A Generalized Blind Scheduling Policy

Hanhua Feng[1], **Vishal Misra**[2,3] and Dan Rubenstein[2]

[1]Infinio Systems

[2]Columbia University in the City of New York

[3]Google

TTIC SUMMER WORKSHOP: DATA CENTER SCHEDULING FROM THEORY TO PRACTICE

Overview

Introduction

Scheduling Policies in Queueing Models

Scheduling is a compromise ...

- not only between individual tasks, but also ...
- between systems with different workload patterns,
- between different performance requirements, including
  - mean response time, mean slowdown, responsiveness, ...
  - fairness measures: seniority, RAQFM, ...

Our work

- Design a flexible scheduling policy to balance these requirements.

Assumptions in this talk

- Single-server queueing model
- Work-conserving, preemption allowed

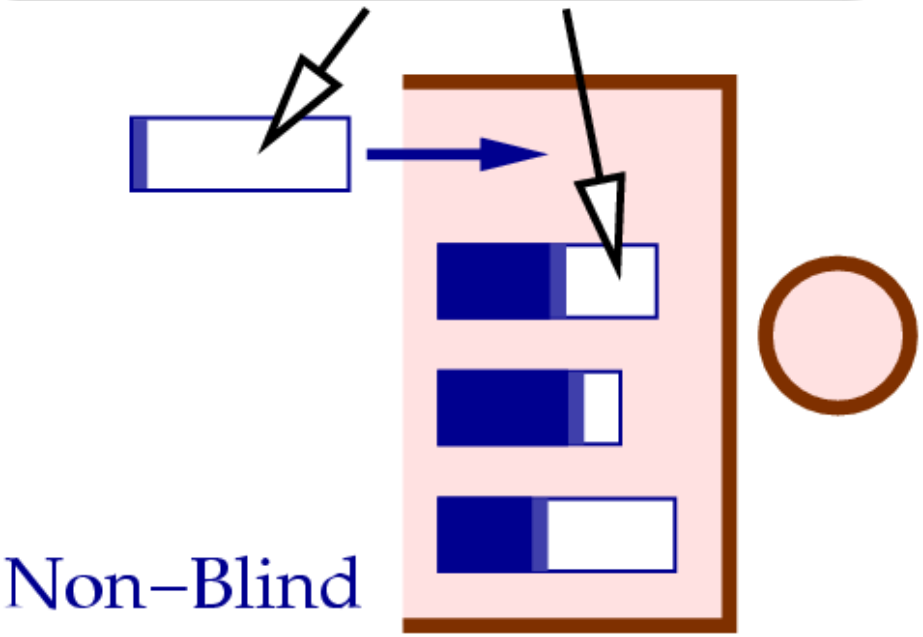
Blind Scheduling Policies

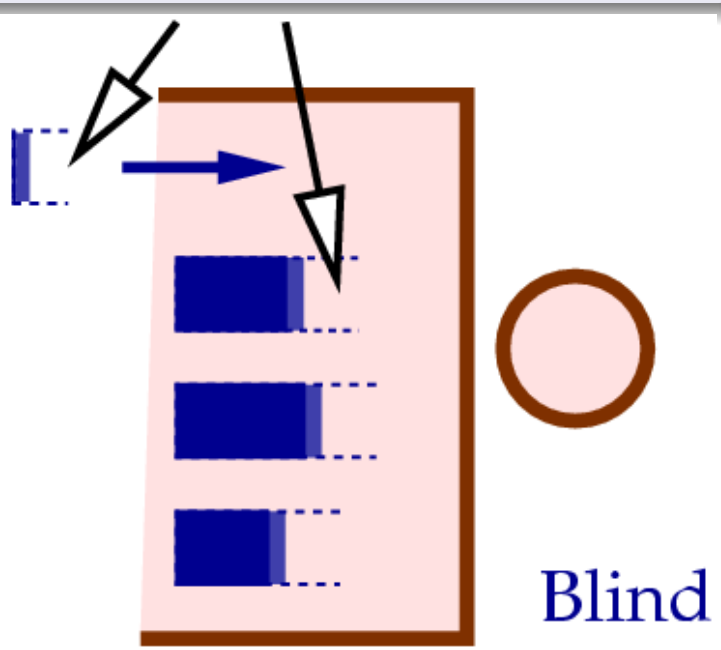
Non-blind policies

Know required and remaining service time when tasks arrive.

