A Framework for Providing Quality of Service in Chip Multi-Processors

Fei Guo[1], Yan Solihin[1], Li Zhao[2], Ravishankar Iyer[2]

[1]North Carolina State University [2]Intel Corporation

Background

- Chip Multi-Processor (CMP) is mainstream architecture
  - Some platform resources (cache, bandwidth) are shared
  - Resource sharing leads to contention
  - Contention may result in a large performance variation
- Future uses of CMP
  - Run diverse applications with diverse requirements
  - Require performance **Quality of Service (QoS)**

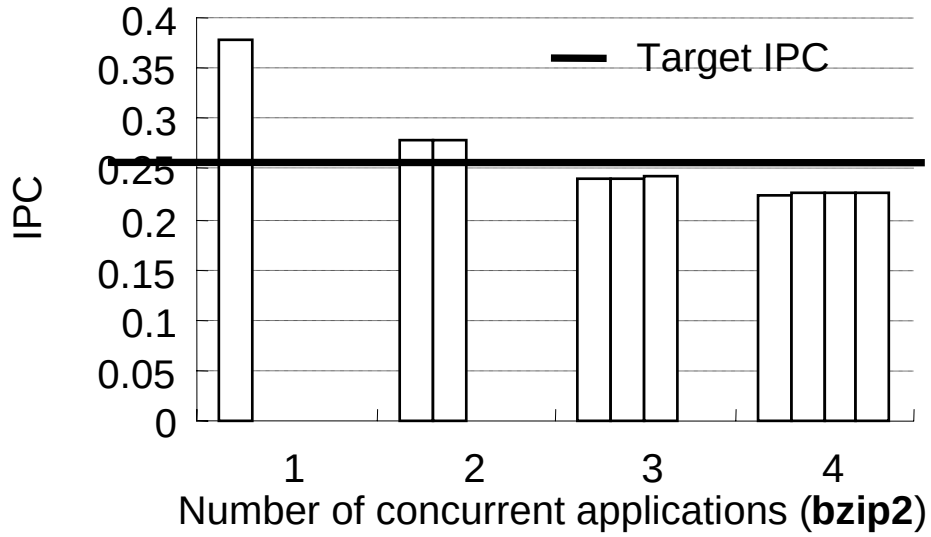
Related Work

- Previous QoS frameworks [Iyer04][Yet05] [Hsu06] [Rafique06] [Nebit07][Iyer07]
  - QoS target specified as
    - IPC or miss rate
  - Resource partitioning (cache, off-chip bandwidth)
  - Resource manager
    - Allocate resource to reach all applications' QoS targets

- Previous QoS frameworks do not *fully* provide QoS

Problems with Previous Frameworks

- 4-core CMP



- QoS targets not met when > 2 jobs run simultaneously
  - CMP cannot check if available resources are sufficient
  - CMP does not know when to reject jobs

Contributions

A framework to provide QoS in a CMP

- **Appropriate QoS target**
  - Allowing admission control policy
- **QoS execution modes**
  - Important for flexibility and throughput
- **Safe throughput optimization techniques**
  - Preserving QoS
  - QoS execution mode downgrade
  - Resource stealing

Outline

- QoS target specification
- QoS execution modes
- Resource stealing
- Evaluation
- Conclusions

QoS Targets for Individual Jobs

- Performance Metrics
  - IPC or cache miss rate
- Resource Usage Metrics (**RUM**)
  - Cache size, bandwidth rate
  - Easily comparable
    - Foundation for constructing admission control
  - Cannot be ill-defined
  - More familiar to the users

Timeslot Resource

- *maximum wall-clock time*
  - Borrowed from batch job systems
- *Deadline*: latest expected completion time
  - Soft deadline
- Timeslot specification is *optional*

