Time-sharing Parallel Applications With Performance Isolation and Control

Bin Lin

Ananth I. Sundararaj

Peter A. Dinda

Department of EECS
Northwestern University

http://presciencelab.org

Take-away points

- Designed, implemented, and evaluated a new approach to time-sharing parallel applications with performance isolation
- Approach based on *periodic real-time scheduling* of nodes combined with *global feedback control* of real-time constraints
- Provides a simple way to control execution rate of applications while maintaining efficiency
- Despite only isolating and controlling CPU, memory, comm I/O, and local disk I/O follow

Outline

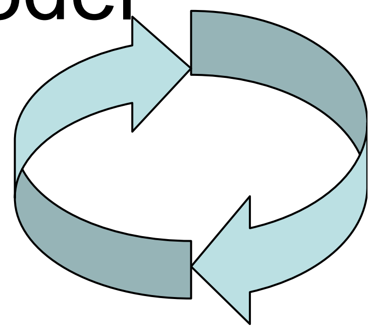
- Batch parallel workload
  - BSP model
  - Challenges
- Periodic real-time scheduling
  - VSched [Lin et al, SC'05]
- Feedback control system
- Evaluation
- Conclusions

Outline

- Batch parallel workload
  - BSP model
  - Challenges
- Periodic real-time scheduling
  - VSched [Lin, SC’05]
- Feedback control system
- Evaluation
- Conclusions

Batch parallel workload

- Use tightly-coupled resources (e.g. cluster)
- Synchronizing collective communication
- Bulk Synchronous Parallel (BSP) model
  - Computation and communication



- If run on time-sharing machines
  - Nodes must be carefully scheduled
  - One thread may stall the whole application

Batch parallel workload (cont.)

- Space-sharing resources to avoid stalls
  - Exclusive resource use
  - Limit utilization; CPU idle during comm or I/O
  - Likely block other processes
  - Coarse control of execution rate & response time
- We propose performance-targetted feedback controlled real-time scheduling.
  - Time-sharing with performance isolation
  - Fine control of execution rate & response time
  - Resource utilization proportional to execution rate

Outline

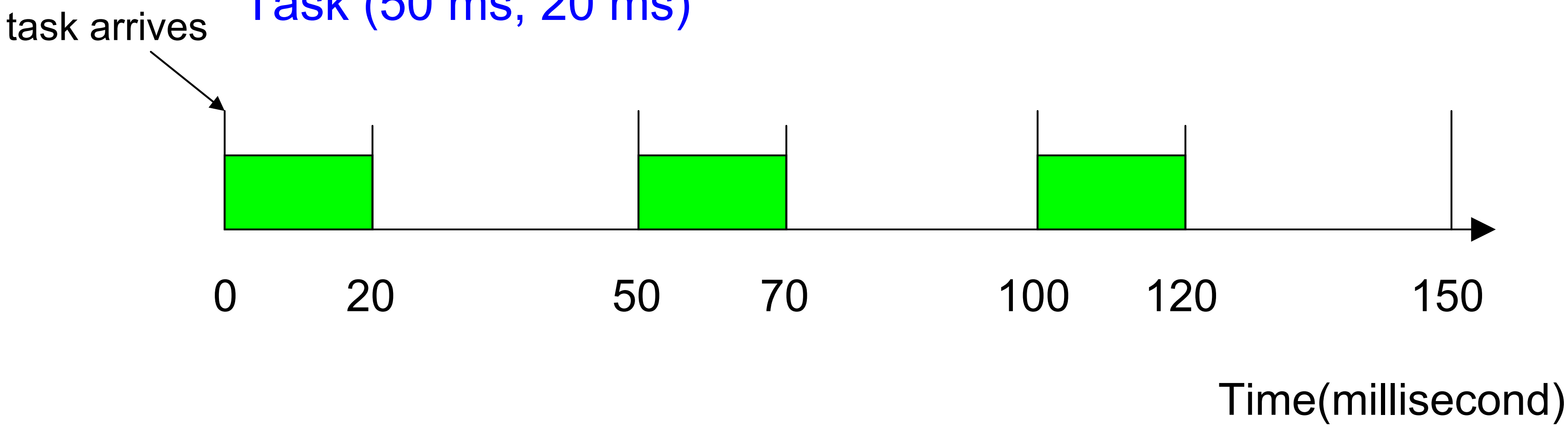
- Batch parallel workload
  - BSP model
  - Challenges
- Periodic real-time scheduling
  - VSched [Lin, SC'05]
- Feedback control system
- Evaluation
- Conclusions

Periodic Real-time Scheduling Model

•Task runs for slice seconds every period seconds
[JACM 1973]

(period, slice) Unit: millisecond

Task (50 ms, 20 ms)



Periodic Real-time Scheduling Model

- Task runs for slice seconds every period seconds
  - "1 hour every 10 hours", "1 ms every 10 ms"
    - Does NOT imply "1 hour chunk" (but does not preclude it)
  - Compute rate: *slice / period*
    - 10 % for both examples, but radically different interactivity!
  - Completion time: *size / rate*
    - 24 hour job completes after 240 hours
- Unifying abstraction for diverse workloads

EDF Online Scheduling

- Dynamic priority preemptive scheduler
- Always runs task with highest priority
- Tasks prioritized in reverse order of impending deadlines
  - Deadline is end of current period

EDF="Earliest Deadline First"

VSched tool

- Provides soft real-time (limited by Linux)
- Runs at user-level (no kernel changes)
- Schedules any set of processes
- Supports very fast changes in constraints

[Lin et al, SC'05]

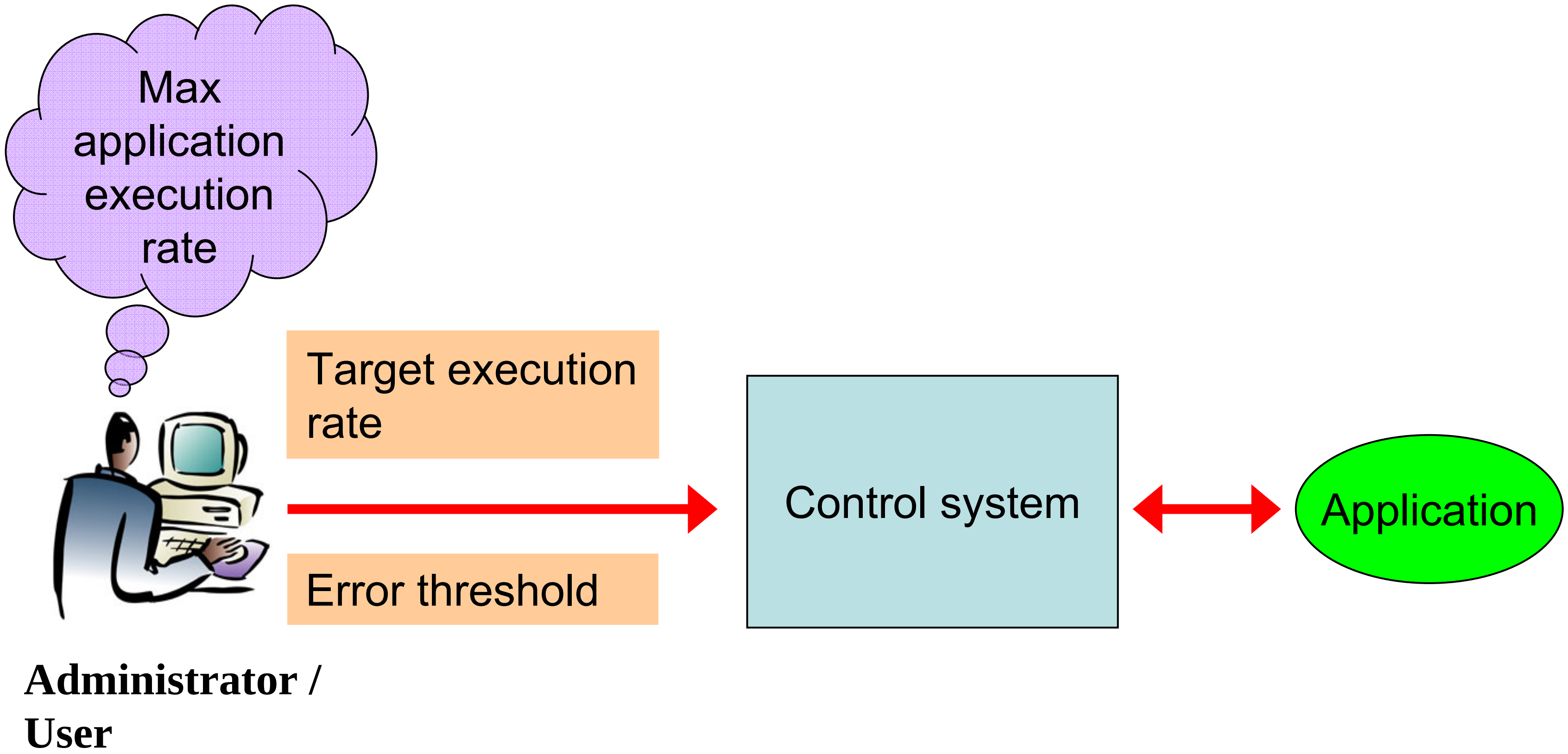
VSched tool

- Supports (slice, period) ranging into days
  - Fine millisecond and sub-millisecond ranges for interactive processes
  - Coarser constraints for batch processes
- Client/Server: remote control scheduling
- Publicly released *http://virtuoso.cs.northwestern.edu.*

