POWER BUDGETING FOR VIRTUALIZED DATA CENTERS

Harold Lim (Duke University)

Aman Kansal (Microsoft Research)

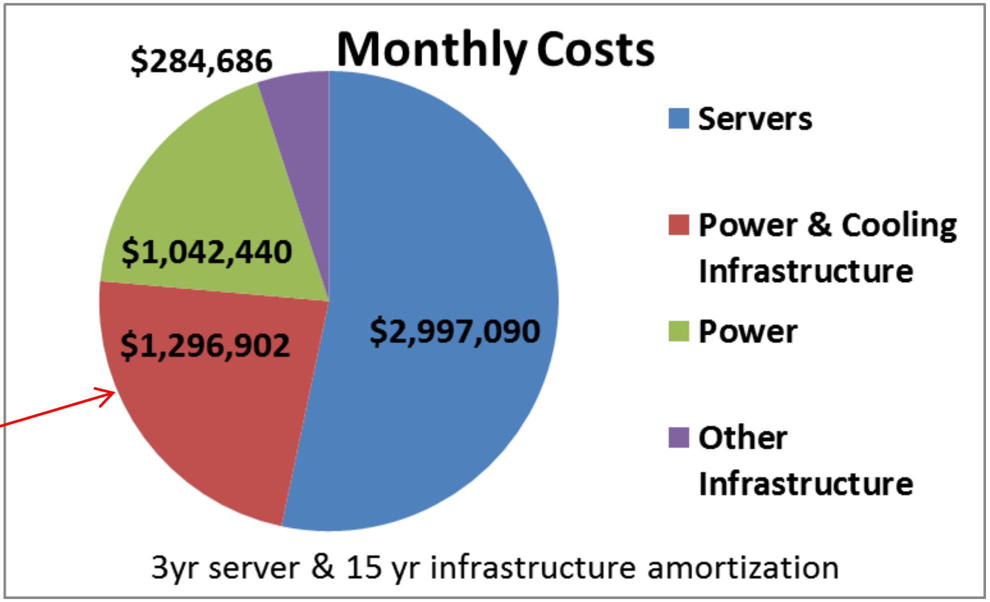
Jie Liu (Microsoft Research)

Power Concerns in Data Centers

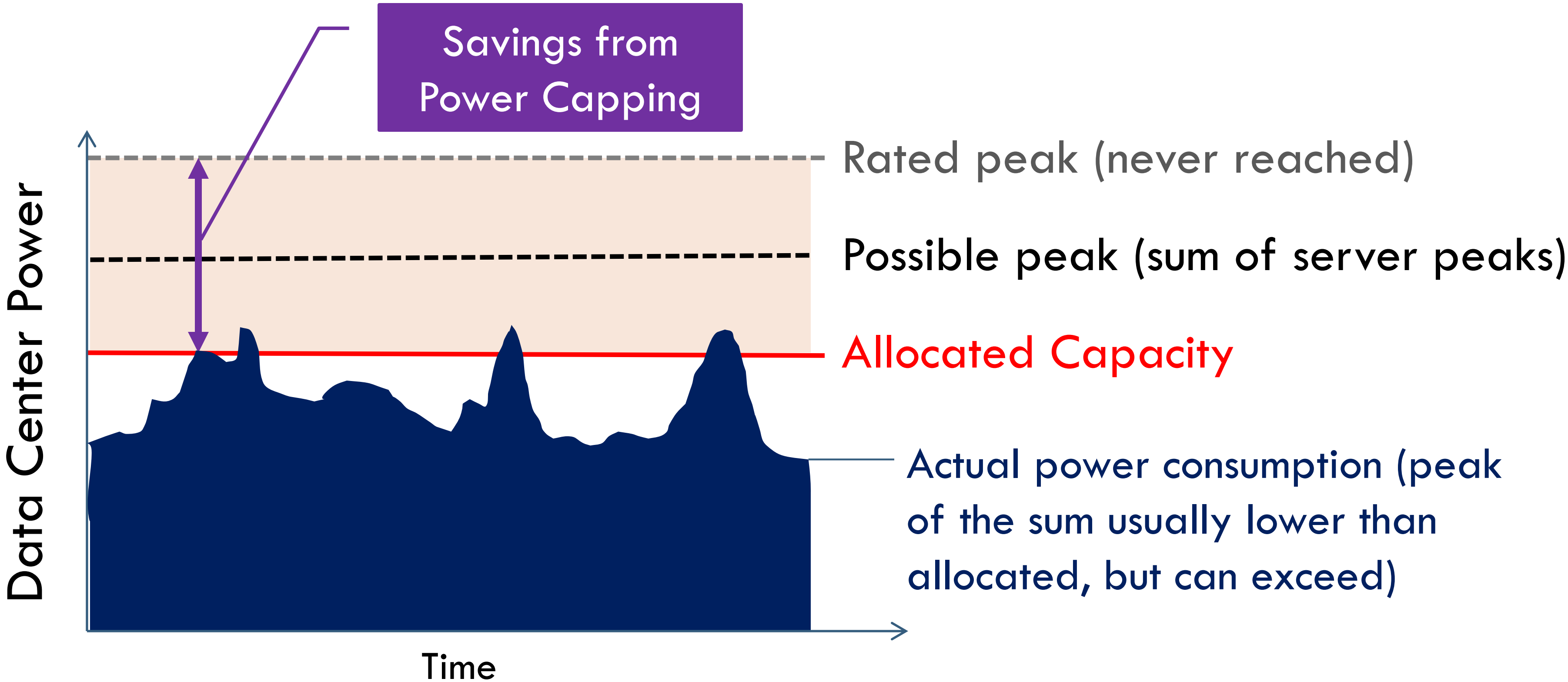
- Consumption costs
- **Provisioning costs**
  - Cost of supply infrastructure, generators, backup UPSs
  - Can be higher than consumption cost in large data centers due to discounted/bulk price on consumption
  - Addressed through **peak power management**

Data from James Hamilton

Provisioning Cost



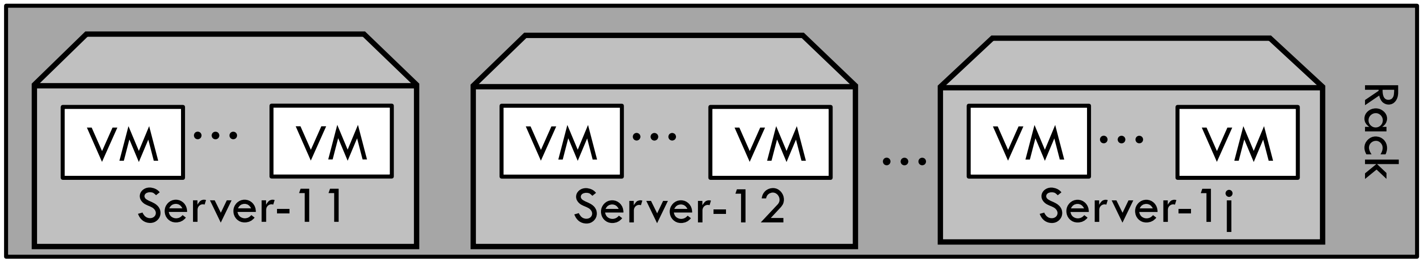
Over-subscription reduces provisioning cost



