

# **Cluster-level Feedback Power Control for Performance Optimization**

**Xiaorui Wang and Ming Chen**

Dept. of EECS  
University of Tennessee, Knoxville

# Introduction

- Power is a serious concern for datacenters
  - Operating costs
  - System failures due to overheating
- High-density servers make it worse
  - Greater probability of thermal failures
- Power control
  - Most components have adjustable performance/power states
    - CPU, hard disk, memory, etc
  - Lower performance state → lower power consumption
- **Goals** of this paper
  1. Control power of a **server cluster**  $\leq$  a power budget
  2. Achieve best possible application performance

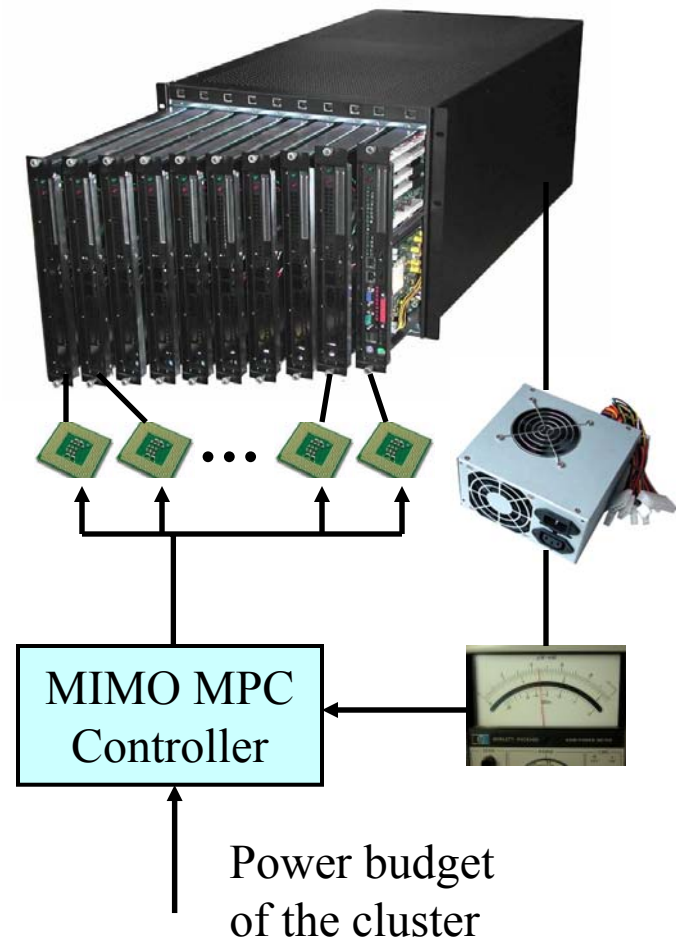


# State of the Art

- Reducing power consumption of a component
  - CPU, memory [Delaluz],[Aggarwal], disk [Gurumurthi],[Carrera]
- Adjusting power to control application performance
  - [Bohrer], [Sharma], [Chen], [Zhu], etc
- Power and thermal control for a single server
  - Heuristic solutions [Zeng], [Lu], [Brooks], etc
  - Control theory has shown promise [Lefurgy], [Minerick], [Wu], etc
  - Power shifting between different components [Felter], etc
- Power control for a server cluster
  - Mostly heuristic solutions [Ranganathan], [Femal], etc