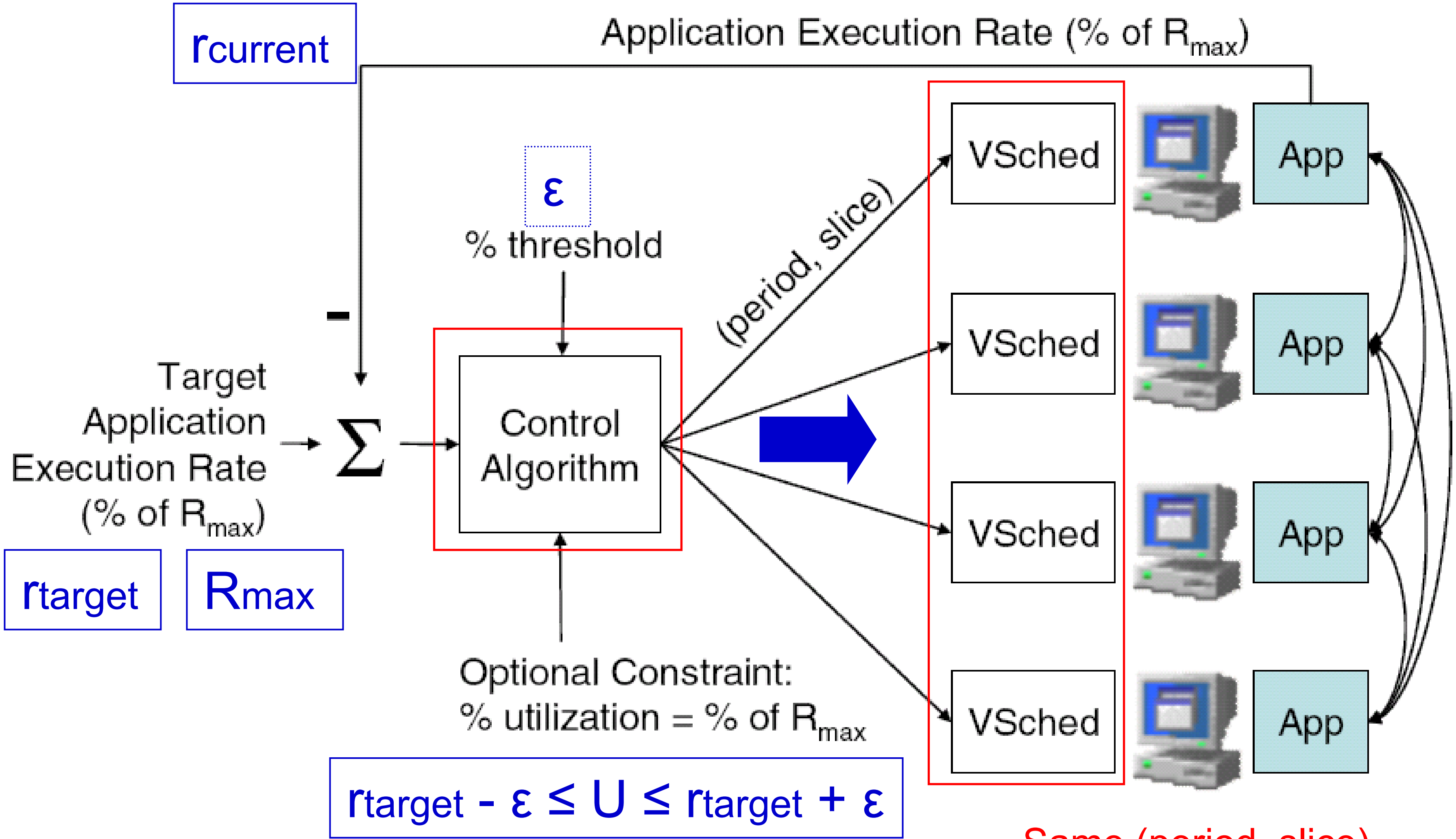
Outline

- Batch parallel workload
  - BSP model
  - Challenges
- Periodic real-time scheduling
  - VSched [Lin, SC'05]
- Feedback control system
- Evaluation
- Conclusions

Overview


Max application execution rate
Target execution rate
Error threshold
Control system
Application
Administrator / User

Our control system



Same (period, slice) constraint is used for each VSched

Input

- R_{max}: max app execution rate
- r_{target}: set point; % R_{max}; supplied by user or system admin
- $r_{current}$: feedback input; current app execution rate; % R_{max}
- ε: error threshold; %
- U: current utilization; slice/period
- $r_{target} - \varepsilon \le U \le r_{target} + \varepsilon$: optional input from user

Control algorithm

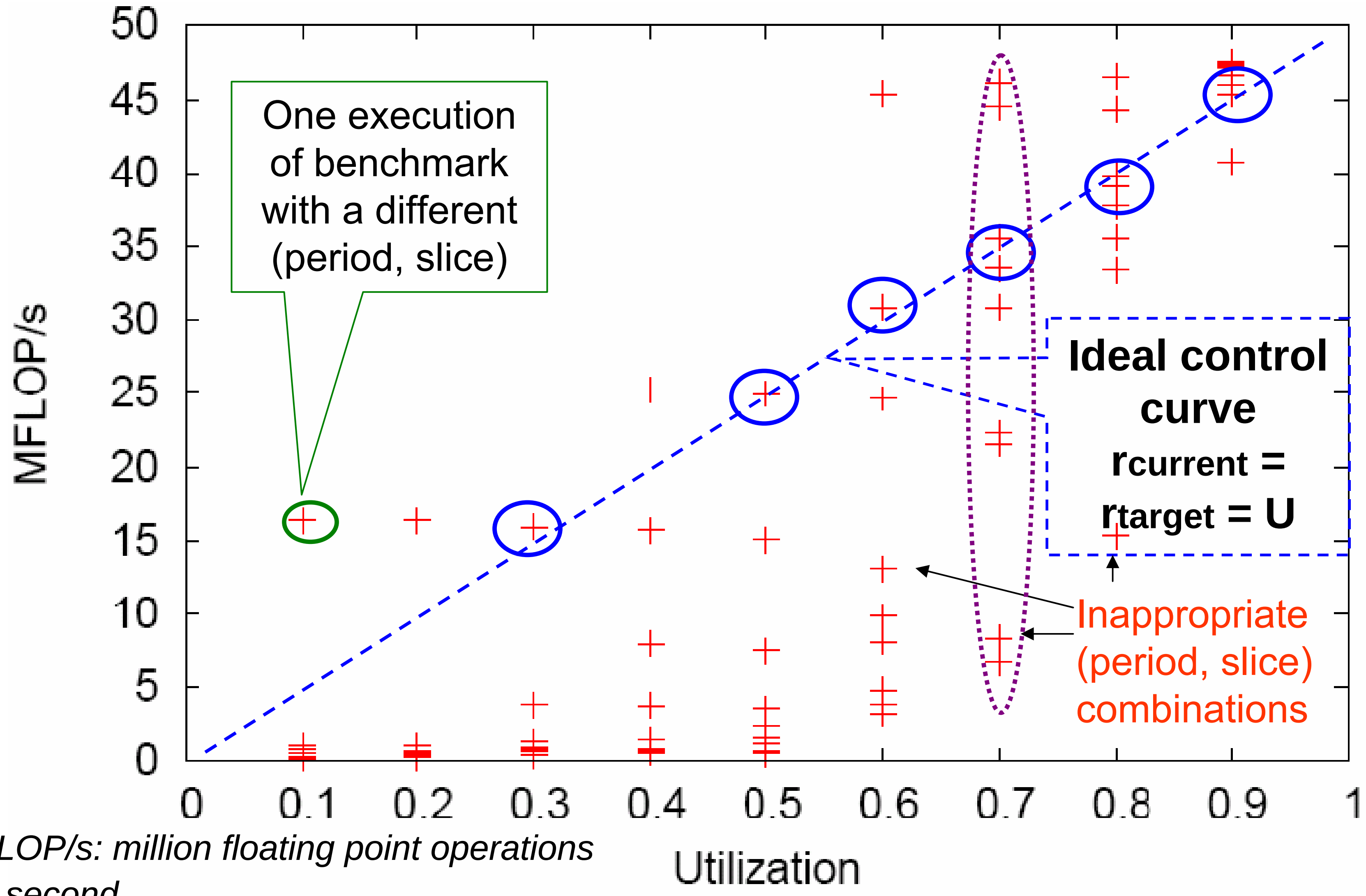
- Define error $e = r_{current} - r_{target}$
- Goal
  - Error is within threshold: $|e| \leq \varepsilon$
  - Schedule is efficient: $U = r_{current} \pm \varepsilon$

Control algorithm

- Define error $e = r_{current} - r_{target}$
- Goal
  - Error is within threshold: $|e| \leq \varepsilon$
  - Schedule is efficient: $U = r_{current} \pm \varepsilon$

