Recap of Last Class

- Power management for virtualized servers
- Power-efficient response time guarantees
  - Presents a two-layer control solution.
  - Guarantees response time while saving power for virtualized servers.
- VirtualPower
  - Provides a framework that transparently supports guest VMs’ power management requests
  - Virtualized “soft” PM states, VPM channels, and VPM mechanisms

Introduction

- Power is a serious concern for datacenters
  - Electricity costs
    - $7.4 billion in 2011 in the US alone
  - System failures due to overheating
  - High-density (blade) servers
    - Greater probability of thermal failures
- Why power control (capping)?
  - Precisely control power to stay below a given budget
    - Power is treated as a first-class resource
  - Safety: Promptly reduce power at runtime in the case of
    - Partial power supply failure, thermal emergency
  - Avoid conservative power over-provisioning – host more servers
  - Flexible power shifting among servers/components
    - Improved performance/power ratio

Power Control (Capping)

- Goals of power control
  1. Power consumption <= a power budget
  2. Achieve the best possible application performance
- How to control power to a budget?
  1. Adjustable perf/power states (e.g., DVFS)
     - Small overhead: Lower perf => less power
  2. Server shutdown/sleep
     - Higher overhead, service interruption, etc.
     - Can be utilized over long time intervals (e.g., hours)
  3. Application-specific workload redistribution/adaptation
     - Limited power adaptation range
   - Impacts on application performance
     - Best-effort performance requirements (scientific computing)
     - Service-Level Agreements (SLAs)

Server-level Power Control

Charles Lefurgy¹, Xiaorui Wang², and Malcolm Ware¹

¹ IBM Research, Austin
² University of Tennessee, Knoxville

- Adaptive power controller
  - Optimizes performance by using right amount of power (budget)!
  - Controls power by dynamically throttling the processors
    - Processors usually are the biggest power consumer
- Contributions
  - Control of peak server-level power (to 0.5 W in 1 second)
  - Guaranteed accuracy and stability
  - Verified on real hardware
  - Better application performance than previous methods
Three Options for Power Control

1. **Open-loop:**
   - No measurement of power
   - Use fixed CPU speed to prepare for most power-hungry workloads

2. **Ad-hoc:**
   - Measures power and compares to power budget
   - Steps up/down one level at a time until reach the budget

3. **Control-theoretic:**
   - Throttles CPU based on control theory
   - Guaranteed controller performance

Design of Open-loop

- P4MAX workload used as basis for open-loop controller
- Graph shows maximum 1 second power for workload
- Open-loop improves the traditional design based on label power
- Example label power: 500W for a server

Control-Theoretic Design: Modeling

- Modeling
- System identification: Power changes as frequency changes
- Linear Model
  - \( p(k) = A(k)p + C \)
- Difference model for control with variations
  - \( p(k+1) = p(k) + g A d(k) \)
  - \( g \) is the “slope” and is unknown at design time.
  - \( g \) captures the model variations.

Design of Proportional Controller

- System model
  - \( p(k+1) = p(k) + g A d(k) \), where \( g \) is the “slope”.
- Design process is very similar to that of the utilization controller
  1. Assume \( g = 1 \) since \( g \) is unknown
  2. Transform to Z-domain for easier control analysis:
     \[
     Ac(z) = \frac{P(z)}{1-z^{-1}}
     \]
  3. Derive closed-loop transfer function:
  4. Minimal prototype to achieve stability, zero steady-state error, and etc:
  5. Controller transformed back to the time domain:

Control-Theoretic Design: Analysis

- Quantitative way to analyze workload uncertainties: what if \( g \neq 1 \)?
  - Conduct analysis by substituting \( C(z) \) into the real system model with \( g \).
  - Pole as a function of \( g \) must locate inside unit circle.
  - Stability: \( 0 < g < 2 \)
  - Steady state error: guaranteed to be 0
  - Settling time: a function of \( g \).