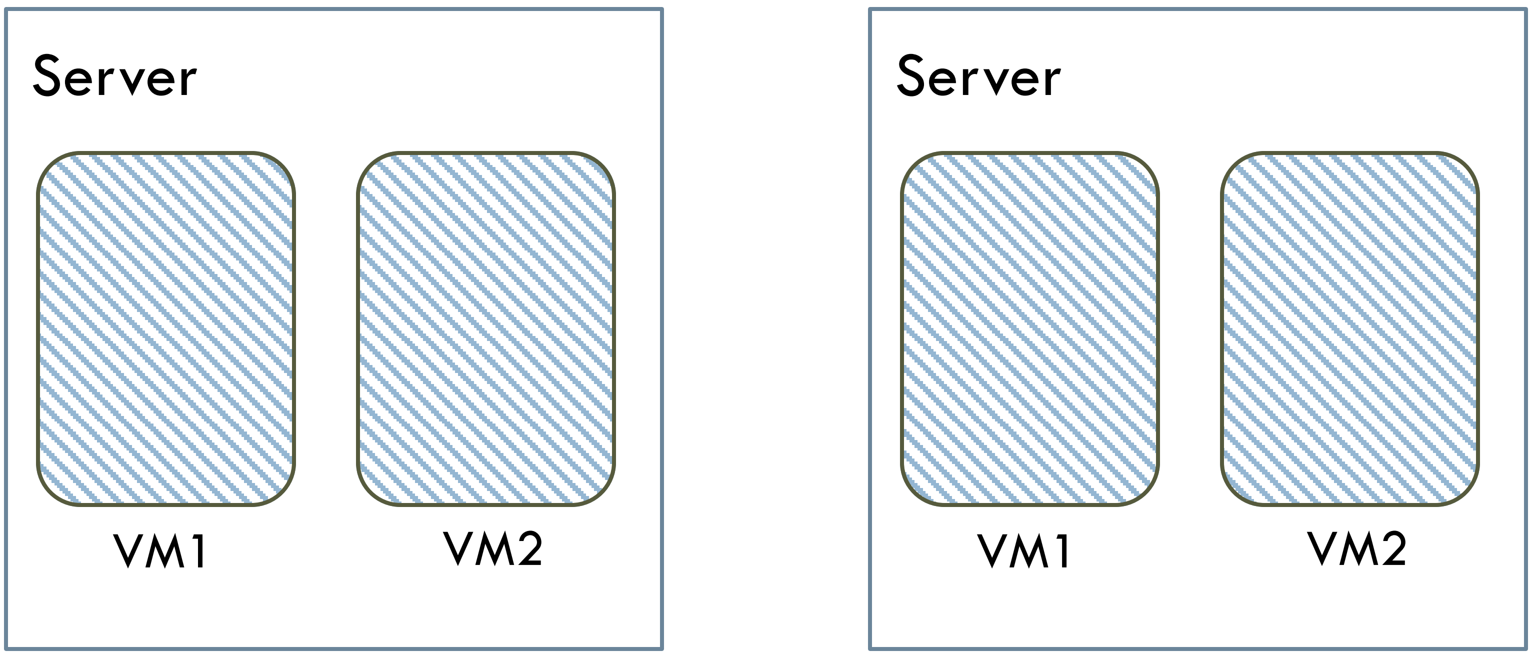
- Lower allocated capacity => lower provisioning cost (Slight perf hit)
- Possible because **power can be capped if exceeds** [Lefurgy et al. 2003, Femal et. al 2005, Urgaonkar et al. 2009, Wang et al. 2010]

Enter Virtualization

- Existing capping methods fall short
  - Servers shared by VMs from different applications: **cannot cap a server or blade cluster** in hardware



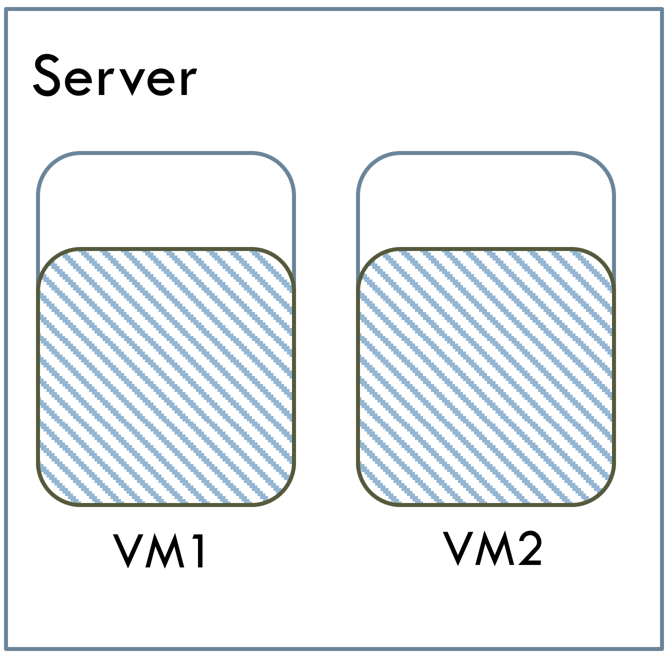
Challenge 1: Disconnect Between Physical Layout and Logical Organization of Resources



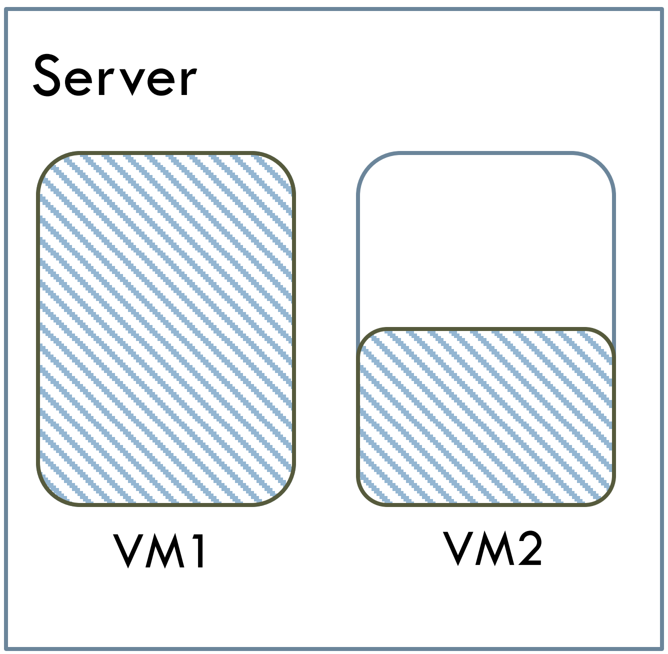
Existing Hardware Capping: Unaware of Applications