- Multiple (period, slice) schedules exist for a given utilization U

Multiple "best" (*period, slice*)s that achieve desired utilization

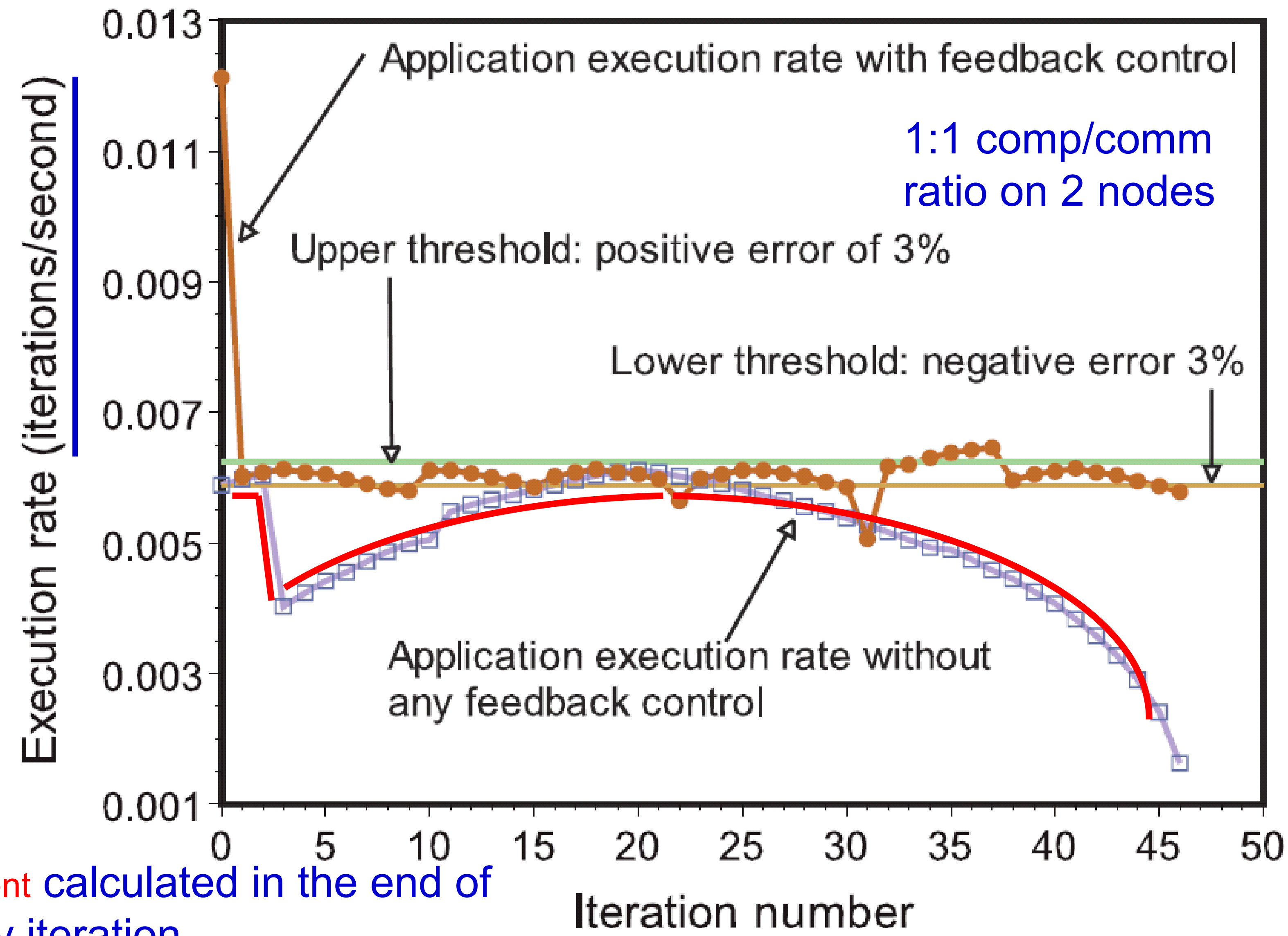


MFLOP/s: million floating point operations per second

Using only local schedulers is not enough

- Best schedule is application dependent
  - Differing comp/comm ratios, granularities, and communication patterns
  - Making the right choice should be automatic.
- User or system admin may want to dynamically change app execution rate.
  - System should react automatically.
- Soft local real-time scheduler
  - Deadline misses will inevitably occur, causing timing offsets b/w app threads to accumulate.
  - Must monitor & correct for these slow errors.

Schedule selection and drift

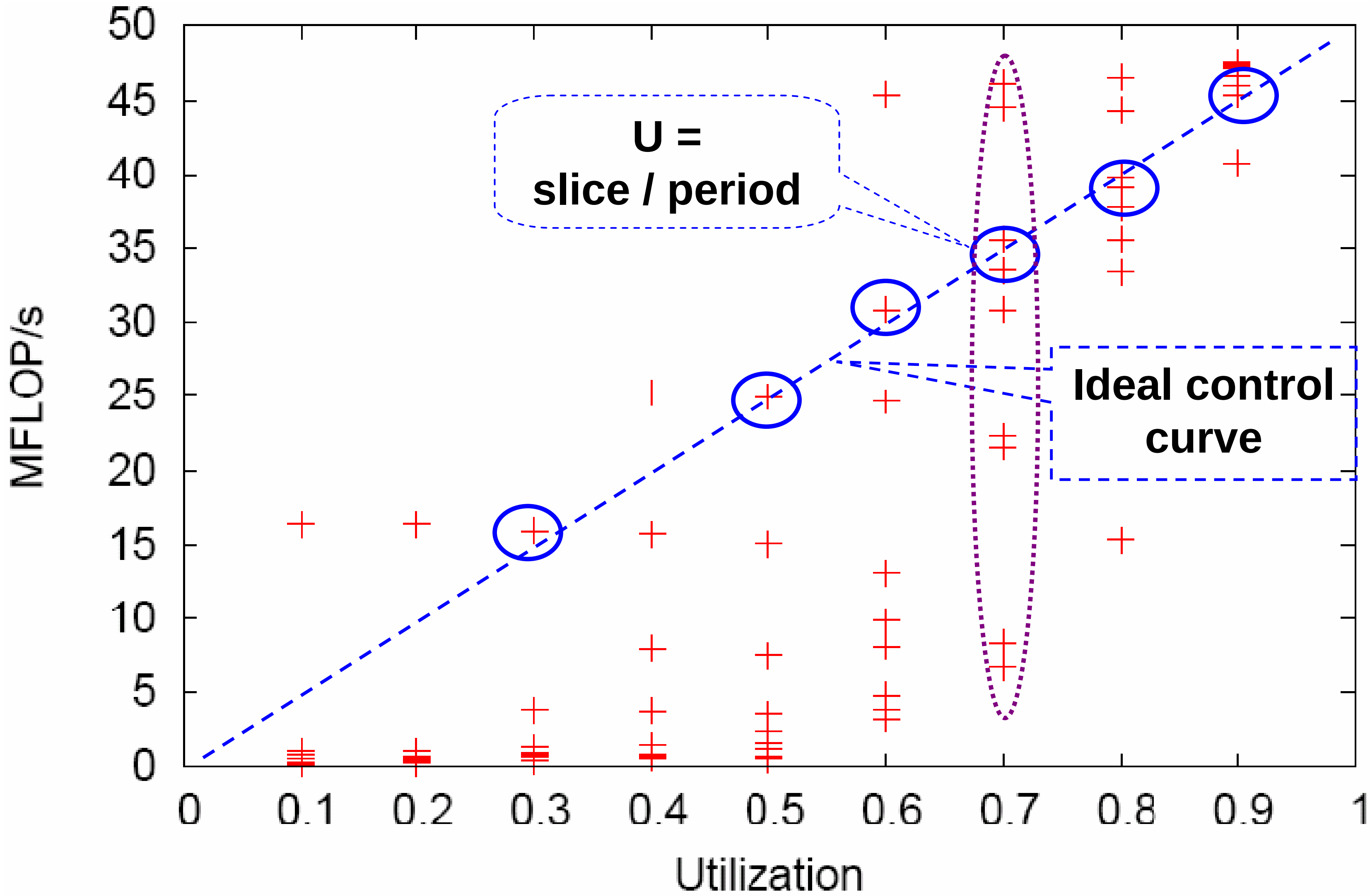


$r_{current}$ calculated in the end of every iteration

Control algorithm (cont.)

- Define error $e = r_{current} - r_{target}$
- Goal
  - Error is within threshold: $|e| \leq \varepsilon$
  - Schedule is efficient: $U = r_{current} \pm \varepsilon$
- If $|e| > \varepsilon$, decrease period by Δ_{period} and decrease slice by Δ_{slice}, such that $U = r_{target}$
  - Startup period 200ms; if period $\leq min_{period}$, reset period
- If $|e| \leq \varepsilon$, do nothing
- Simple linear search
  - Maintains U and searches (period, slice) space from larger to smaller granularity

Multiple “best” (*period, slice*)s that achieve desired utilization



Outline

- Batch parallel workload
  - BSP model
  - Challenges
- Periodic real-time scheduling
  - VSched [Lin, SC'05]
- Feedback control system
- Evaluation
- Conclusions

Evaluation framework

- IBM e1350 cluster (Intel Xeon 2.0 GHz, 1.5 GB RAM, Gigabit Ethernet interconnect, Linux 2.4.20)
- BSP benchmark; *Patterns*; all-to-all communication
- NAS benchmark; IS (integer sort)
- Each node runs VSched, and a separate node runs the controller.

Evaluation framework

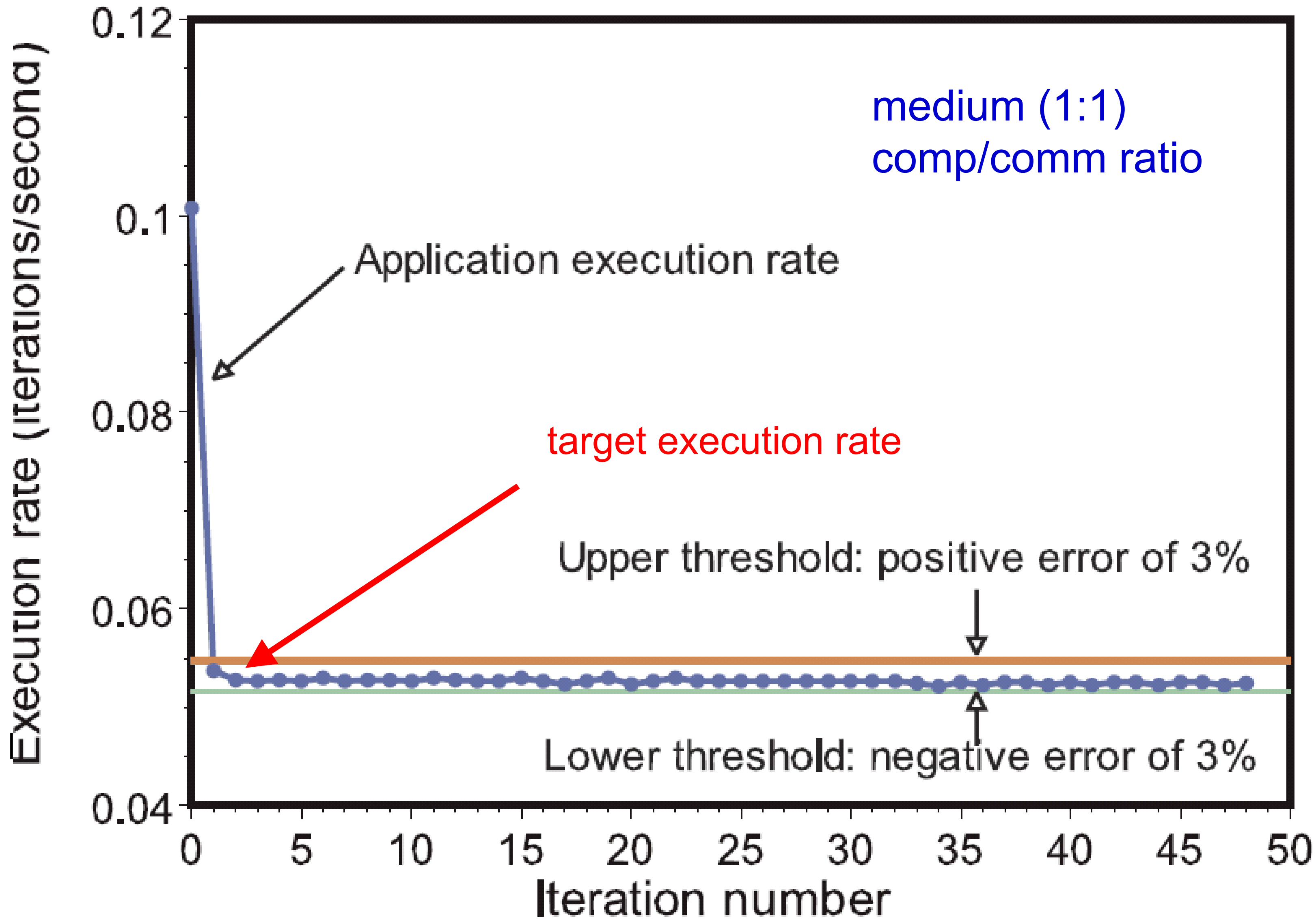
- IBM e1350 cluster (Intel Xeon 2.0 GHz, 1.5 GB RAM, Gigabit Ethernet interconnect, Linux 2.4.20)
- BSP benchmark; *Patterns*; all-to-all communication
- NAS benchmark; IS (integer sort)
- Each node runs VSched, and a separate node runs the controller.