Blind policies

No information about remaining service until tasks complete.





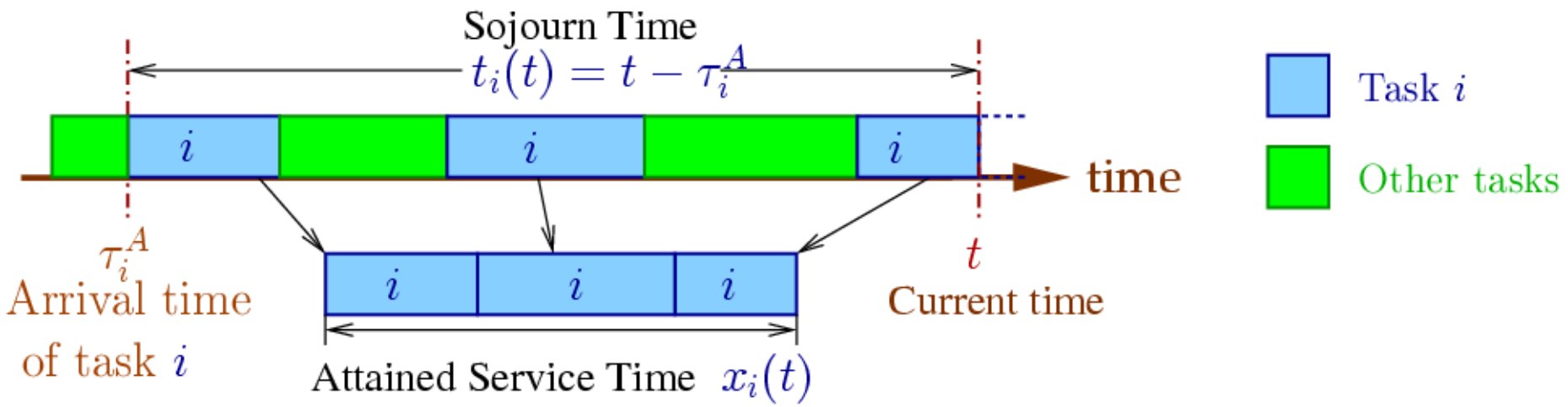
Non-blind policy examples

SJF, SRPT, SMART ...

Blind policy examples

FCFS, PS, LAS, LCFS, ...

How Do We Measure Fairness of a Policy?



Fairness criteria [cf. Raz,Levy&Avi-Itzhak 2004]

- Task seniority (emphasis on t_i) $\Rightarrow$ FCFS
- Task service requirements (emphasis on x_i)
  - Equal attained service $\Rightarrow$ LAS/FBPS
- Combination of the two: Equal share of processor
  - Current: $dx_i(t)/dt_i(t) \equiv x_i'(t)$ $\Rightarrow$ PS
  - Aggregated: $x_i(t)/t_i(t)$ $\Rightarrow$ GAS

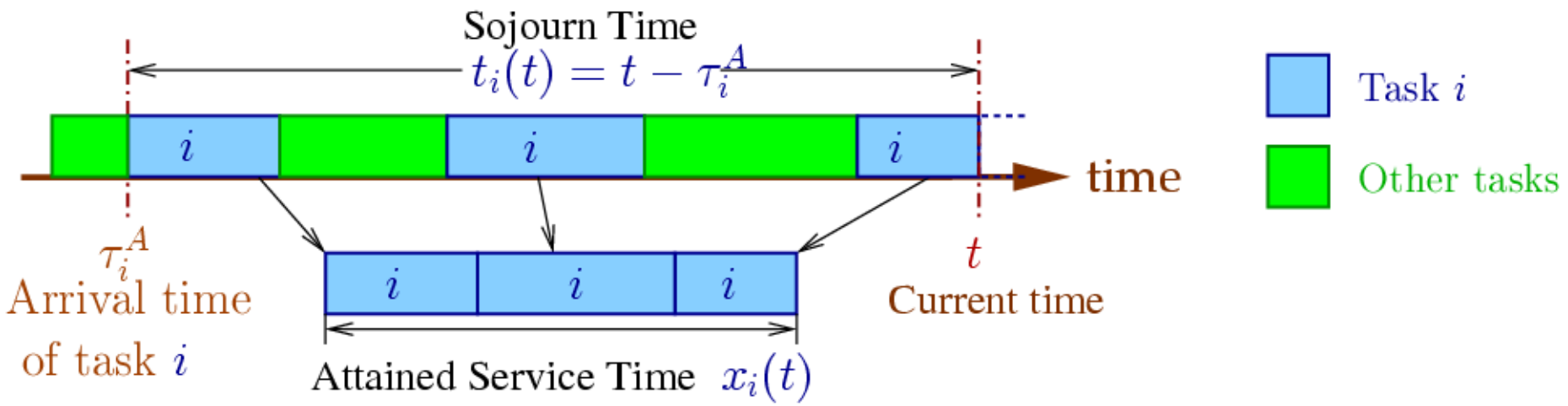
How to Measure Fairness of a Policy? (cont'd)