QoS Execution Modes

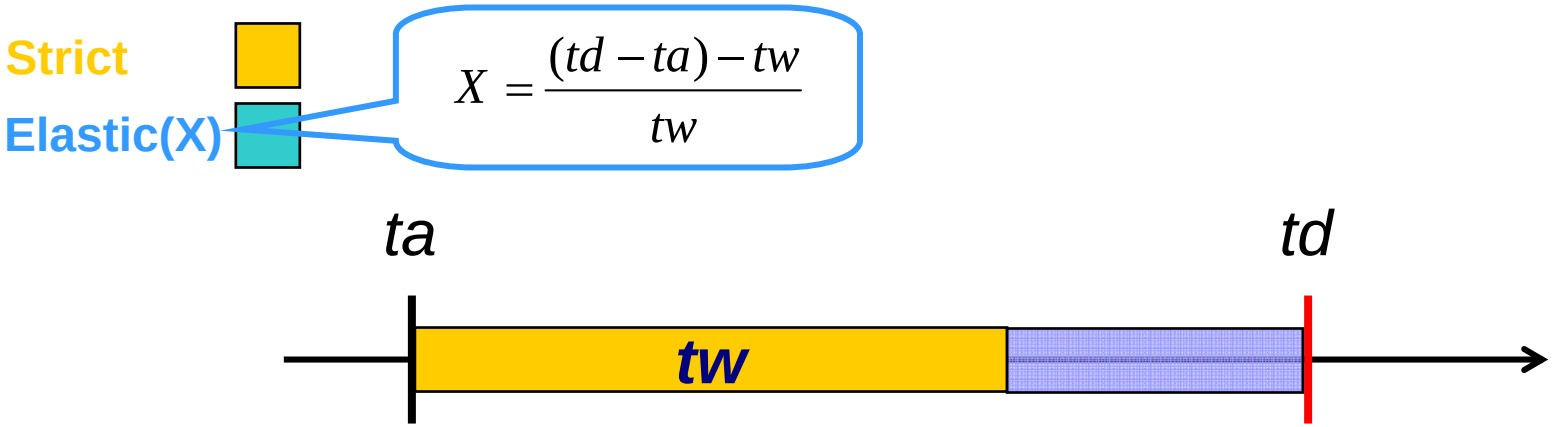
- Provide various strictness levels in meeting QoS targets
- **Strict:**
  - Rigid "implied" throughput and deadline requirements
    - Resources and timeslot must be strictly reserved
- **Elastic(X):**
  - Rigid deadline requirement
  - Can tolerate throughput deviation (X% max slowdown)
- **Opportunistic:**
  - Flexible throughput and deadline requirements

Strong

Weak

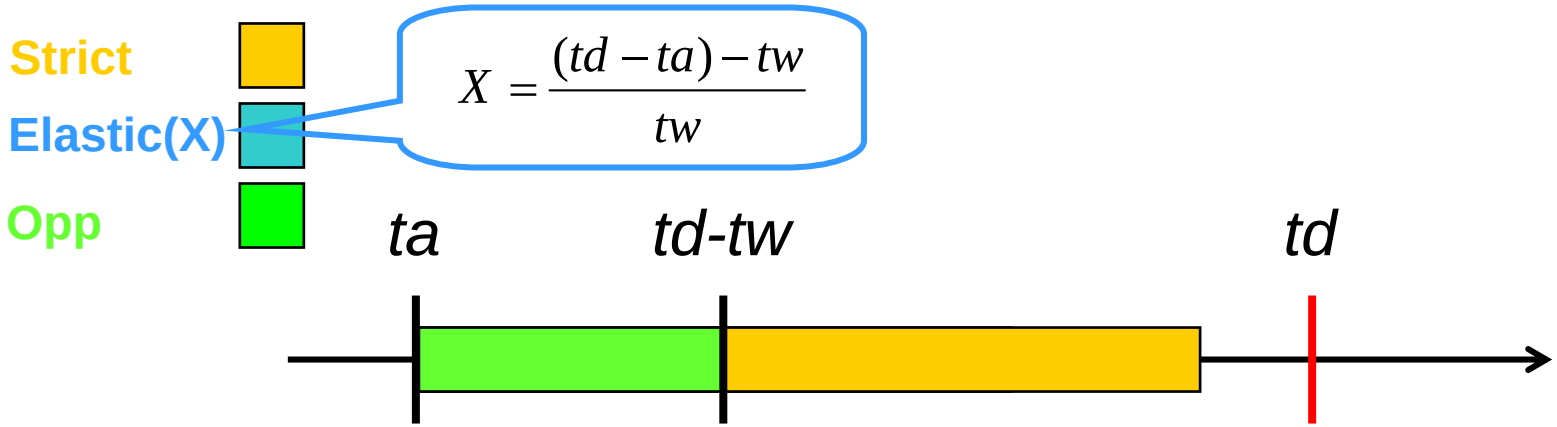
Mode Downgrade

- Manual mode downgrade
  - Requires users to change a job's mode to weaker ones
  - Users fully aware of the consequences
- Automatic mode downgrade
  - Transparent to users
    - Deadlines are preserved
    - Throughput variation tolerable by jobs



