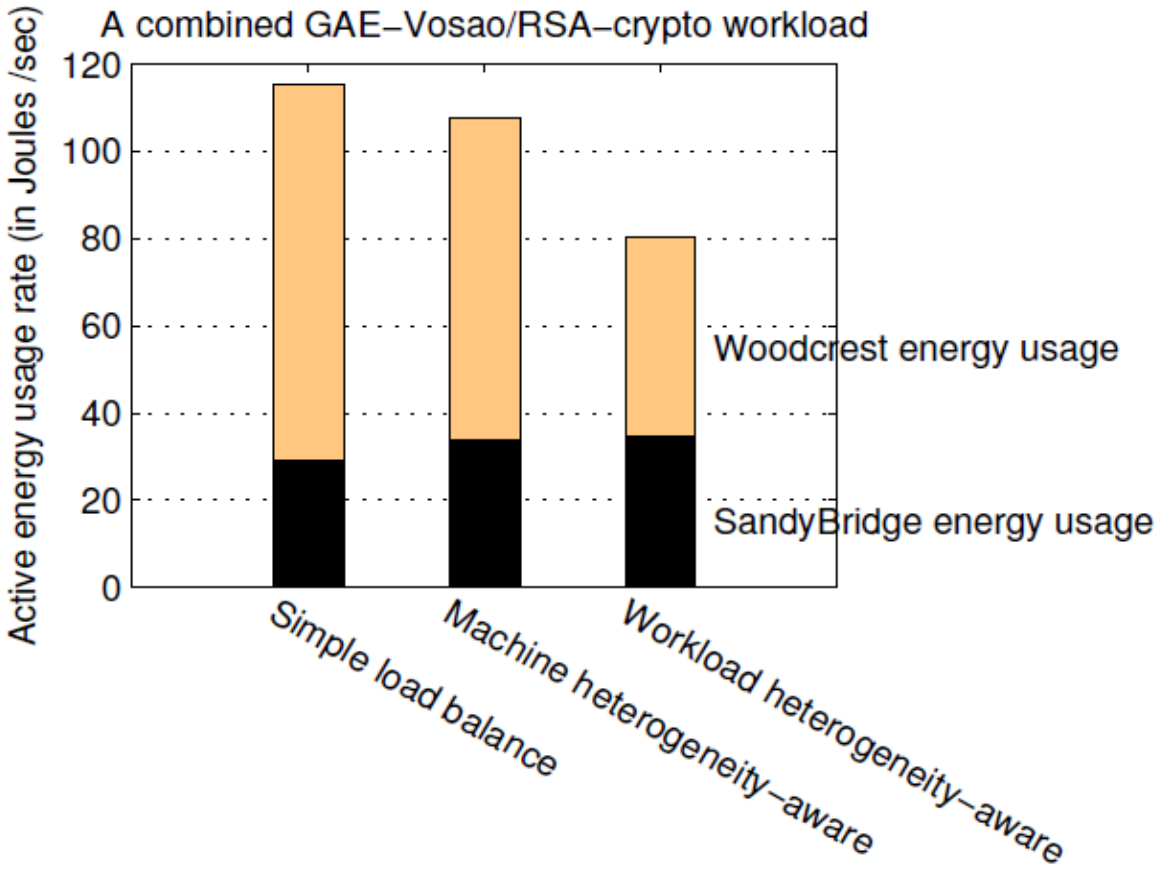
- **Heterogeneity-aware request distribution**



(energy usage on SandyBridge over that on Woodcrest)

Evaluation

- **Heterogeneity-aware request distribution**



→ Heterogeneity-aware request distribution by request tracking is effective to low power consumption

Summary

- **Fair request power conditioning**
  - Uncore's power consumption-aware power model
  - Recalibration with power measurement for better accuracy of prediction
  - Prevent power spike
  - Server power cap : entire system reliability
- **High throughput & QoS**
  - Per-request power management
    - Guarantee performance service required by per users within limited power budget(cap)
    - Per-request context tracking
- **Heterogeneity**
  - Load placement and distribution on available
  - By using cumulated power consumption results

THANK YOU