Need: Application Aware Capping

Challenge 1: Disconnect Between Physical Layout and Logical Organization of Resources



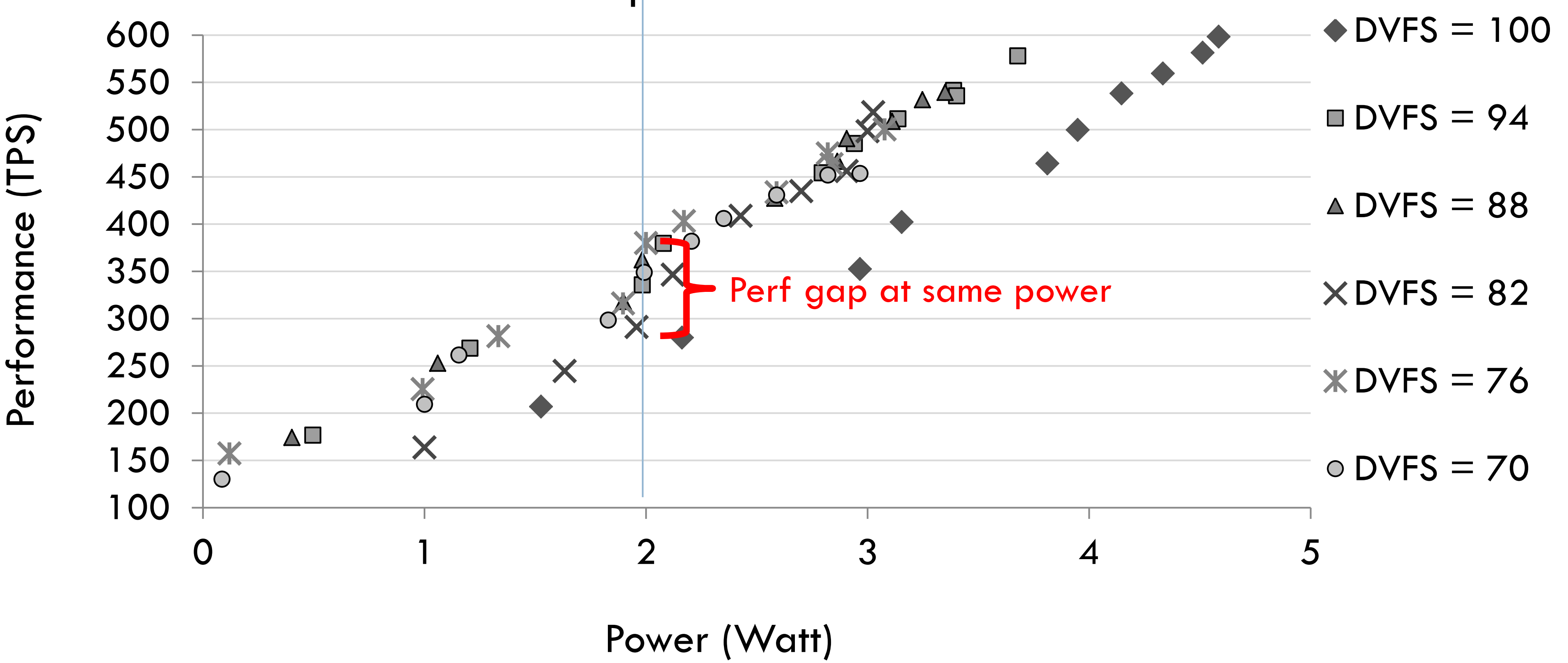
Existing Hardware Capping: Unaware of Applications



Need: Application Aware Capping

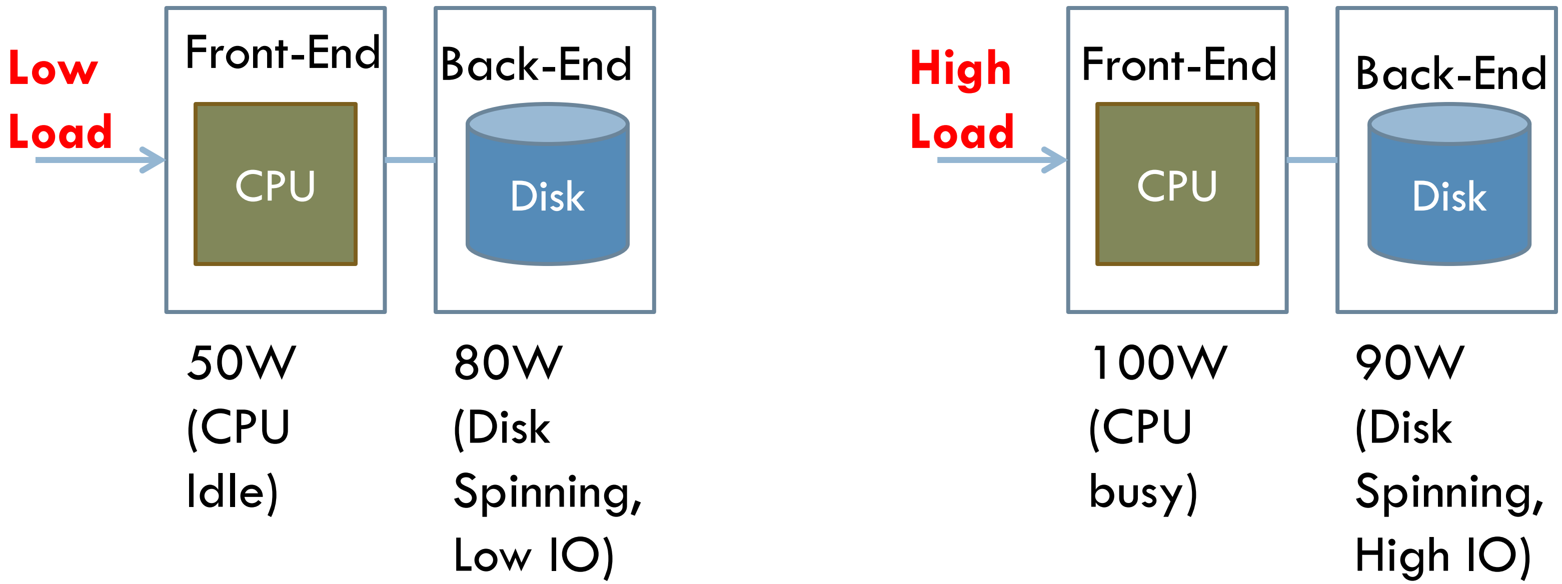
Challenge 2: Multi-dimensional Power Control

Two knobs: DVFS and CPU time cap
Different marks are different DVFS levels, multiple marks correspond to different CPU time caps