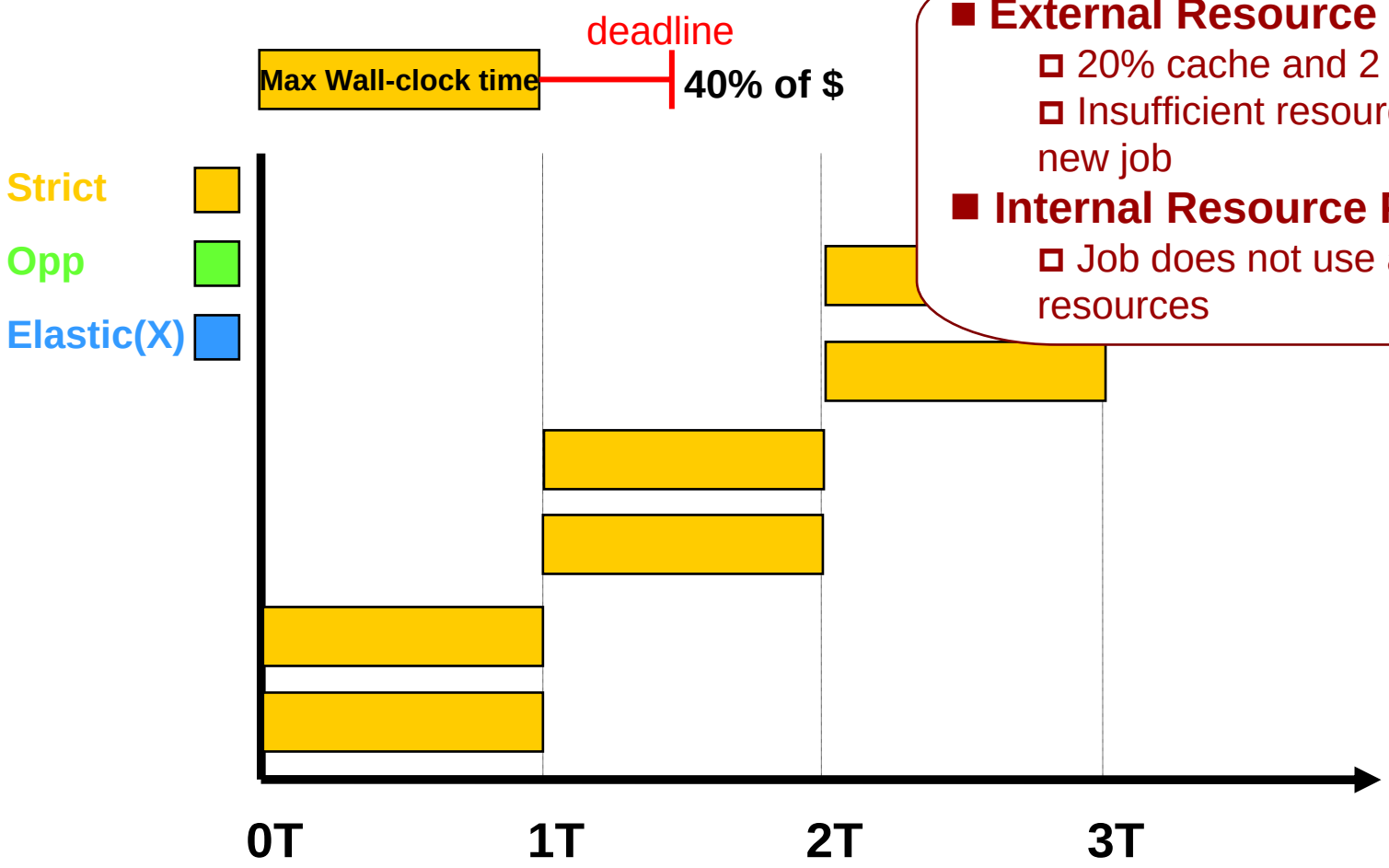
Mode Downgrade

- Manual mode downgrade
  - Requires users to change a job's mode to weaker ones
  - Users fully aware of the consequences
- Automatic mode downgrade
  - Transparent to users
    - Deadlines are preserved
    - Throughput variation tolerable by jobs