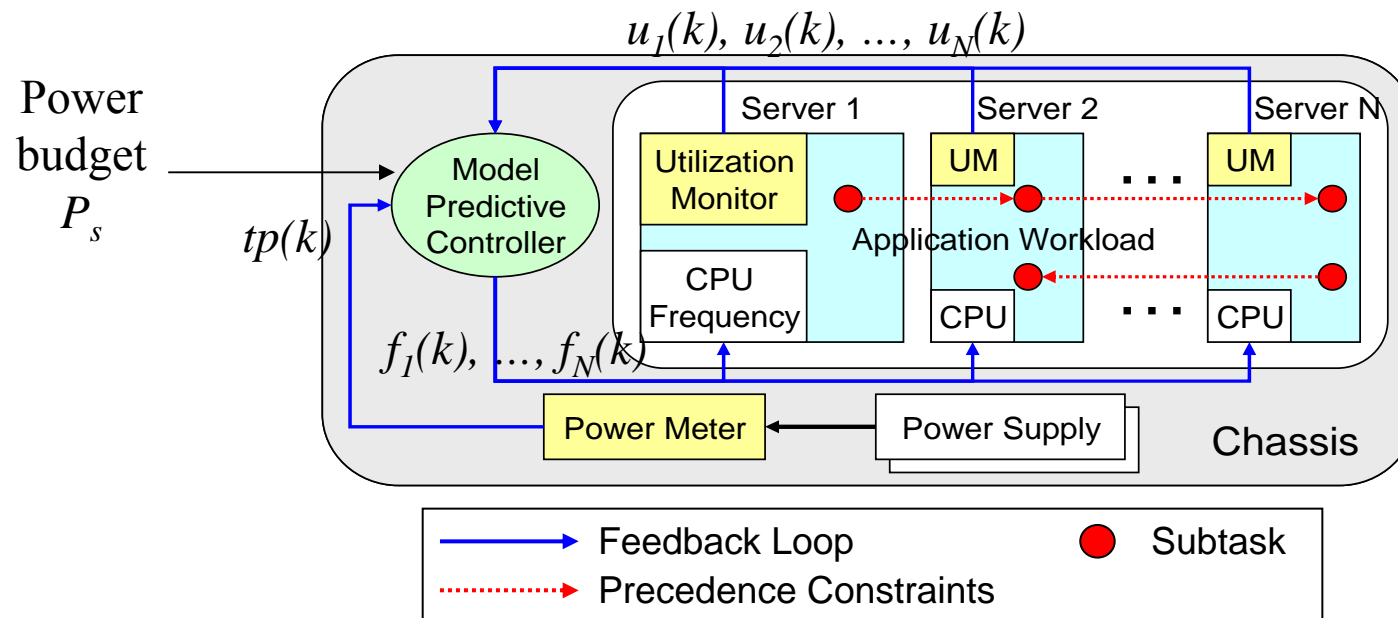
# What is This Paper About?

- **Cluster-level** power control
  - Servers are often correlated together
    - e.g., due to shared power supply
  - Power shifting among servers
- **Control-theoretic** design
  - Guaranteed control accuracy and stability
- Multi-Input-Multi-Output (MIMO) control
  - Controlled variable: **total power consumption** of the cluster
  - Manipulated variables: **processor Dynamic Frequency Scaling (DFS)**
  - Control algorithm: **Model Predictive Control (MPC)**



# Power Control Loop

- Control loop invoked **periodically**
  - Power meter sends the total power to the controller
  - CPU utilization monitors send utilizations to the controller
  - Controller conducts constrained MPC computation
  - New CPU frequency levels are sent to the frequency modulators



# Steps of Model Predictive Control

1. Derive a dynamic model for the controlled system
2. Design the controller
3. Analyze stability and control accuracy

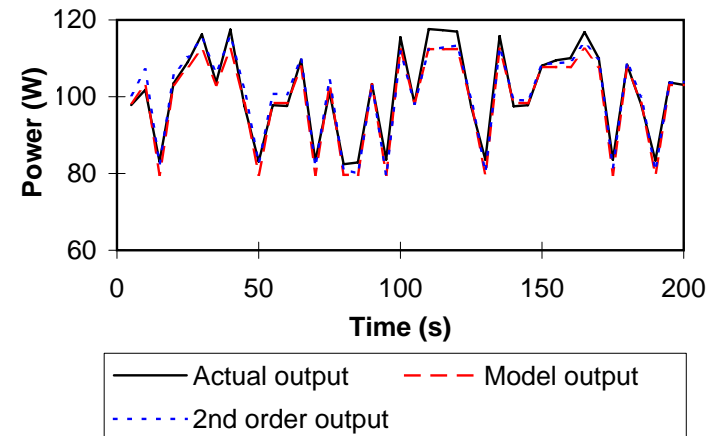
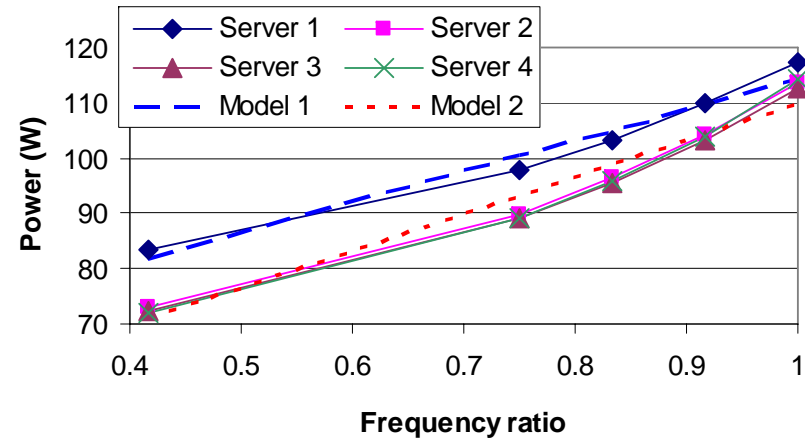
Control objective:

$$\min_{\{f_j(k)|1 \leq j \leq N\}} (P_s - tp_i(k))^2$$

subject to **constraints:**  $F_{min,j} \leq f_j(k) \leq F_{max,j} \quad (1 \leq j \leq N)$   
 $tp(k) \leq P_s$

# System Modeling and Verification

- Modeling
  - System identification
  - Power changes as frequency changes
- Linear Model
  - $P_i(k) = A_i f_i(k) + C_i$
- Model verification
  - White noise input
- Difference model for control
  - $P_i(k+1) = P_i(k) + A_i d_i(k)$   
where  $d_i(k) = f_i(k+1) - f_i(k)$

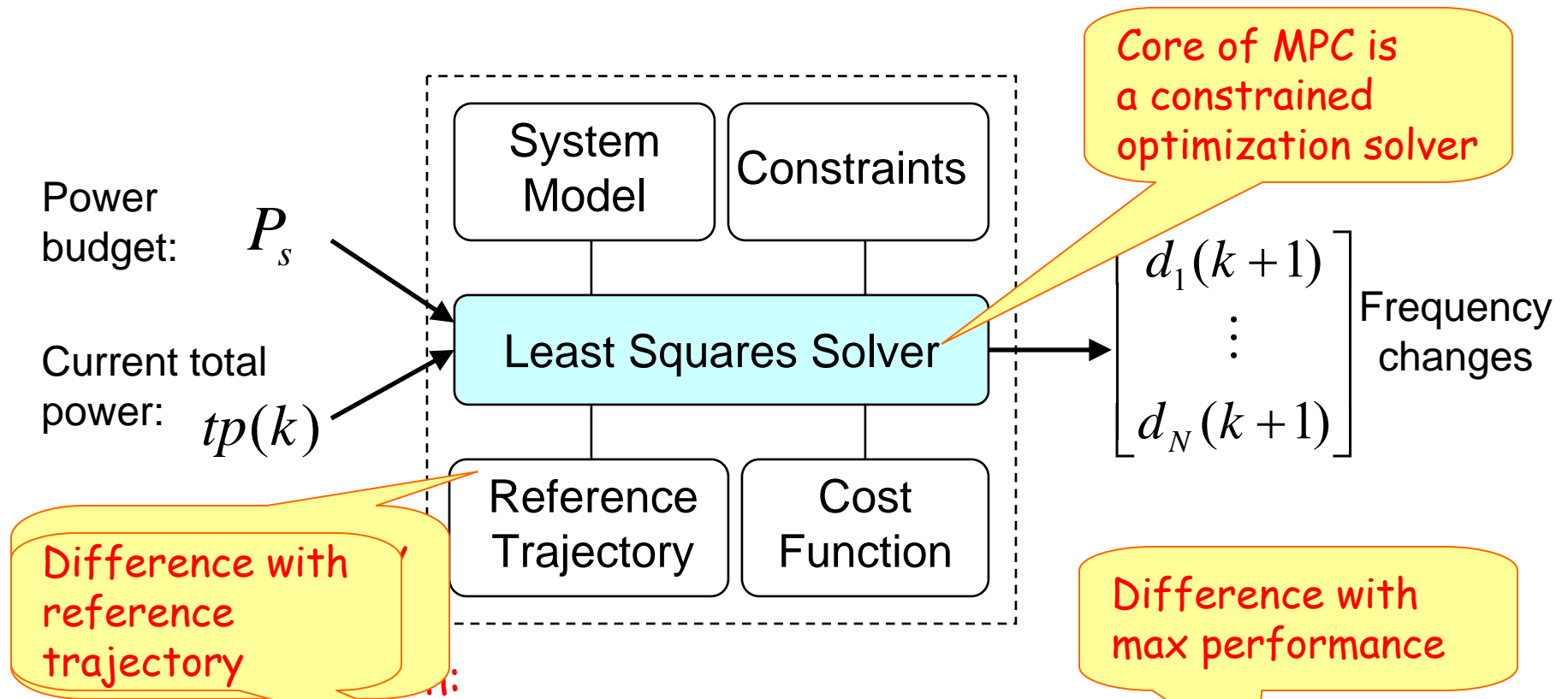


# Modeling Workload Variations

- Model is inaccurate when the controller is used for