Evaluating control algorithm

- Three comp/comm ratios
  - high (5:1) ratio, medium (1:1) ratio, and low (1:5) ratio
- Different r_{target} (% of R_{max})
- Different error threshold ε

- Δ_{period} = 2ms, Δ_{slice} adjusted such that $U = r_{target}$

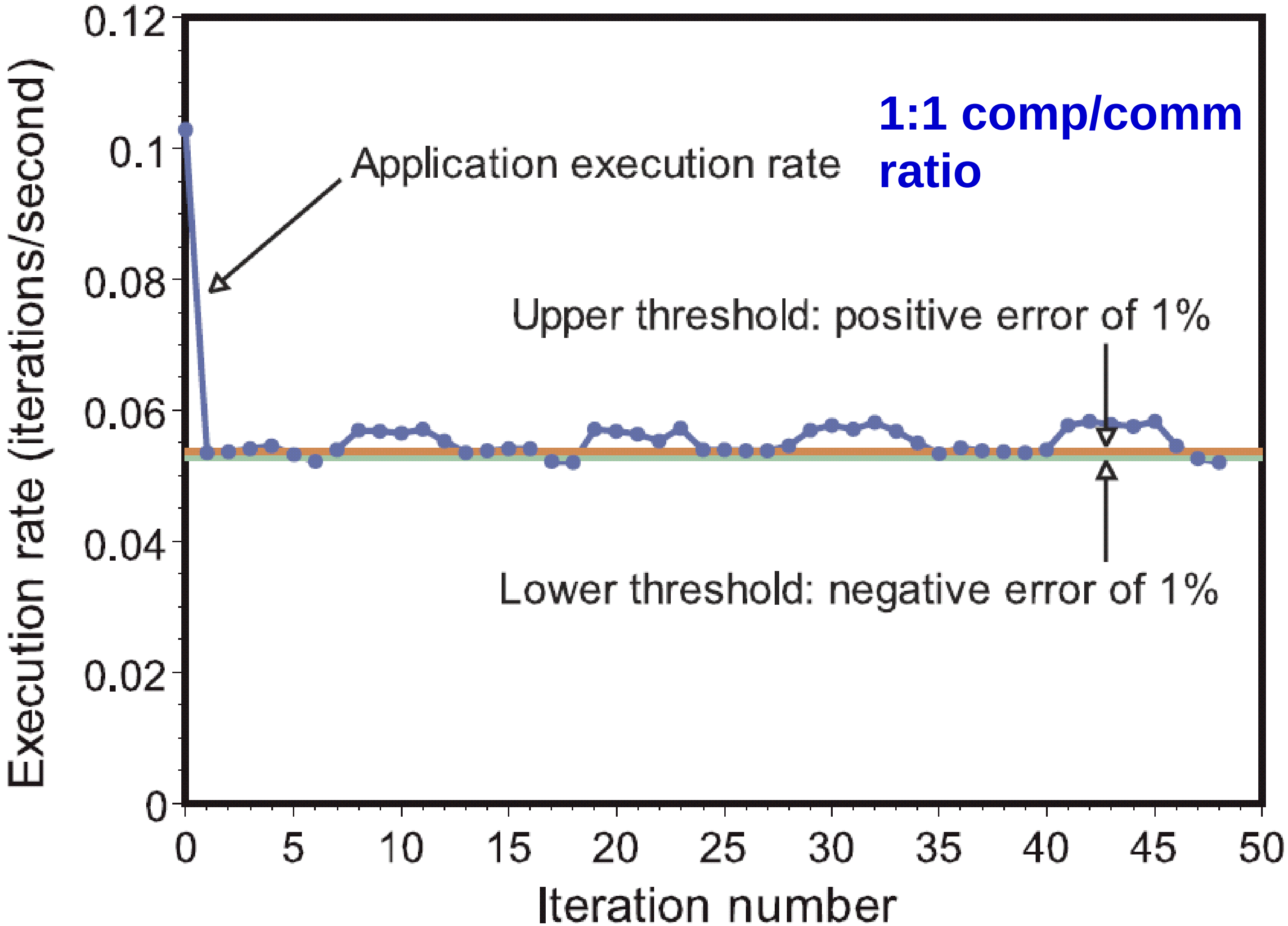
Quick and stable control of app execution rate



Evaluating control algorithm (cont.)

- Three comp/comm ratios
  - high (5:1) ratio, medium (1:1) ratio, and low (1:5) ratio
- Different r_{target} (% of R_{max})
- Different error threshold ε
  - Minimum threshold: the smallest ε below which control becomes unstable
- Δ_{period} = 2ms, Δ_{slice} adjusted such that $U = r_{target}$

System in oscillation when error threshold is too small

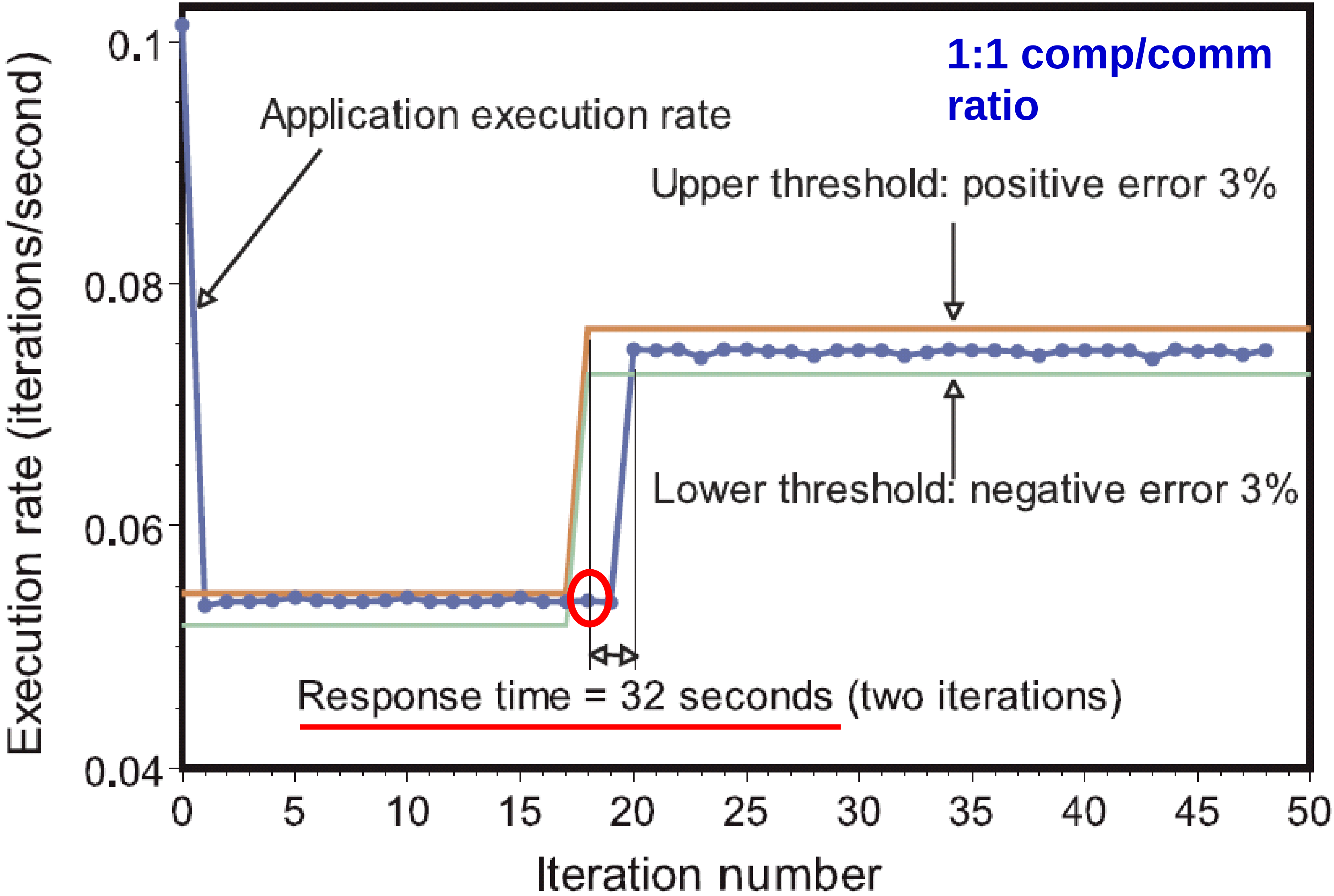


While system is oscillating, it appears to degrade gracefully.