Impact of Mode Downgrade

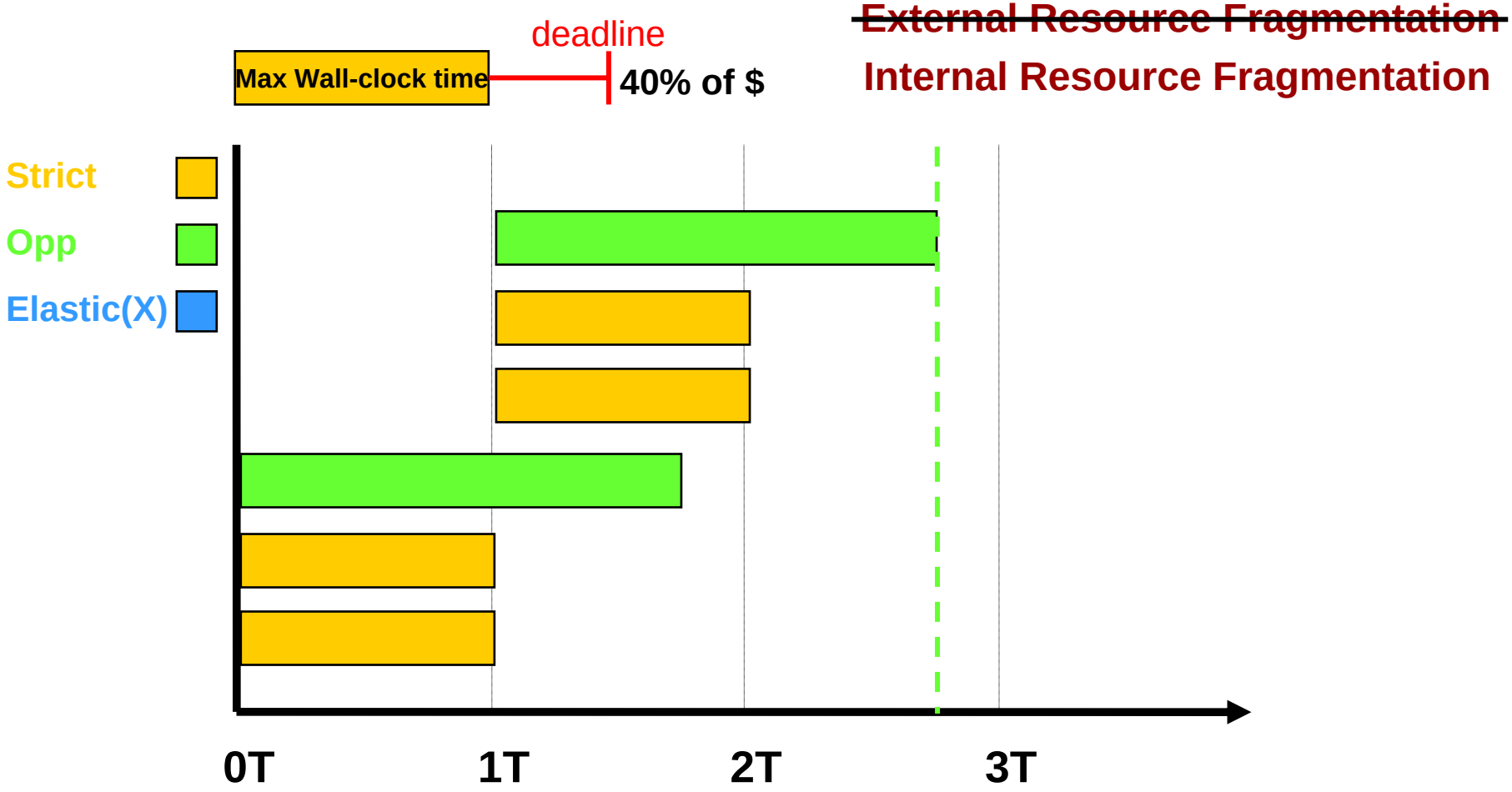
4-core CMP receives jobs. 6 jobs are illustrated.