Controller Implementation

- IBM BladeCenter HS20 blade server with Intel Xeon processors
- Standard benchmarks: SPEC and Linpack
- Xeon supports only 8 discrete freq levels
  - How to approximate the desired level (continuous) from the controller?
  - For 3.2GHz, use 3, 3, 3, 3, 4 on a smaller timescale (subintervals)
Comparison of Control Accuracy

• Ad-hoc: steps up/down one level at a time
• Ad-hoc has a positive steady-state error of 4 W

Average is 215 W

Performance Comparison

• P controller can precisely achieve all the set points
• Ad-hoc often has positive steady-state errors
• May cause undesired server shutdown in real systems
• Improved ad-hoc: Add safety margin of 6.1 W to ad-hoc

Conclusions

• Power is a system resource and must be controlled.
  • Power is no longer just the accidental result of computation.
  • Reduce power supply capacity, safely, for lower costs.
  • Relax design-time constraints, enforce run-time constraints.
  • Install more servers per rack.
• This paper presents a server-level power controller
  • Designed based on control theory for theoretical guarantees.
  • Outperforms two solutions commonly used in industry.

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  ➔ 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]

Cluster-level Power Control

- Cluster-level power control
  - Servers are correlated together
  - Shared power supply
- Multi-Input-Multi-Output (MIMO) control
  - Controlled variable: Total power of the rack
  - Manipulated variables: Processor DVFS level
  - Control algorithm: Model Predictive Control (MPC)

Power Control Loop

- Control loop invoked periodically
  1. Power meter sends the total power to the controller
  2. CPU utilization monitors send utilizations to the controller
  3. Controller conducts constrained MPC computation
  4. 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:
\[
\min_{f(k)} (P_s - P_{t,p}(k))^2
\]
subject to constraints:
\[
F_{\min,j} \leq f_j(k) \leq F_{\max,j} \quad (1 \leq j \leq N)
\]
\[
f(k) = f_j(k)
\]
\[
\|q(k)\| \leq P_s
\]

System Modeling and Verification

- Modeling
  - System identification
  - Power changes as frequency changes
- Linear Model
  - \( P(k) = A \cdot f(k) + C \)
- Model verification
  - White noise input
- Difference model for control
  - \( P(k+1) = P(k) + A \cdot d(k) \)
  - where \( d(k) = f(k+1) - f(k) \)

Modeling Workload Variations

- Model is inaccurate when the controller is used for
  - A different server or a different workload
- Actual system model
  - \( P(k+1) = P(k) + g \cdot \Delta \cdot d(k) \)
  - where \( g \) is unknown at design time and captures the model variations
- Actual model of whole cluster
  - \( y(k+1) = q(k) \cdot g \cdot \Delta \cdot d(k) \)
Model Predictive Controller Design

Core of MPC is a constrained optimization solver

Least Squares Solver

System Model

Constraints

Frequency changes

$d_i(k+1)$

$P_d(k)$

$P_f(k)$

$P_u(k)$

Diff with reference trajectory

Diff with max performance

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: ±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 = 100µs/100ms = 0.1%

First Baseline – Ad Hoc

  - 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)

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

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
    - $0 < g < 8.8$: slope of the real model is 8.8 times of nominal model
  - Assuming one server has variations while others don’t
    - $0 < g_1 < 42.1$ and $0 < g_2, g_3, g_4 < 29.9$
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 small SM: 0.5 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

Comparison of the Two Papers

<table>
<thead>
<tr>
<th></th>
<th>Server-level</th>
<th>Cluster-level</th>
</tr>
</thead>
<tbody>
<tr>
<td>What power?</td>
<td>Power of a single server</td>
<td>Power of a group of servers in a rack</td>
</tr>
<tr>
<td>Control algorithm</td>
<td>SISO, P controller</td>
<td>MIMO, MPC controller</td>
</tr>
<tr>
<td>Knob</td>
<td>Clock modulation</td>
<td>DVFS</td>
</tr>
<tr>
<td>Power shifting?</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>