  - A different server or/and a different workload
- Actual system model
  - $P_i(k+1) = P_i(k) + g_i A_i d_i(k)$ ,
  - where  $g_i$  is unknown at design time and captures the model variations
- Actual model of whole cluster

$$tp(k+1) = tp(k) + \begin{bmatrix} g_1 A_1 & \dots & g_N A_N \end{bmatrix} \begin{bmatrix} d_1(k) \\ \vdots \\ d_N(k) \end{bmatrix}$$

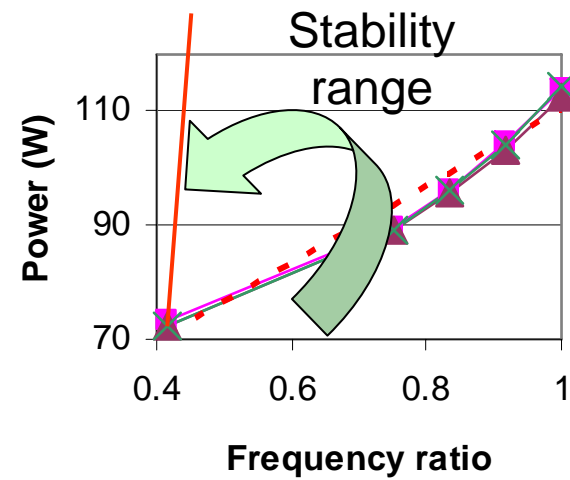
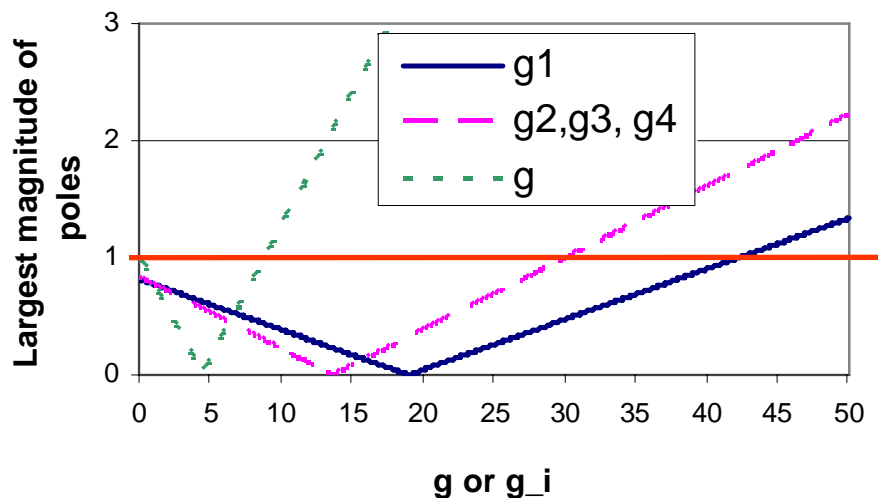
# Model Predictive Controller Design



$$V(k) = \sum_{i=1}^P \|tp(k+i/k) - ref(k+i/k)\|_{Q(i)}^2 + \sum_{i=0}^{M-1} \|d(k+i|k) + f(k+i|k) - \mathbf{F}_{\max}\|_{\mathbf{R}(i)}^2$$