Challenge 3: Dynamic Power Proportions

- Applications' input workload volume changes over time
  - Proportion among applications changes
  - Proportion of power among app tiers changes

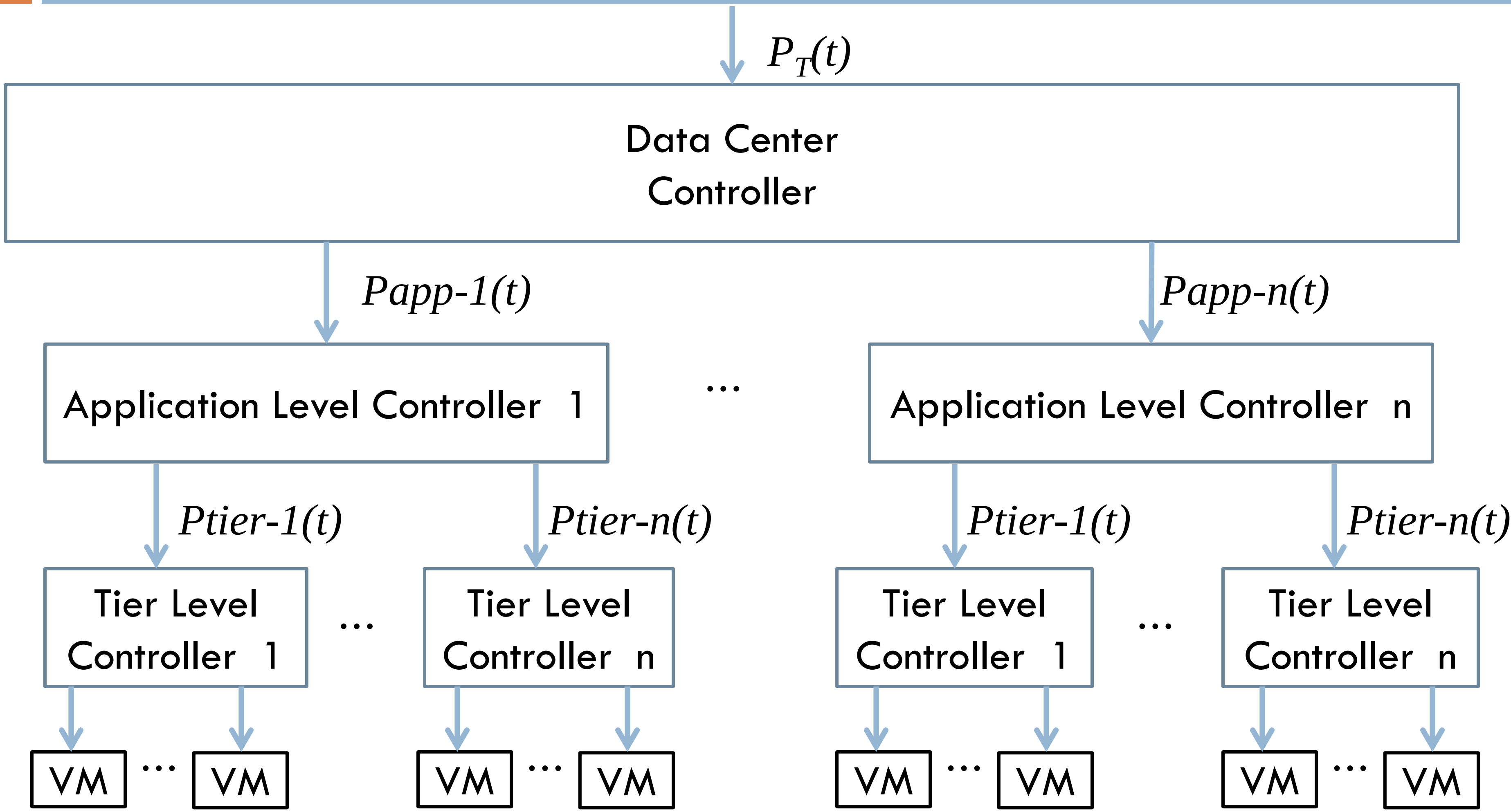


Virtualized Power Shifting (VPS): A Power Budgeting System for Virtualized Infrastructures

Addresses the above three challenges

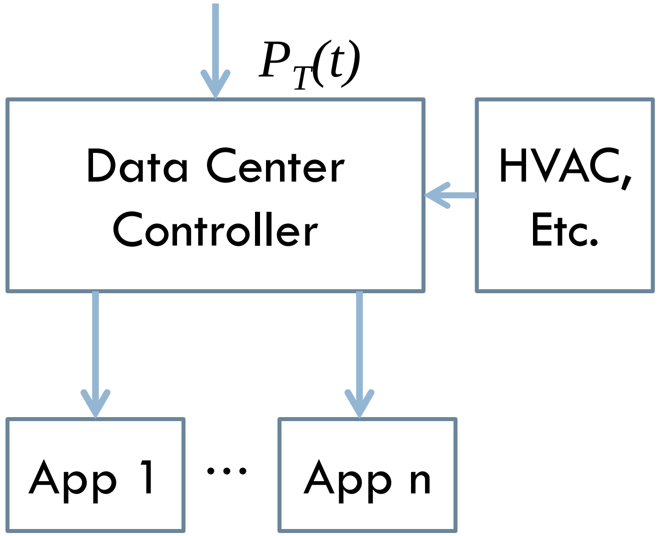
- Application-aware
  - Eg. Interactive apps not affected during capping
- Shifts power dynamically as workloads change
  - Distributes power among applications and application tiers for best performance
- Exploits performance information (if available) and multiple power knobs
  - Selects optimal operating point within power budget

Application-aware Hierarchy

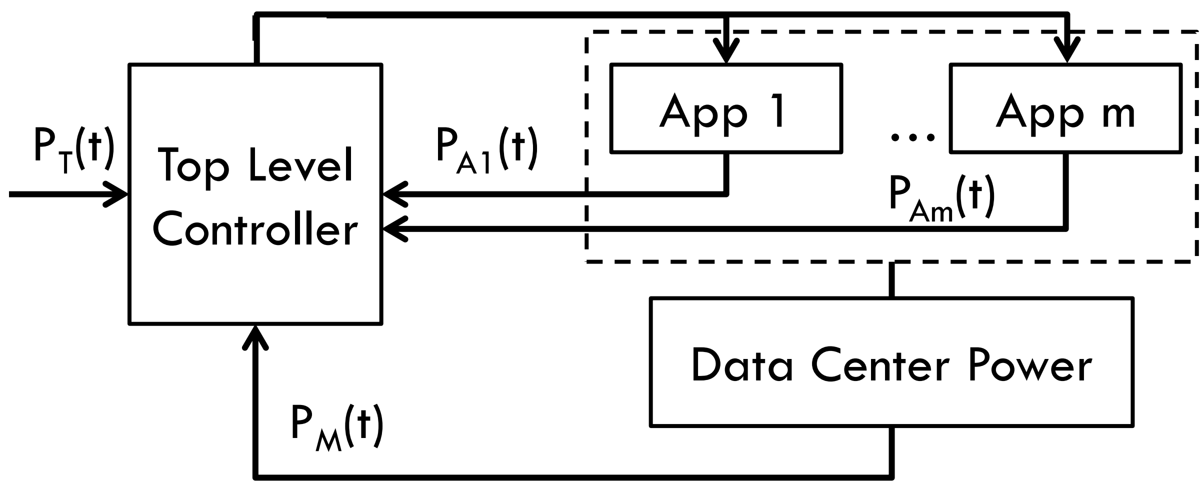


Top Level Controller: Issues

- Determines amount of power for each application
- **Static allocations does not work**
  - Dynamic workloads and power usage
  - Unused power wasted
- Must **compensate for hidden power increase** in shared infrastructure (e.g., cooling load) that are hard to assign to each application