- **External Resource Fragmentation**
  - 20% cache and 2 cores are unused
  - Insufficient resources to accept a new job
- **Internal Resource Fragmentation**
  - Job does not use all allocated resources

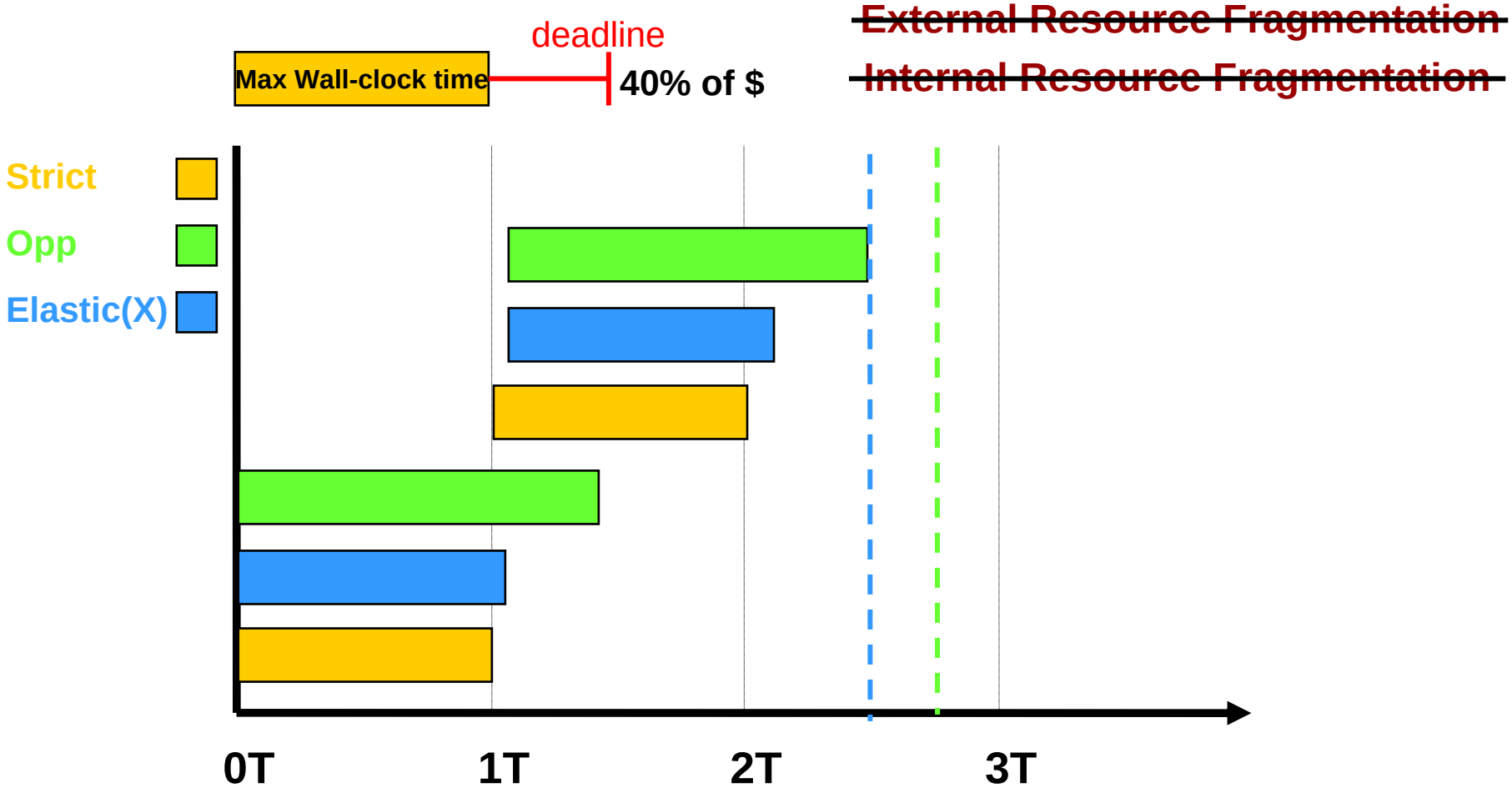
Impact of Mode Downgrade

4-core CMP receives jobs. 6 jobs are illustrated.



Impact of Mode Downgrade

4-core CMP receives jobs. 6 jobs are illustrated.



Cache Capacity Partitioning

- Based on a fine-grain per-set partition scheme [Iyer04][Nesbit07]
  - Job specifies number of cache ways
  - Steal cache capacity = steal cache ways