Fairness measures in literature

- Comparison vs FCFS [Wang & Morris 1985]
- RAQFM: Comparison vs PS [Raz,Levy&Avi-Itzhak 2004]
  - A quantitative measure.
  - Difficult to analyze: with results for FCFS, LCFS, PLCFS, and Random in $M/M/1$.
  - $G/D/m$ [Raz,Levy&Avi-Itzhak 2005]
- Expected slowdown for given required service $E[S|X = x]$ compared with PS [Wierman&Harchol-Balter 2004]
  - A classification: always fair/unfair, sometimes fair.
  - Assume $M/G/1$.
  - Extended in [Wierman&Harchol-Balter 2005].
- SQF [Avi-Itzhak,Brosh&Levy 2007]

PBS Policy

Balance Between Two Fairness Criteria



Two fairness criteria (cont'd)

- Seniority — Prefer larger sojourn time $t_i(t)$
- Service requirements — Prefer smaller attained service $x_i(t)$

Our idea: A configurable balance

- Schedule a task with maximal $t_i(t) - \alpha x_i(t)$.
- More general: $g(t_i(t)) - \alpha g(x_i(t))$, e,g., $\log t_i(t) - \alpha \log x_i(t)$.

Our Parameterized Scheduler: PBS

The PBS policy with a single server

- For every task i, compute its **priority value**

$$p_i(t) = \log t_i(t) - \alpha \log x_i(t), \quad \text{Equivalent to} \quad P_i(t) = \frac{t_i(x)}{[x_i(t)]^\alpha}$$

- α is a configurable parameter in $[0, \infty)$.
- At time t, serve the task with the highest priority p_i (or P_i).
  - Randomly choose among equal-priority tasks.
  - Preempt low-priority tasks, if currently been served.
- Can be used in continuous time (theory) or in discrete time (practice).

PBS: A Unified Priority-Based Scheduler Sigmetrics 2007

PBS: Priority-based Blind Scheduling (cont'd)

Why PBS?

- Tunable: Parameter α can be changed from 0 to ∞.
  - Emulate well-known policies: $P_i = t_i / x_i^{\alpha}$
  - $\alpha = 0$: First-come first-serve (FCFS) $P_i = t_i$
  - $\alpha \to \infty$: Least attained service (LAS), $P_i \sim 1/x_i$
    a.k.a. Foreground-Background Processor-Sharing (FBPS)
  - $\alpha = 1$: Greatest Attained Slowdown (GAS), $P_i = t_i / x_i$
    closely emulate Processor-Sharing (PS).
  - $\alpha =$ other values: Hybrid policies.
- Blind: Using only past information (t_i, x_i)
- Simple: Easy to implement.
- Dimensionless: Not dependent on scale of time unit (minute, second).