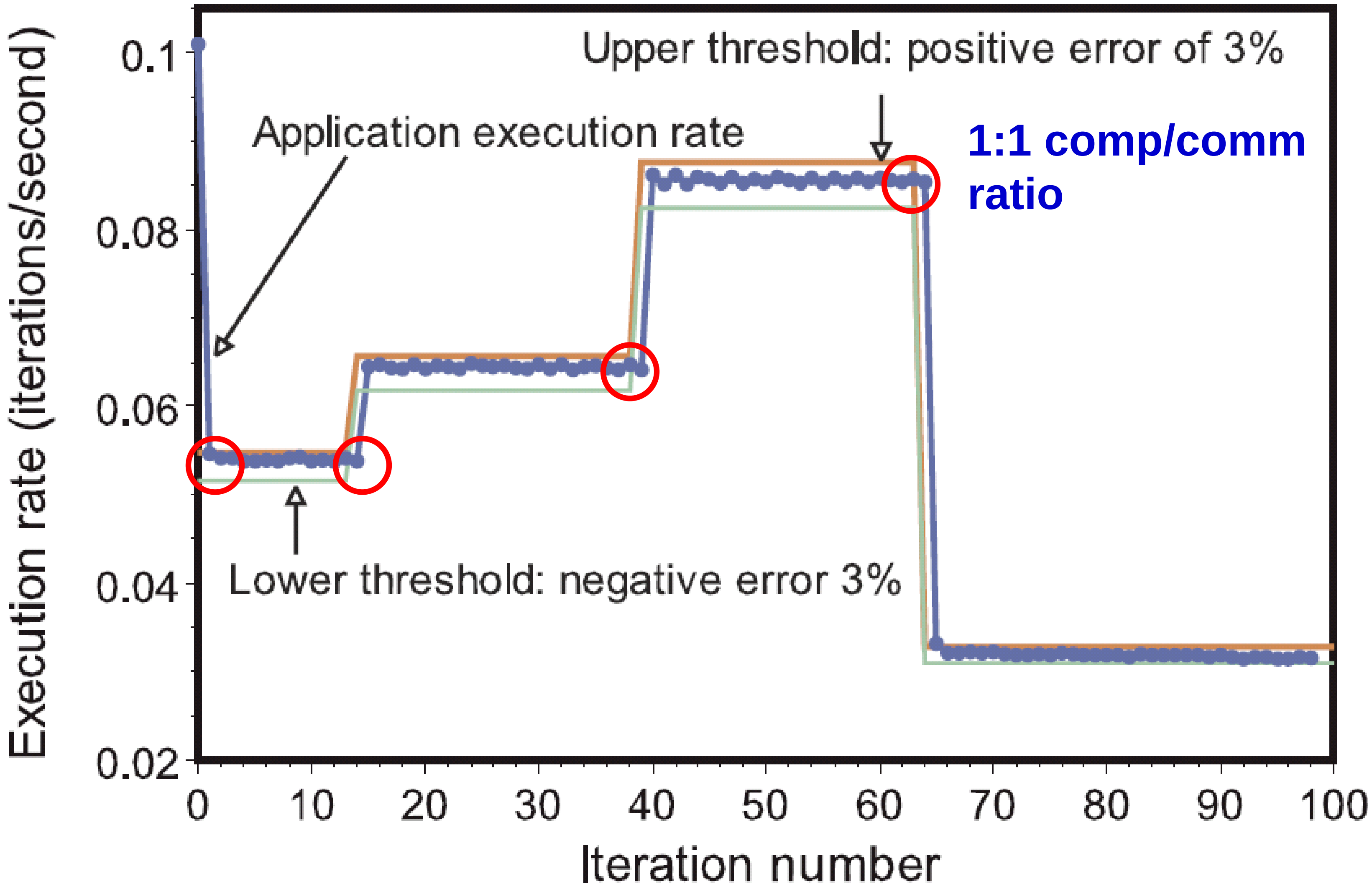
Evaluating control algorithm (cont.)

- Three compute/communicate ratios
  - high (5:1) ratio, medium (1:1) ratio, and low (1:5) ratio
- Different r_{target}
- Different error threshold ε
  - Minimum threshold: the smallest ε below which control becomes unstable
- Δperiod = 2ms, Δslice adjusted such that U= r_{target}
- Response time
  - for stable configurations, time between when r_{target} changes and when $r_{current} = r_{target} \pm \varepsilon$

Response time of control algorithm



Dynamically varying execution rates



Summary of alg limits on our testbed & benchmarks

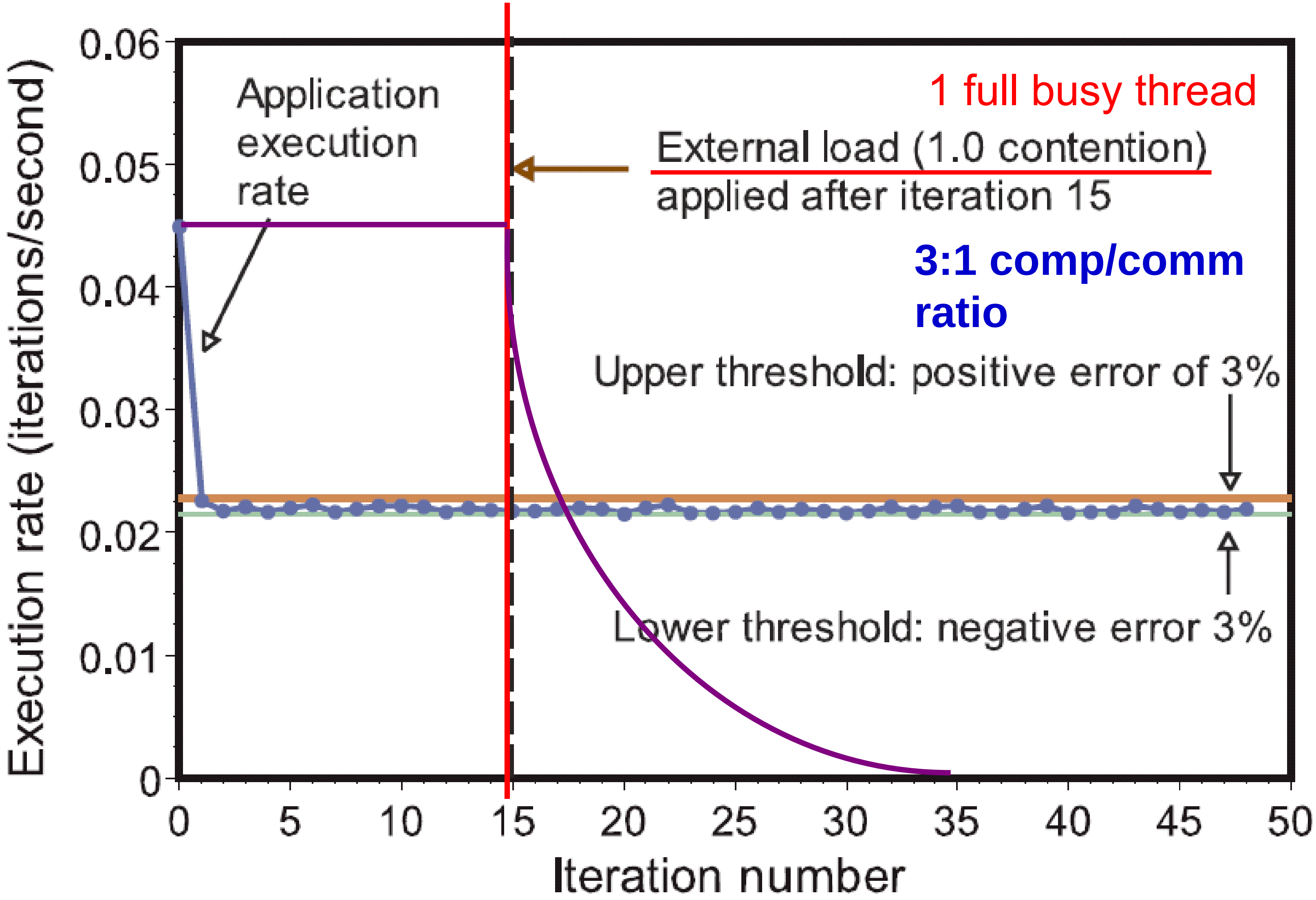
High (5:1) compute/communicate ratio		
Response time	Threshold limit	Feedback comm. cost
29.16 s	2 %	32 bytes/iter

Medium (1:1) compute/communicate ratio		
Response time	Threshold limit	Feedback comm. cost
31.33 s	2 %	32 bytes/iter

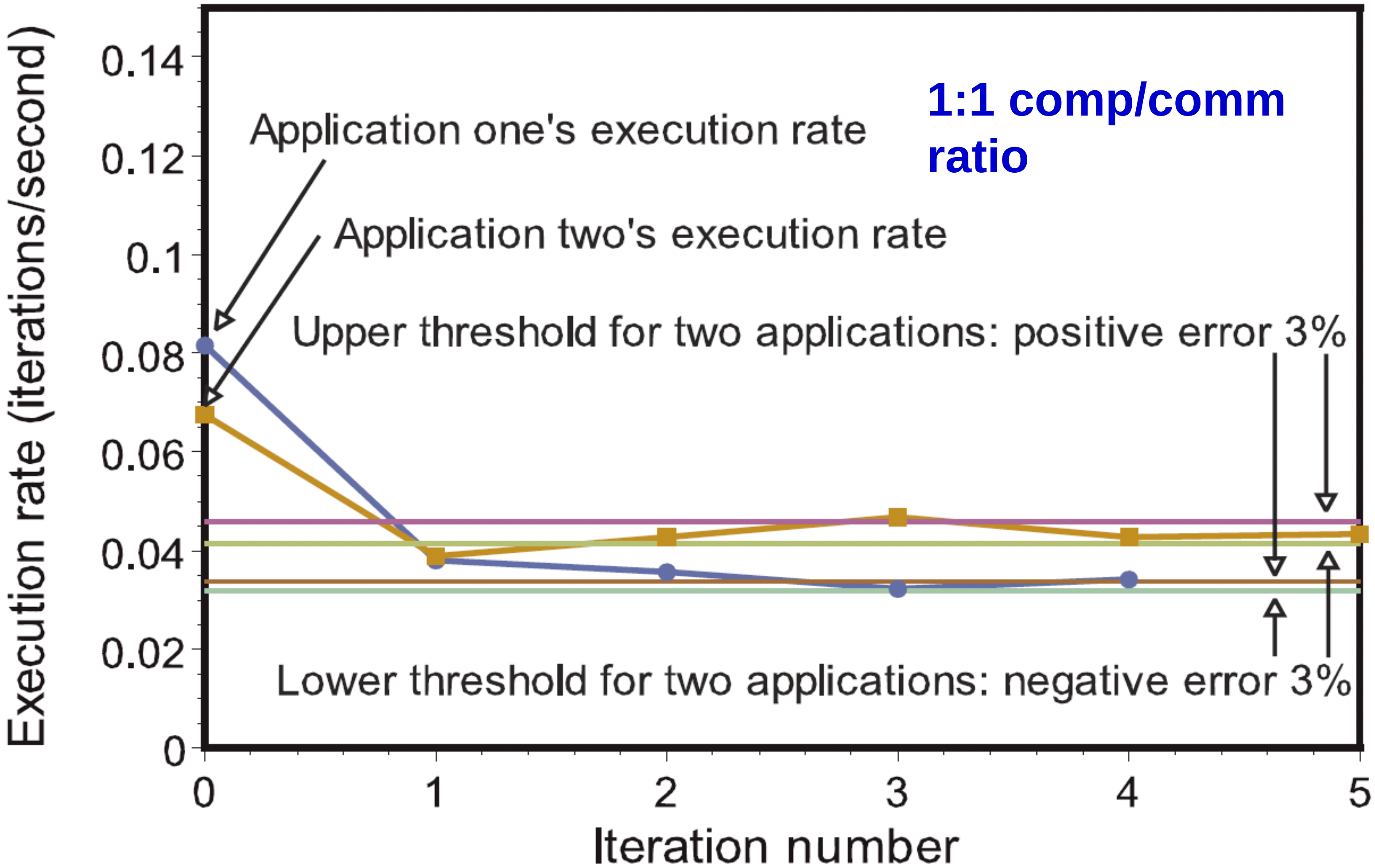
Low (1:5) compute/communicate ratio		
Response time	Threshold limit	Feedback comm. cost
32.01 s	2 %	32 bytes/iter