Resource Stealing (RS) Overview

- In Elastic(X), X = maximum CPI increase
  - Must know CPI with and without RS
  - Only know one of them at any given time
- Observation: CPI components are additive

$$CPI = CPI_{L2->\infty} + h_m t_m$$

  - h_m = L2 miss per instruction
  - t_m = average L2 miss latency
- t_m can be kept constant
- Resource stealing only changes h_m

h_m increase X% => CPI increase < X%

Resource Stealing (RS) Overview

- In Elastic(X), X = maximum CPI increase
  - Must know CPI with and without RS
  - Only know one of them at any given time
- Observation: CPI components are additive

$$CPI = CPI_{L2->\infty} + h_m t_m$$

  - h_m = L2 miss per instruction
  - t_m = average L2 miss latency
- t_m can be kept constant
- Resource stealing only

h_m increase X

- **Monitored through duplicate tags**
  - use original partitions

Evaluation Methodology

- Simulation environment (based on Simics)
  - 4-core CMP running Fedora Core 4 Linux
    - 2MB 16-way shared L2 cache
- Applications
  - Selected from 15 SPEC2006 C/C++ benchmarks
    - bzip2 (Highly cache sensitive)
    - hmmer (Moderately cache sensitive)
    - gobmk (Not cache sensitive)

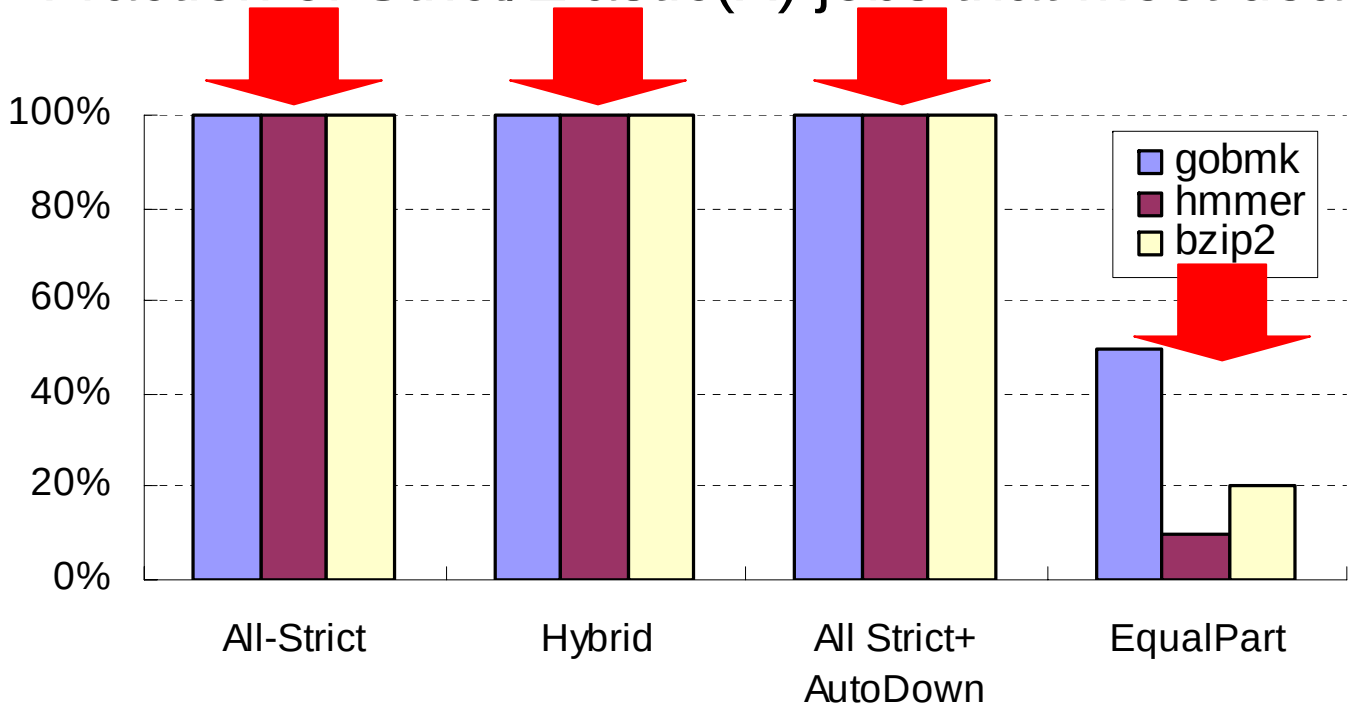
Evaluation Methodology

- Workload composition
  - Workload 1: Ten identical jobs
  - Workload 2: Ten mixed jobs
  - Tight deadline (5 jobs), moderate deadline (3 jobs) and relaxed deadline (2 jobs)

- Execution mode configurations
  - *All-Strict* (**Base**)
  - *Hybrid*: 4 Strict + 3 Elastic(5%) + 3 Opportunistic jobs
  - *All-Strict+AutoDown*: 10 Strict jobs $\rightarrow$ Opportunistic
  - *EqualPart*: Cache equally partitioned among cores
    - No admission control