Top Level Controller: Solution



- Uses feedback (PID) to adapt to dynamic workload and power
- Estimates uncontrollable power
  - $P_U(t) = P_M(t) - \mathrm{Sum}(P_{ai}(t))$
- Outputs application power to be allocated
  - $P_{app}(t+1) = P_M(t) + \Delta(t+1) - P_U(t)$

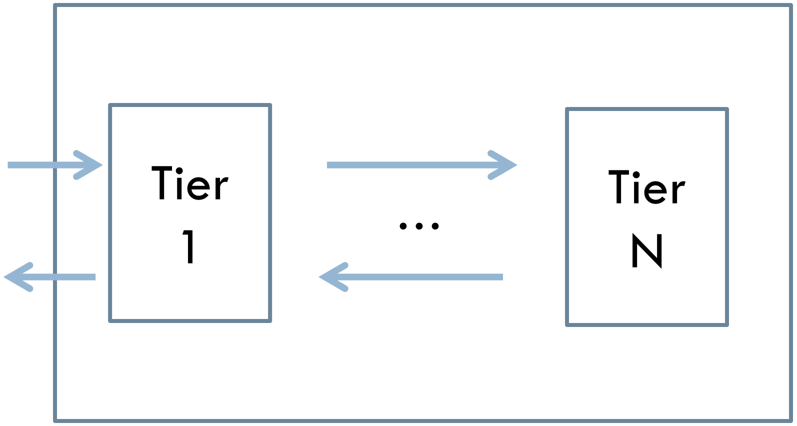
Top Level Controller: Power Split

How is P_{app} distributed among apps?

- Using Weighted Fair Sharing (WFS)
  - Each application has an initial budget
    - E.g., 99^{th} percentile of its max power
  - In each priority class, allocate power needed to each app, up to its initial budget
  - If not enough power, allocate proportion via WFS

Application Level Controller: Issues

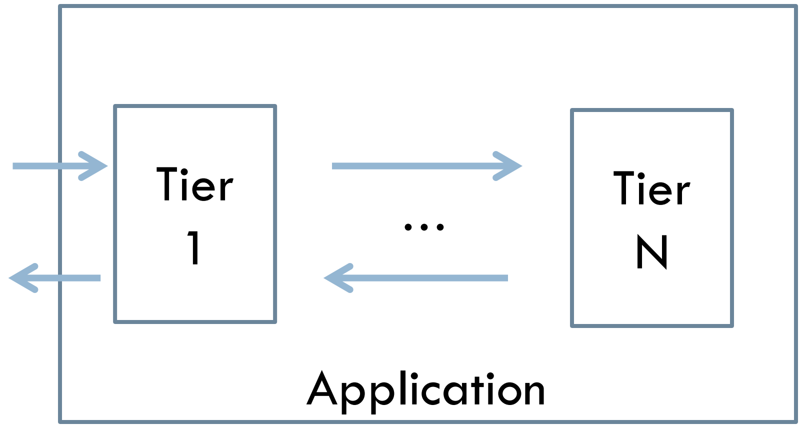
- Determines how much budget to allocate to each tier
- Prior work: Learn model of power ratios among tiers a-priori. **Problems:**
  - Model changes with workload
  - Depends on the control knobs used
  - Application behavior may change over time



Application

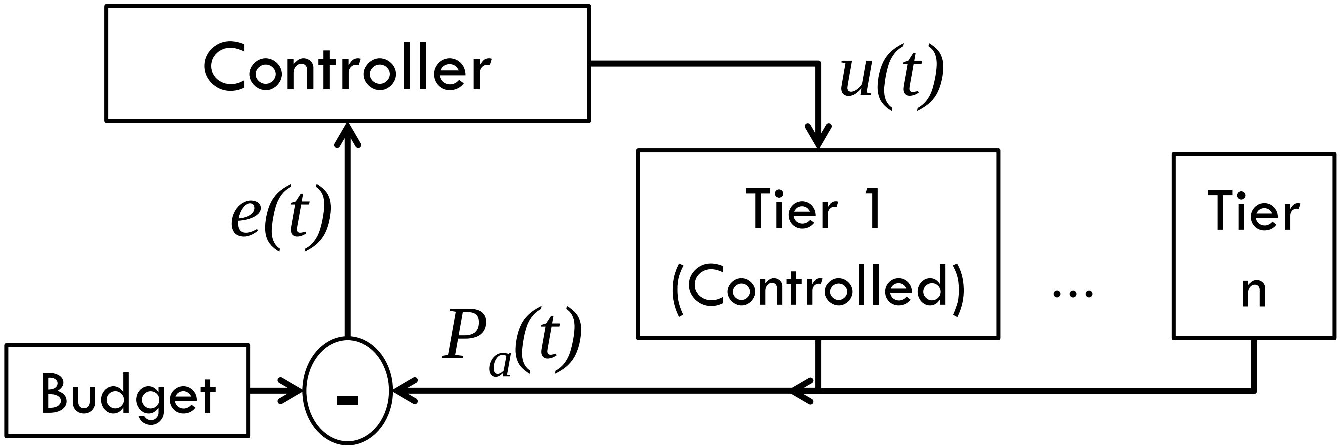
Application Level Controller: Solution

- VPS: dynamically tunes power allocations without relying on learned models
- Observations:
  - Application tiers are **arranged in a pipeline**
  - Throttling one tier affects other tiers



Application Level Controller (contd.)