- Small error threshold
- Low response time
- Tiny communication cost
- Results largely independent of comp/comm ratio

Ignore external load

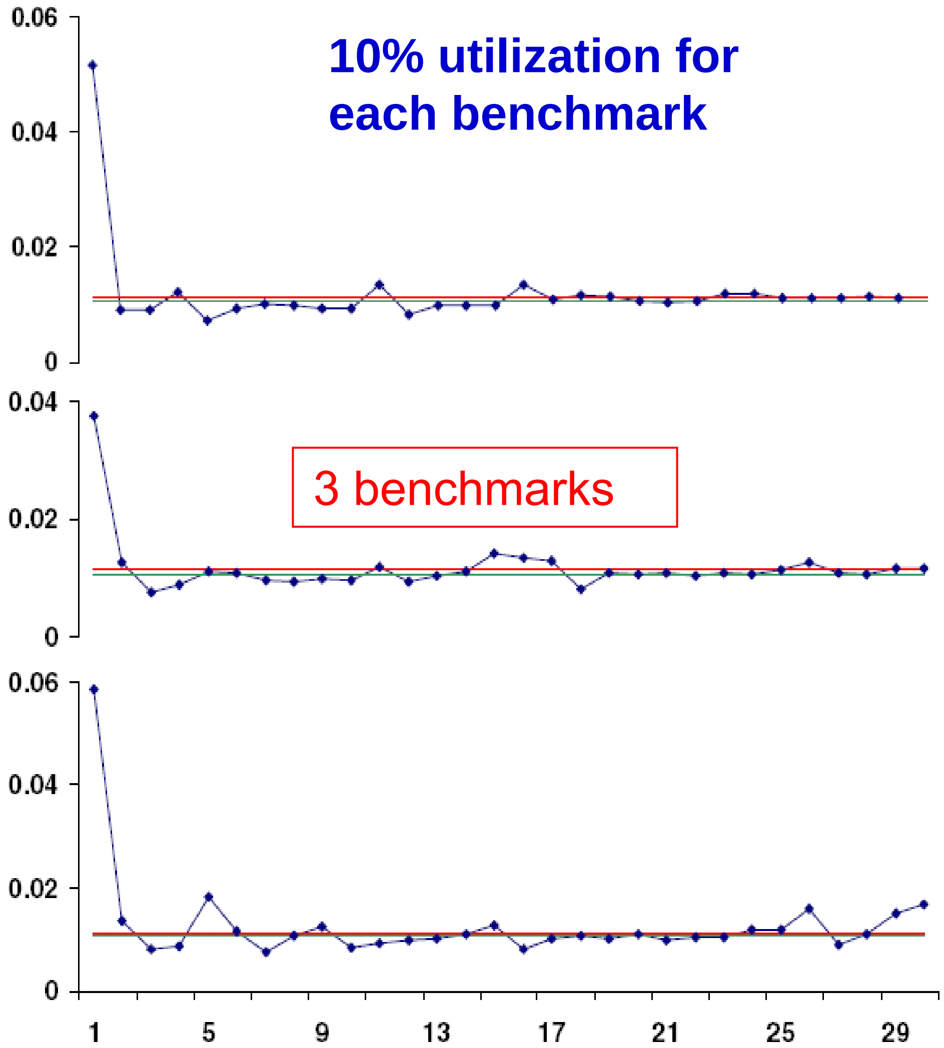


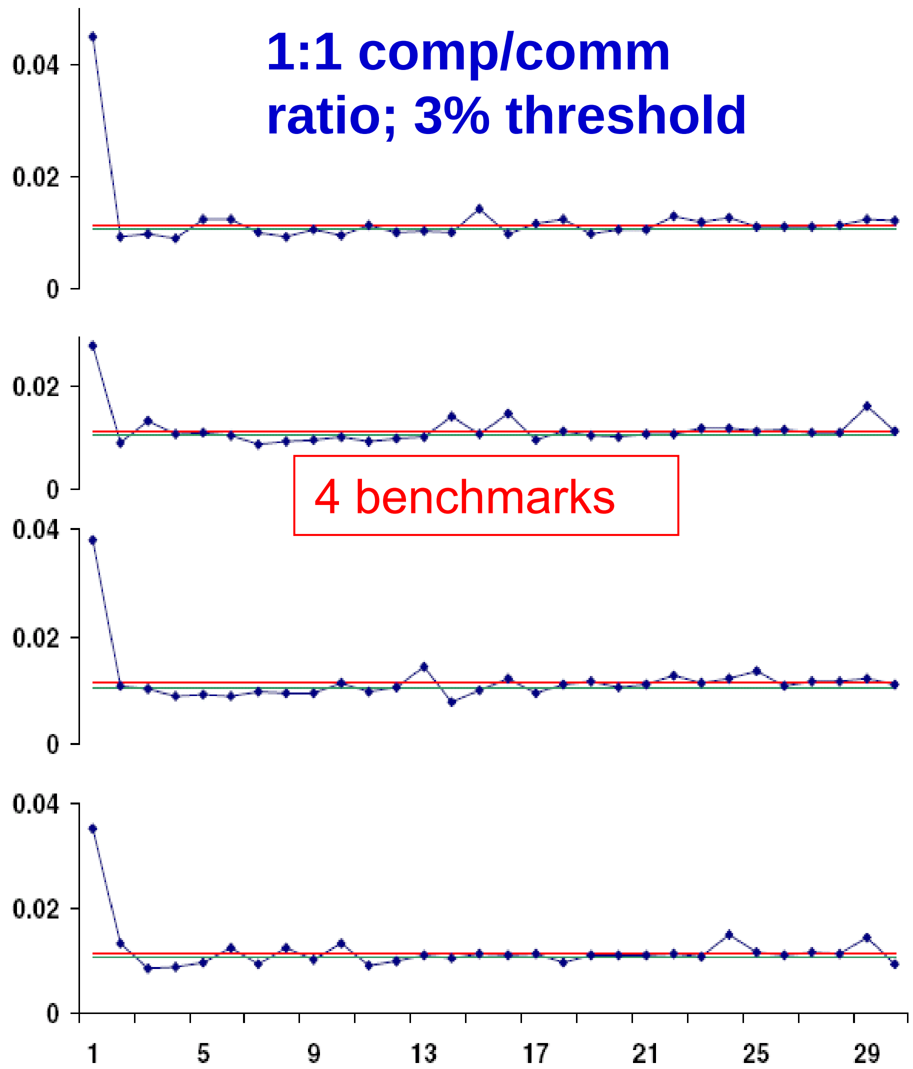
Time-sharing multiple BSP applications



What happens as we increase the number of benchmarks running simultaneously?

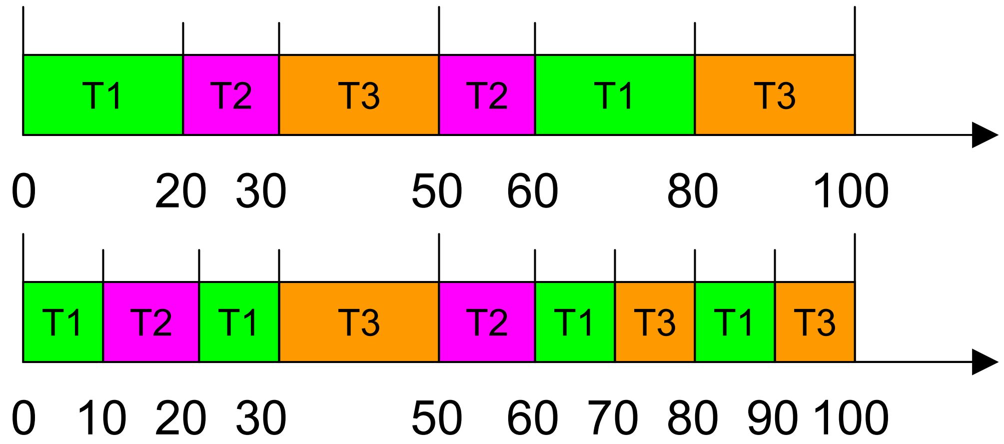
Time-sharing multiple BSP applications (cont.)





Time-sharing multiple BSP applications (cont.)

- Maintain reasonable control as we scale
- Certain degree of oscillation
  - Local scheduler schedule interrupt
  - Individual host, num of processes increases
    - Smaller chance of running uninterrupted throughout its slice
    - Smaller chance of starting its slice at same time.



Time(millisecond)

(period, slice)

Task1(50, 20) ms

Task2(50, 10) ms

Task3(100, 40) ms

Time-sharing multiple BSP applications (cont.)

- Maintain reasonable control as we scale
- Certain degree of oscillation
  - Local scheduler schedule interrupt
  - Individual host, num of processes increases
    - Smaller chance of running uninterrupted throughout its slice
    - Smaller chance of starting its slice at same time.