Impact of Different Modes

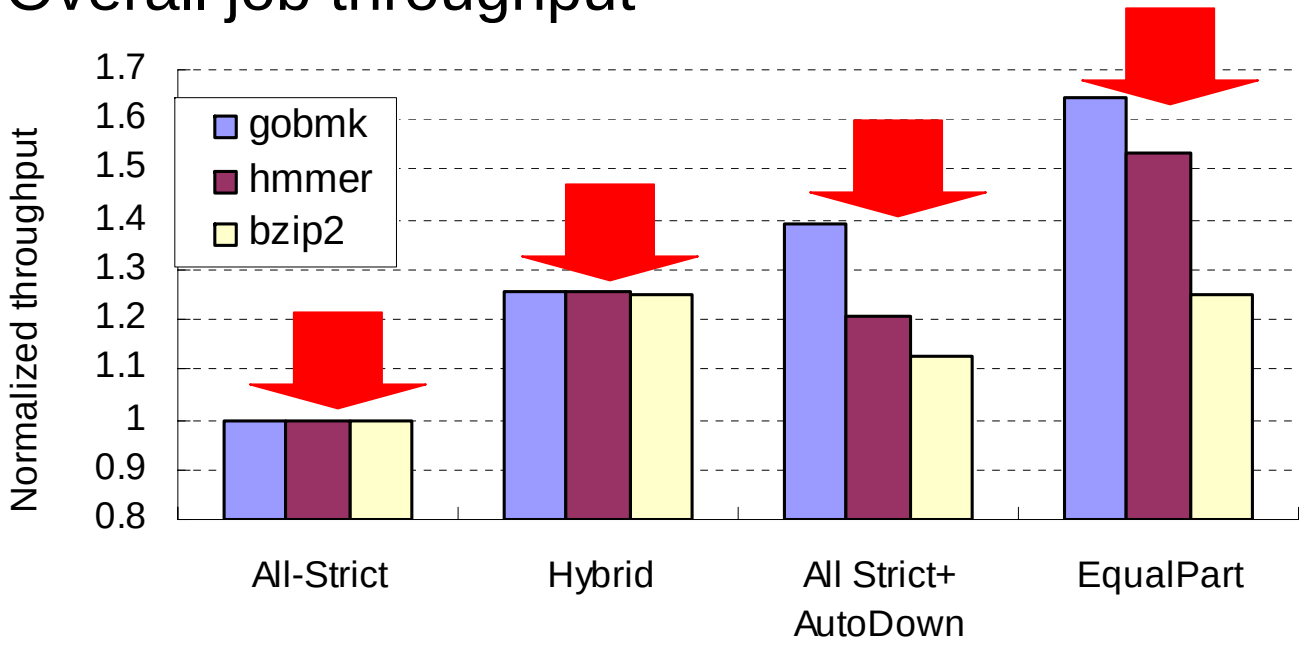
- Fraction of Strict/Elastic(X) jobs that meet deadlines



  - *EqualPart*: most jobs miss deadlines
  - Our schemes: all Strict/Elastic(X) jobs meet deadlines

Impact of Different Modes

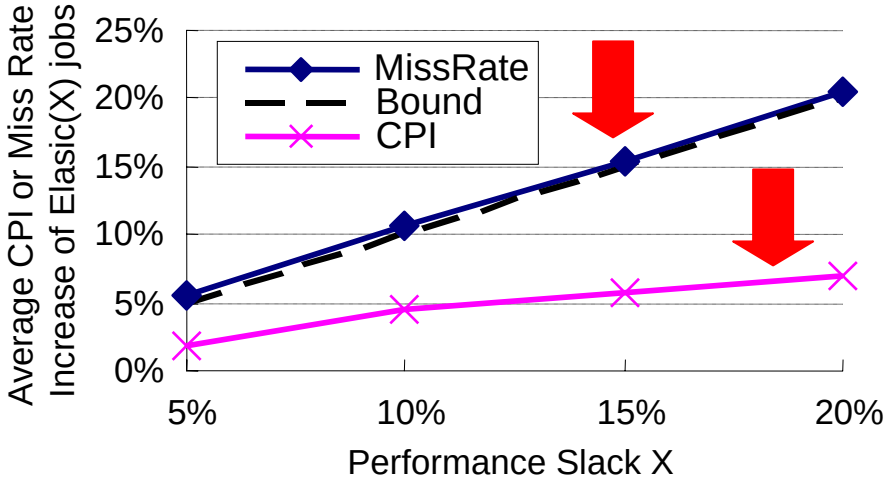
- Overall job throughput



  - Strong QoS – throughput trade-offs
  - Execution mode variety boosts throughput
  - Auto mode downgrade “transparently” boosts throughput
  - Moderate/Relaxed deadlines help