# Stability Analysis

- Stability: closed-loop poles are within unit circle
  - The largest magnitude of all the poles is smaller than 1
- Stability condition: tolerable range of workload variations
  - Assuming all servers have uniform variations
    - $0 < g \leq 8.8$ : slope of the real model is 8.8 times of nominal model
  - Assuming one server has variations while others don't
    - $0 < g_1 \leq 42.1$  and  $0 < g_2, g_3, g_4 \leq 29.9$

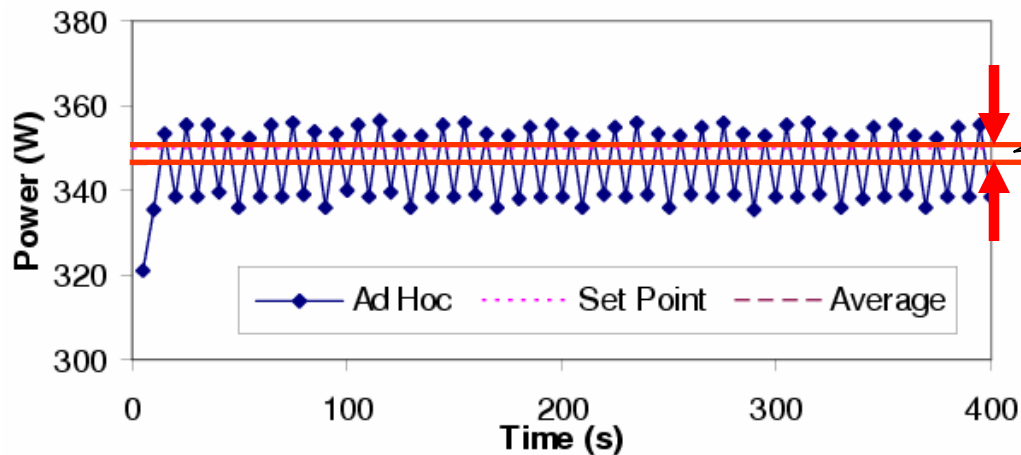


# System Implementations

- Test-bed
  - 4 Linux servers with AMD 3800+ processor (5 freq levels)
  - Controller runs on a separate Linux server
- Wattsup power meter
  - Accuracy:  $\pm 1.5\%$ ; Sampling period: 1 second
- Controller
  - Calls Matlab function for MPC computation
  - Control period: 5 seconds (5 power readings)
- CPU frequency modulator
  - 5 discrete freq levels to approximate a fractional level?
    - For 3.2, use 3, 3, 3, 3, 4 on a smaller timescale (subintervals)
  - 50 subintervals (100ms) in each control period
    - Worst actuation overhead =  $100\mu s / 100ms = 0.1\%$

# First Baseline – Ad Hoc

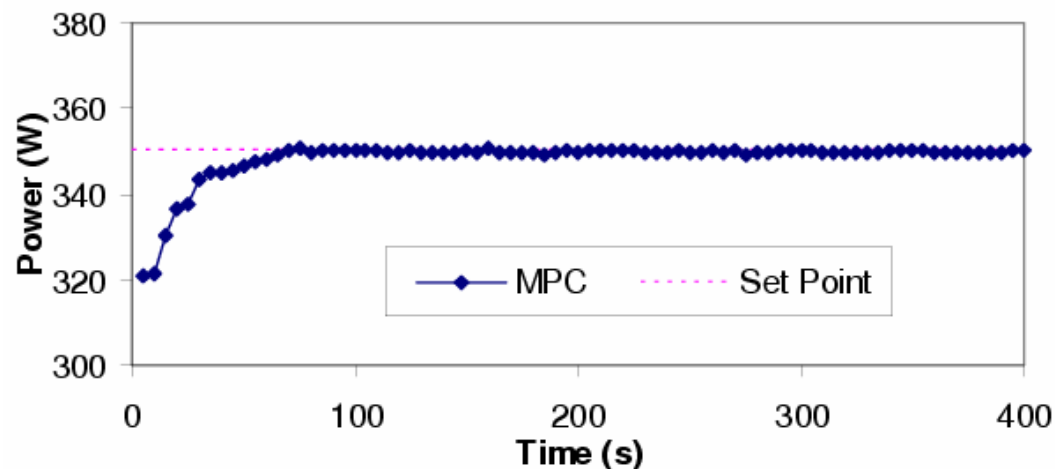
- Ad Hoc: a state-of-the-art paper (2006)
  - Represents a typical industry solution
  - Power < budget:
    - Find the server with highest CPU utilization
    - Increase its power level by one
  - Power > budget:
    - Find the server with lowest CPU utilization
    - Decrease its power level by one
- Typical run of Ad Hoc (with Linpack workload)