Less synchronized with processes on other nodes

Global controller invoked more often

System begins to oscillate

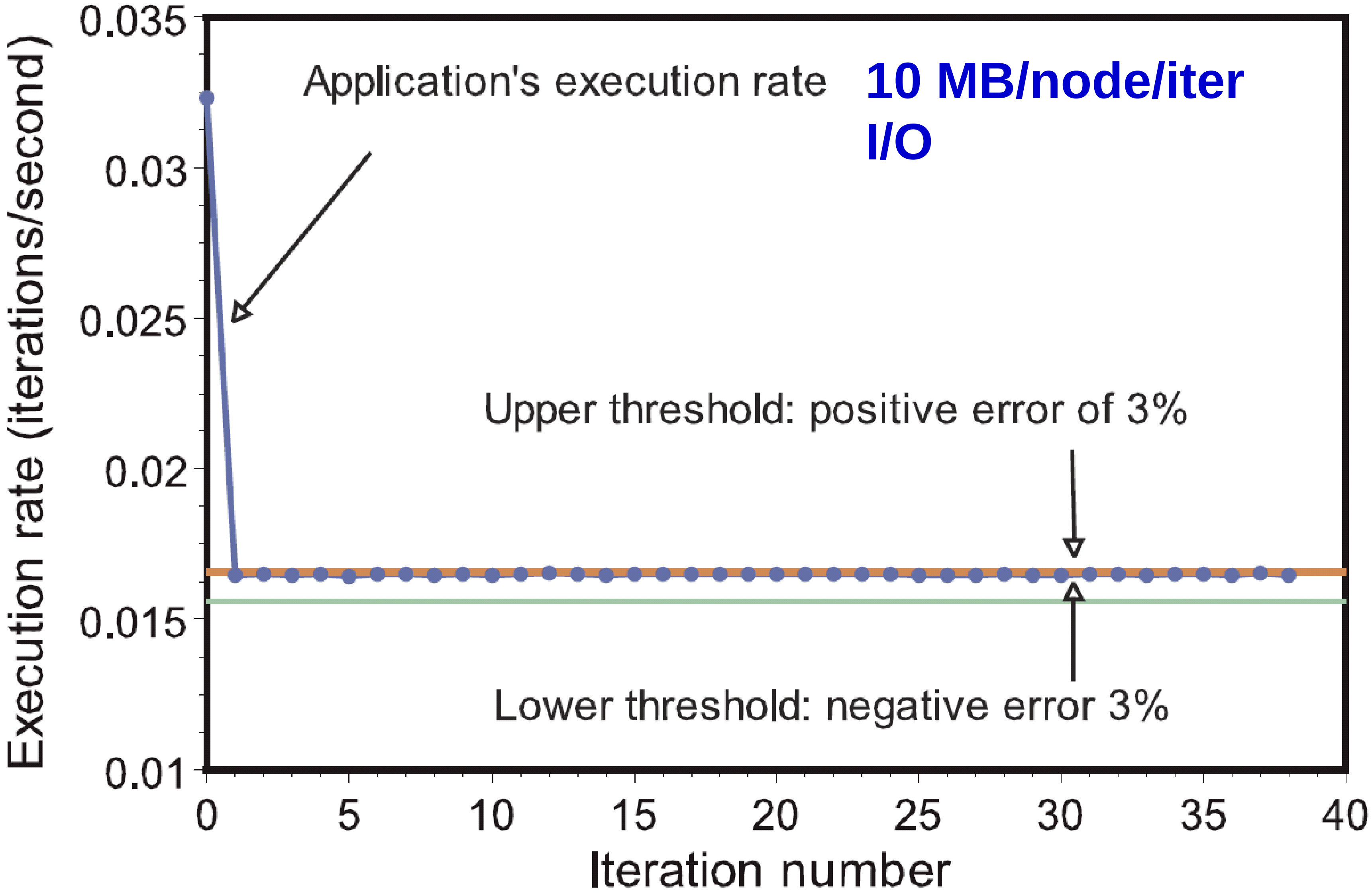
Feedback control loop freq less than freq of small performance changes

However, degradation is graceful, and long term averages are well behaved.

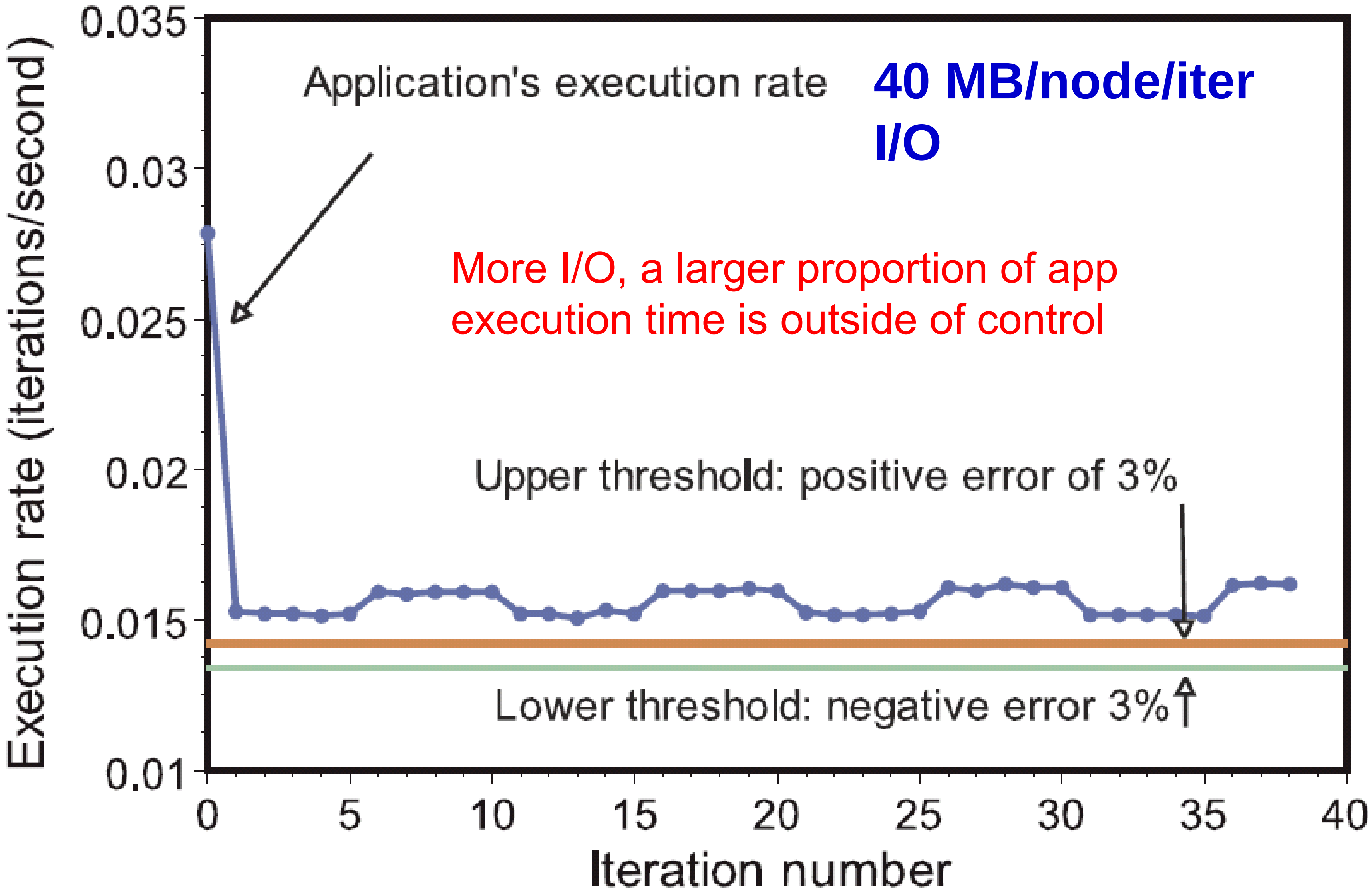
Effects of local disk I/O

- Modified benchmark to write to disk in every iteration; fsync()
- 1) high comp/comm ratio
  - 0, 1, 5, 10, 20, 40, 50 MB/node/iter disk I/O
- 2) 10MB/node/iter disk I/O
  - different comp/comm ratios

Effectively control execution rates



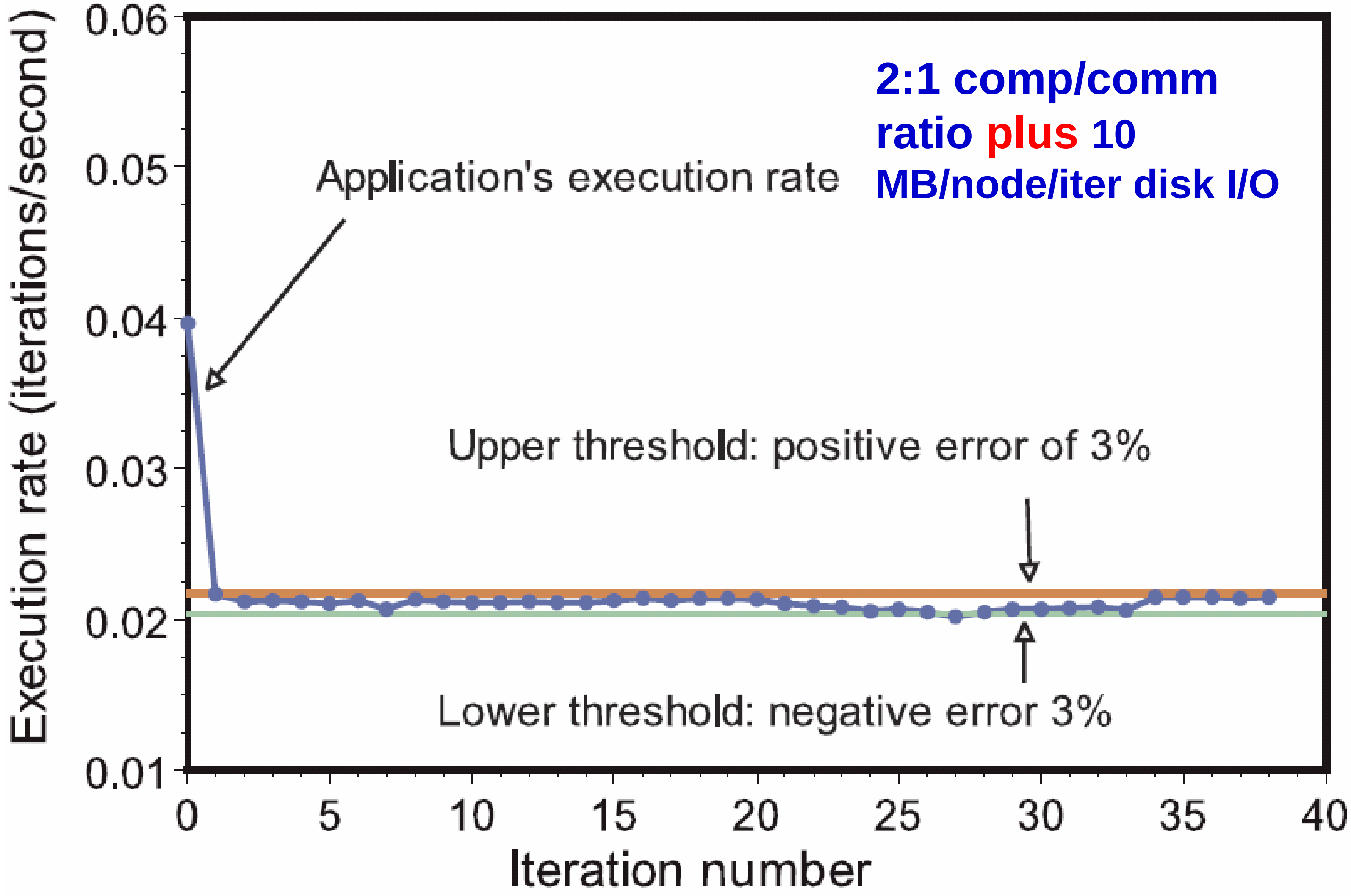
Positive bias; app runs faster than desired