Resource Stealing

- Impact of performance slack X in *Hybrid* case (***bzip2***)



  - Using duplicate tags is effective
  - CPI increase < miss rate increase
  - Miss rate is a safe proxy

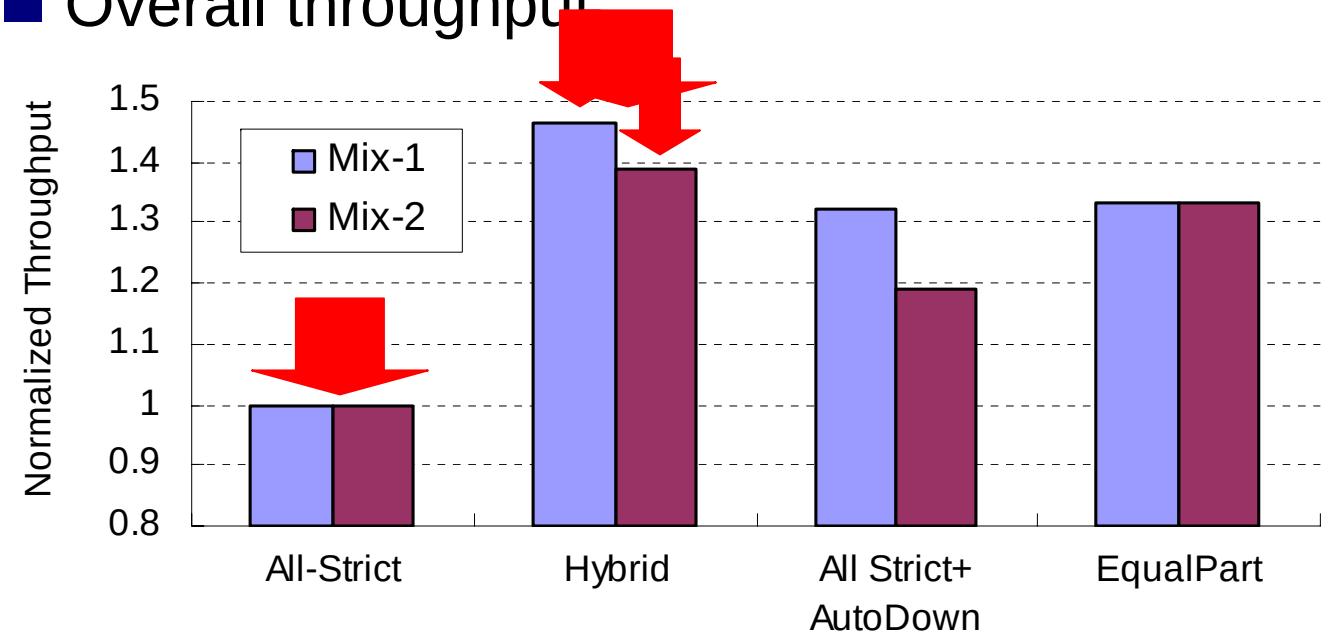
Mixed-Benchmark Workloads

	Mix-1	Mix-2
Strict	*hmmer*	*hmmer*
Elastic(5%)	*gobmk*	*bzip2*
Opportunistic	*bzip2*	*gobmk*

- Mix-1 favorable for resource stealing
  - *bzip2* (cache sensitive) is the recipient
  - *gobmk* (cache insensitive) is the donor
- Mix-2 not favorable for resource stealing

Mixed-Benchmark Workloads

- Overall throughput



- In *Hybrid*, Mix-1 outperforms Mix-2
- Mix-1 can boost throughput up to 46%

Conclusions

- Appropriate QoS target?
  - Resource Usage Metrics (RUM)
  - Allowing admission control policy
- Strong QoS – throughput trade-offs
- Throughput can be safely boosted
  - Significantly (13-46%)
  - Through execution mode downgrade
    - Manually
    - Automatically (transparent to users)
  - Resource stealing effective

Thank You!

Presenter: Fei Guo
fguo@ncsu.edu