Ad Hoc has  
steady-state error:  
**-3.8 W**

# Typical Run of MPC

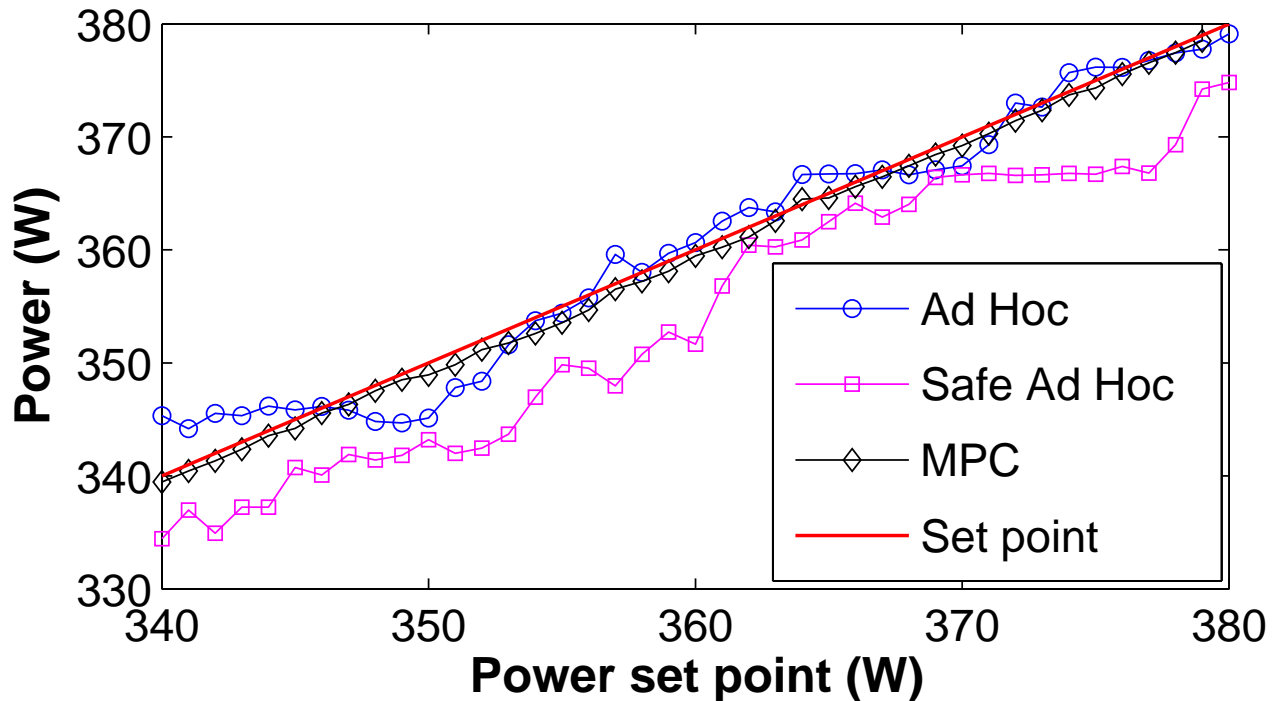
- MPC has more accurate power control (Linpack)