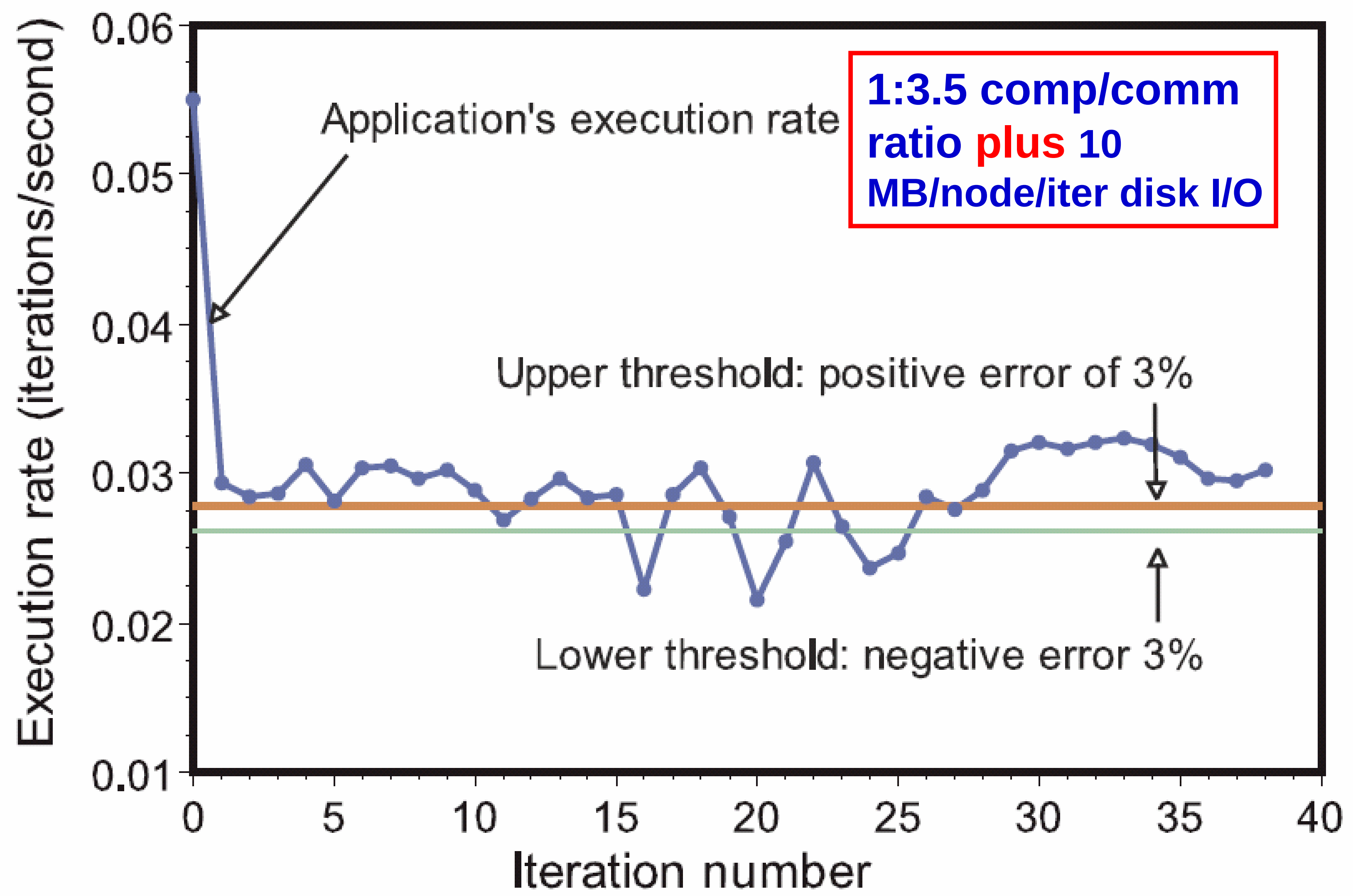
Effect of local disk I/O

- Modified benchmark to write to disk in every iteration; fsync()
- 1) high comp/comm ratio
  - 0, 1, 5, 10, 20, 40, 50 MB/node/iter disk I/O
- 2) 10MB/node/iter disk I/O
  - different comp/comm ratios

Effectively control execution rates despite significant amounts of network and disk I/O



Degrades gracefully when limits are exceeded



Effect of local disk I/O (cont)

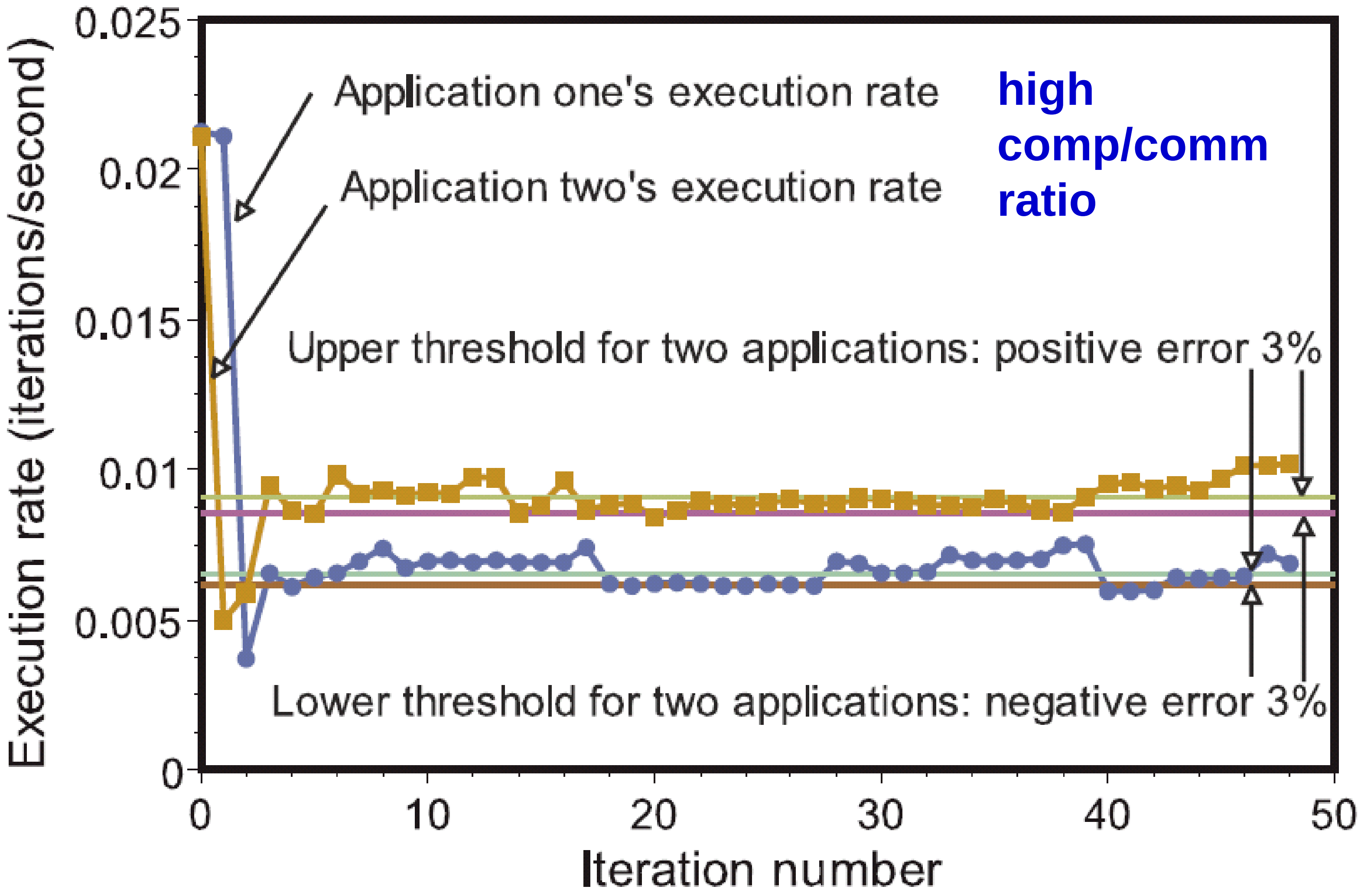
- Modified benchmark to write to disk in every iteration; fsync()
- 1) high comp/comm ratio
  - 0, 1, 5, 10, 20, 40, 50 MB/node/iter disk I/O
- 2) 10MB/node/iter disk I/O
  - different comp/comm ratios

- Effectively control exe rates of apps performing significant amounts of network & disk I/O
- Points at which control begins to decline depends on comp/comm ratio & amount of disk I/O

Effects of physical memory use

- Modified benchmark to control its mem working set size
- 1.5GB physical mem; cluster node
- Run 2 instances of benchmark on 4 nodes
- 1.3GB (700 + 600) combined working set

Despite significant memory use, our system maintains control



Conclusion

- Designed, implemented, and evaluated a new approach to time-sharing parallel applications with performance isolation
- Approach based on *periodic real-time scheduling* of nodes combined with *global feedback control* of real-time constraints
- Provides a simple way to control execution rate of applications while maintaining efficiency
- Despite only isolating and controlling CPU, memory, comm I/O, and local disk I/O follow.

Future work

- Apply our approach to wider range of workload, e.g.
  - Web applications (complex comm & sync behavior)
  - High-performance parallel scientific applications (requirement not know a priori)
- Exploit direct feedback from end-user to solve optimization problem

Thank you!

- **Bin Lin**'s homepage: http://www.cs.northwestern.edu/~blin
- Thesis: User-directed Adaptation

- Group project webpage:
  http://virtuoso.cs.northwestern.edu
- Presciencelab webpage:
  http://presciencelab.org

Related work

- Gang scheduling
  - Fine-grain scheduling
  - Schedule all app’s threads at identical times on different nodes
  - Complex code
  - High cost of communication for synchronization
- Implicit co-scheduling
  - Reduce communication by inferring remote scheduler
  - Complexity in inference & adapting local schedule
  - Difficult to control execution rate, response time and resource usage
- Feedback based control in many other domains