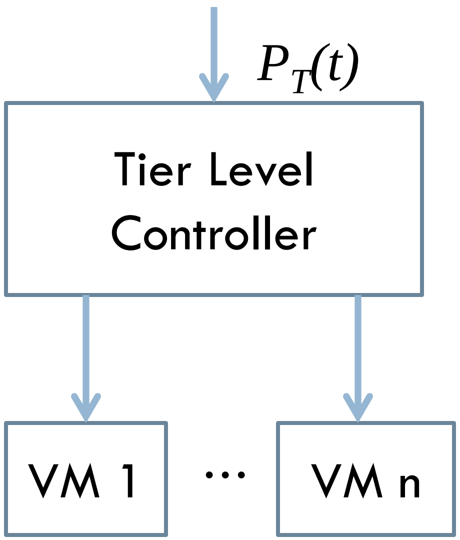
- Uses PID control
  - Measures **total** application power usage but **only control one** tier
  - Automatically achieves right proportion



Controller: $$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t)$$

Tier Level Controller

- Tracks tier power budget by controlling VM power usage
- Many power control knobs available
  - Use DVFS and VM CPU time allocation as knobs



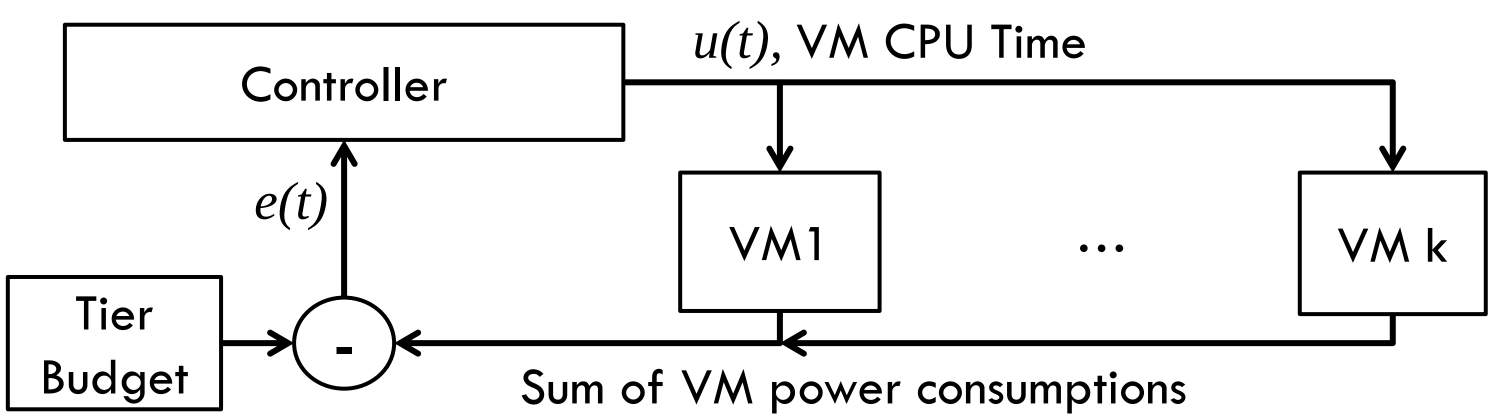
- **Multiple trade-offs exist** w.r.t accuracy, speed, models needed, app visibility needed
  - Study 3 design options

Option 1: Open Loop Control

- Uses power model to convert power budget to control knob setting
  - E.g., $P_{VM}=c_{freq}*u_{cpu}$
- Easy and **instantaneous**
- Does not require visibility into application performance
- *But* does not compensate for errors

Option 2: PID Control

- Real time measurements to tune power settings: **compensates for error**
  - Slower (needs time to converge)
  - Single control knob (no notion of performance optimality)



Option 3: Model Predictive Control (MPC)

- **Optimizes performance** using multiple power control knobs (DVFS and VM CPU time)
  - Uses a cost function that consists of error and **performance** terms
  - Solves for the optimal outputs for the next N steps but only apply the setting for next time step
- Requires application performance measurement
- Requires system models that relate control knobs to system state

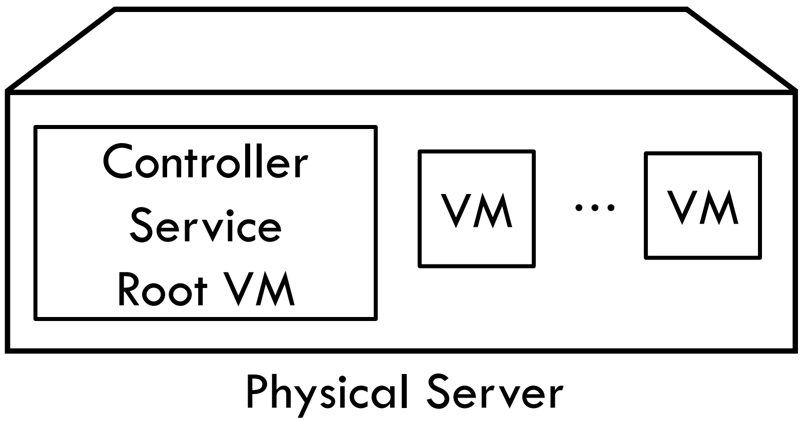
$$J = \sum_{i=1}^{N} ||f_{power}(dvfs(i), v(i)) - P_{VM}|| + w\sum_{i=1}^{N} ||f_{perf}(dvfs(i), v(i)) - \alpha_{max}||$$

Summary of Design Options

	Pros	Cons
Open Loop	Fast	Needs power models Higher error
PID	Low error	No performance optimization Slower
MPC	Optimizes performance	Needs system models Needs performance measurement

Experiments

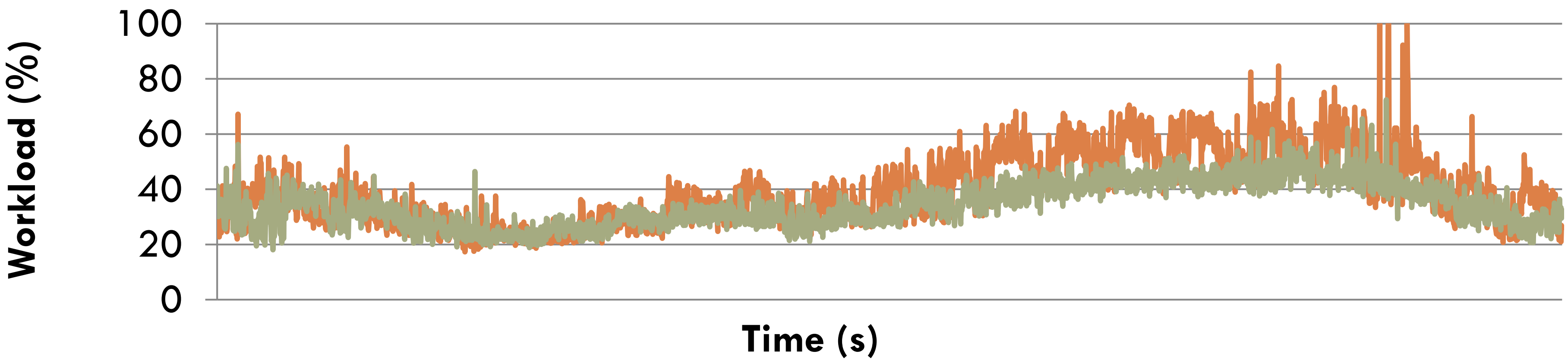
- VPS controllers run as network services in root VM on each server
  - Controller tuned using known methods