- Difference between MPC and Ad Hoc
  - MPC generates a fractional freq level based on control theory
  - MPC uses smaller subintervals to approximate the fractional level
  - Ad Hoc simply jumps up and down without system model
  - Ad Hoc cannot use subintervals because it has no fractional level

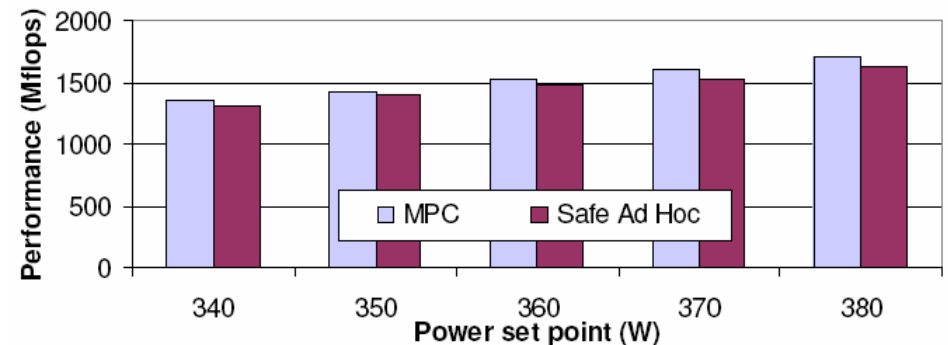
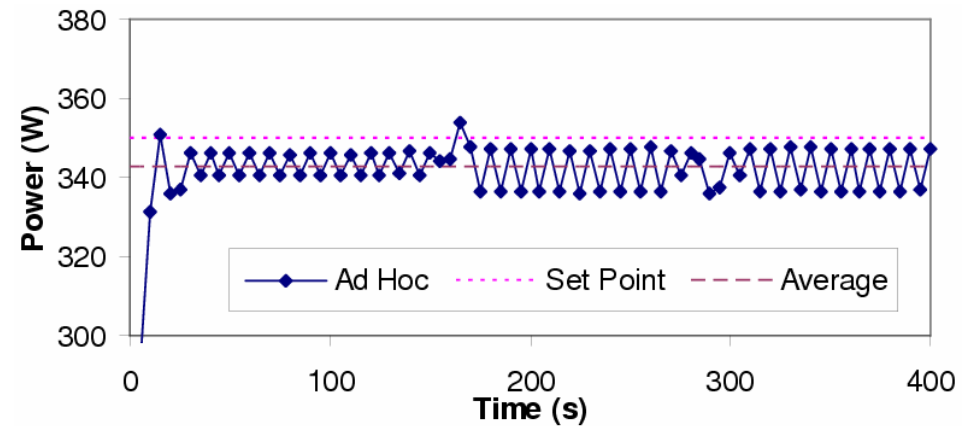
# Steady-State Error

- **MPC** achieves all the power set points
- **Ad Hoc** always has steady-state errors, oftentimes positive



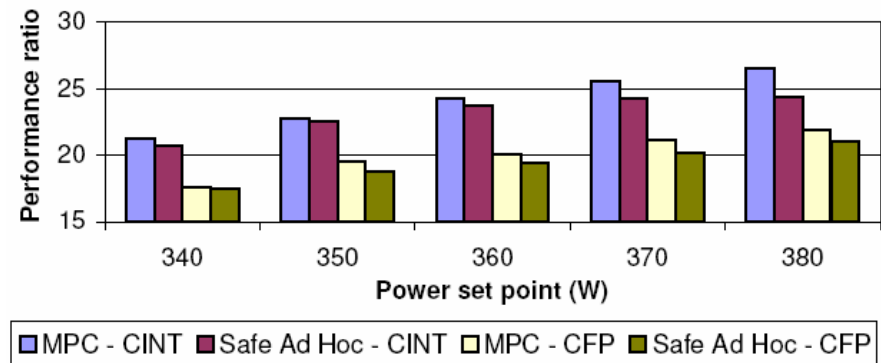
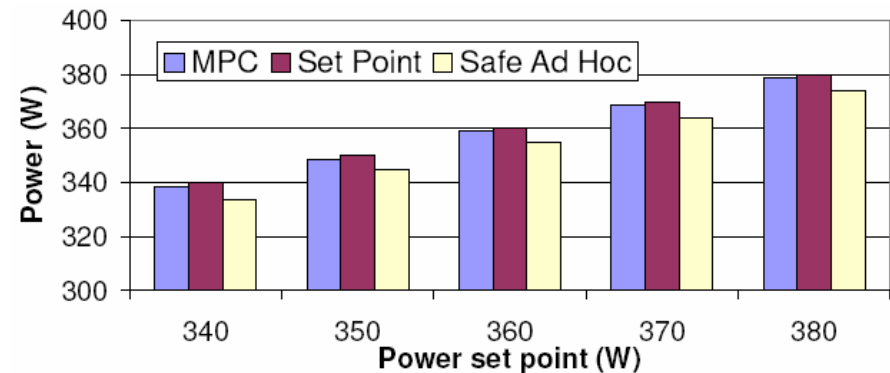
# Modified Baseline - Safe Ad Hoc