- **Testbed:** 17 Quad core HP Proliant servers (11 host the apps, 6 generate the workload)
  - VMs mixed across the physical servers
  - VM power measured using Joulemeter, Hardware power using WattsUp PRO meters

Experiment Workloads

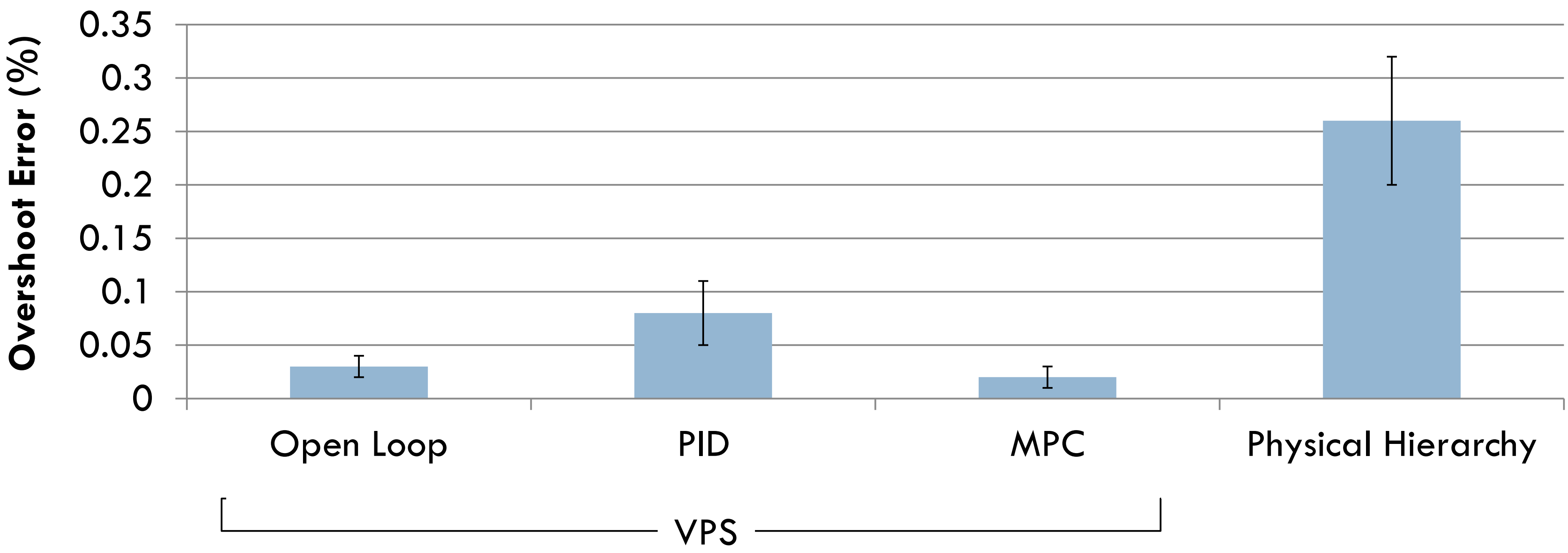
- Applications
  - **Interactive:** StockTrader – open source multi-tiered cluster web application benchmark
    - 3 instances, 2 are High priority
  - **Background:** SPEC CPU 2006 benchmark
    - Low priority
- Use Microsoft data center traces as input to simulate realistic workloads that vary over time



Metric: Total Budgeting Error

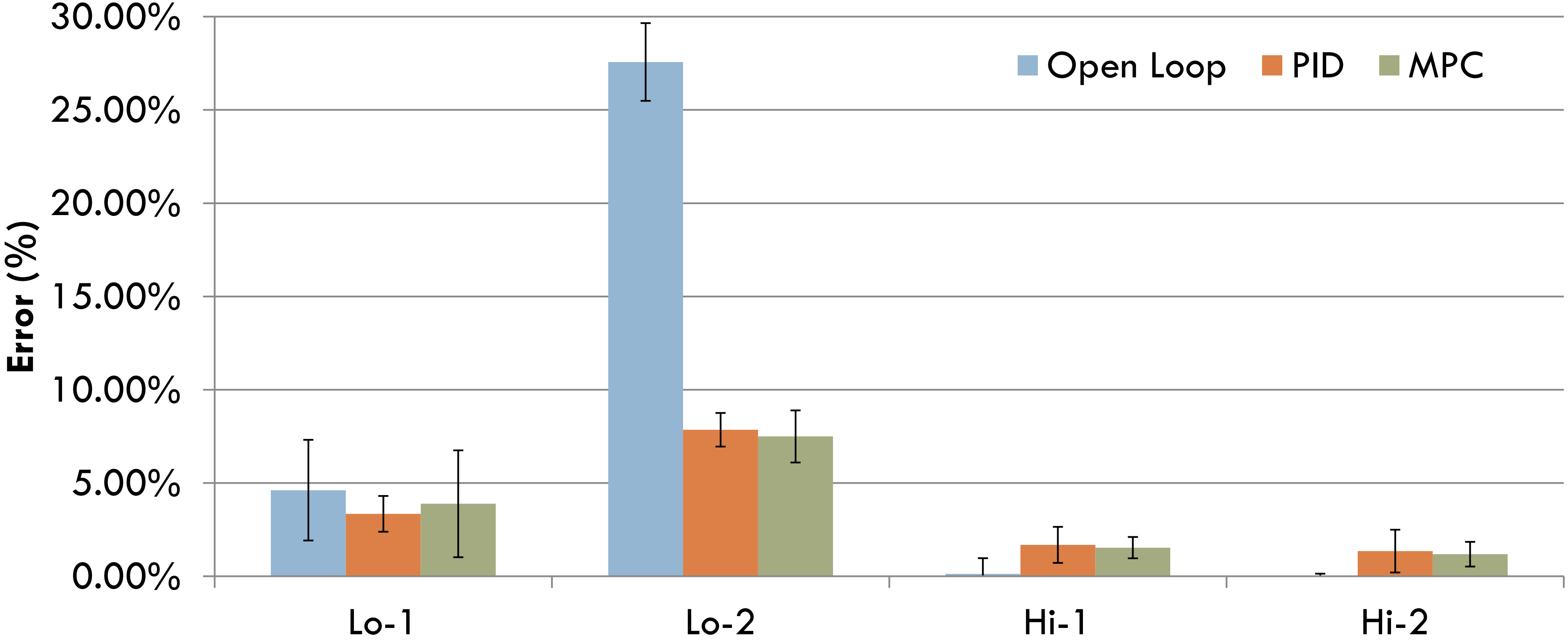
- Error = excess power consumed above the assigned budget, normalized by the power budget