- Ad Hoc with safety margin (SM)
  - SM is maximum positive error of all set points
  - Modified set point = real set point – SM
- MPC has **5%** better performance on average
- Improvement would be **bigger** in real systems
  - Impossible to have such a small SM: 5.5 Watts



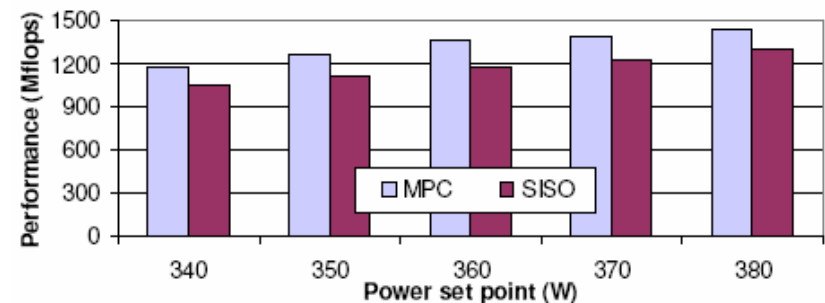
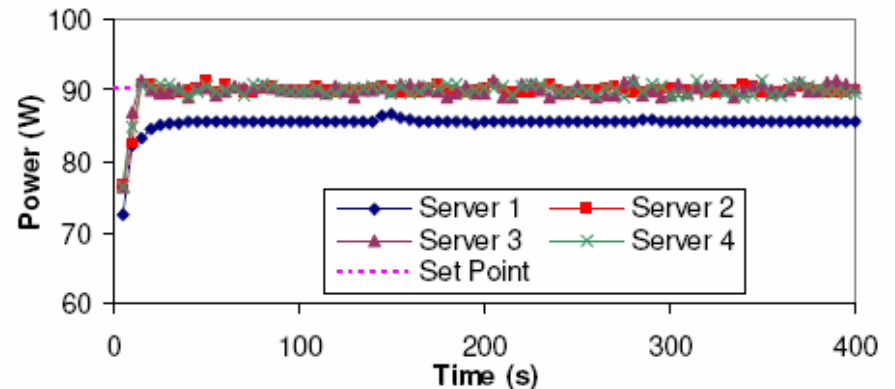
# Results Using SPEC CPU 2006

- Effectiveness of MPC with **other workloads**
- MPC vs. Safe Ad Hoc
- **Similar results** as Linpack
  - More accurate power control
  - Better application performance



# Second Baseline - SISO

- SISO for a single server
  - Single-Input-Single-Output (SISO) controller
  - A state-of-the-art work (2007)
  - Multiple controllers **evenly** share the budget
- Experiments using Linpack
  - One server is **idling** while others run Linpack
  - SISO **fails** to use up budget because of the idle server
  - MPC has **12% better** performance on average



# Conclusions

- A cluster-level power controller
  - Designed based on **MPC** control theory
  - **Shifts** power based on server performance needs
  - Theoretically guaranteed **stability** and **accuracy**
- Compared with state-of-the-art work
  - **More accurate** power control
  - **Better** application performance

# Questions?

- Acknowledgement
  - Charles Lefurgy and Malcolm Ware at the IBM Austin Research Lab
  - Supported by NSF CNS-0720663

**Thank you!**