$$TrackingError = MAX\left\{\frac{P_M(t) - P_T(t)}{P_T(t)}, 0\right\}$$



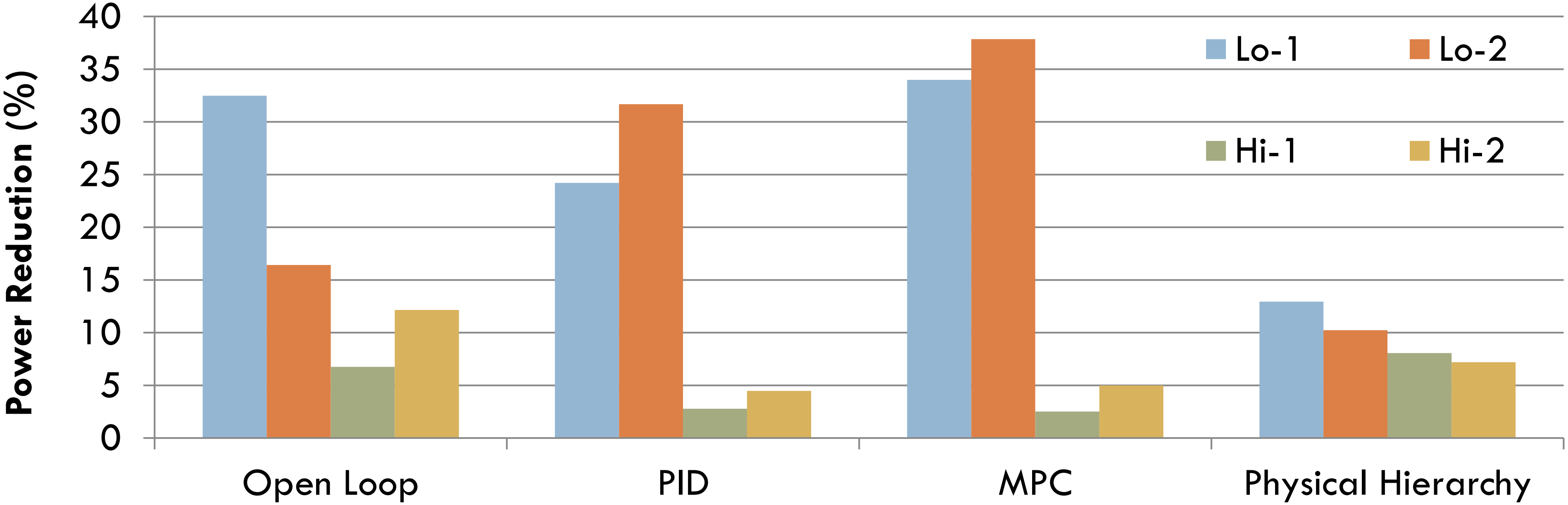
Metrics: Errors within App Hierarchy

- Application power enforcement errors



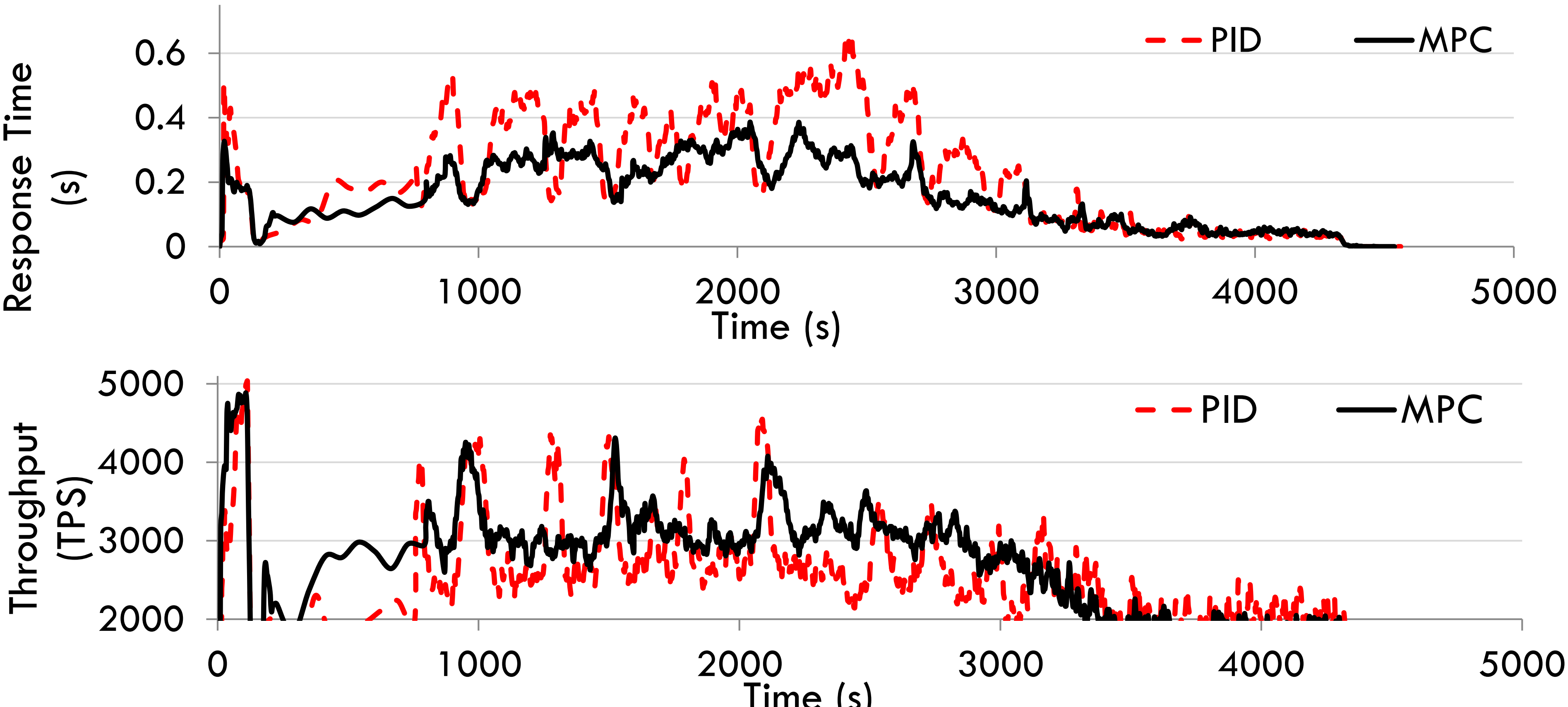
Metric: Power Differentiation

- VPS is designed to **respect application priorities** and QoS constraints in a shared infrastructure
- PID and MPC perform appropriate application differentiation



Metric: Application Performance

- Performance of (low priority) app that was capped



Conclusions

- VPS: power budgeting system for virtualized data centers
- Hierarchy of control follows application layout
  - Respects application priorities and application VM boundaries
- Optimizes application performance, given a power budget
  - Dynamically adjusts power proportions
  - Exploits multiple knobs