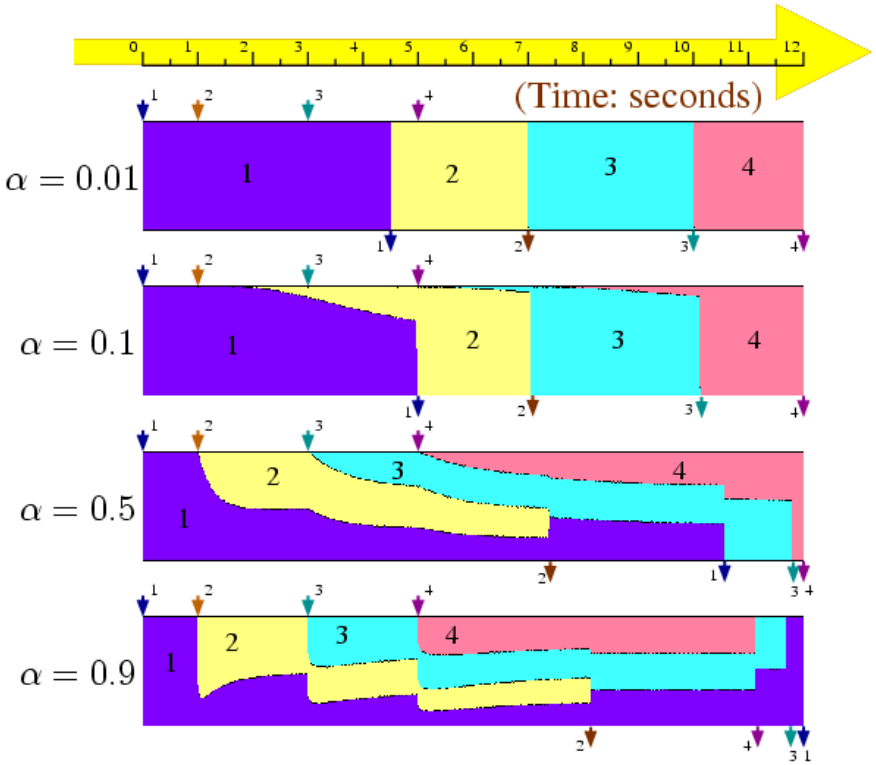
Behavior of PBS

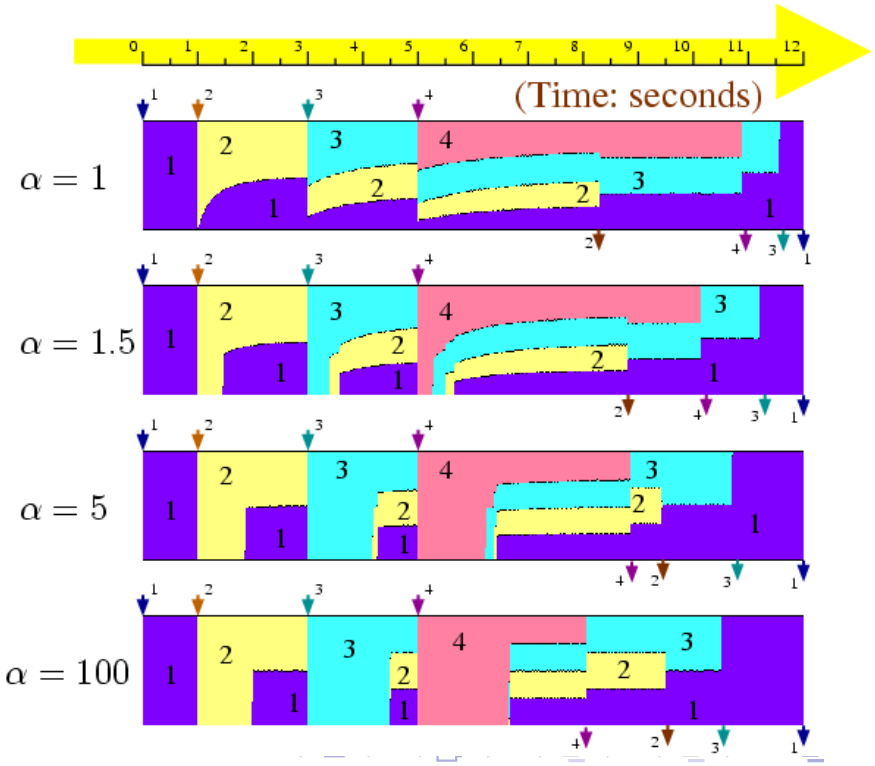
An example

- Four tasks in 4 colors
- Arrival time: 0s,1s,3s,5s
- Service: 4.5s,2.5s,3s,2s

How to read the graphs

- X-axis: Time
- Y-axis: CPU utilization per task.
- Area: Service received.





Properties

Properties of PBS for $0 < \alpha < \infty$

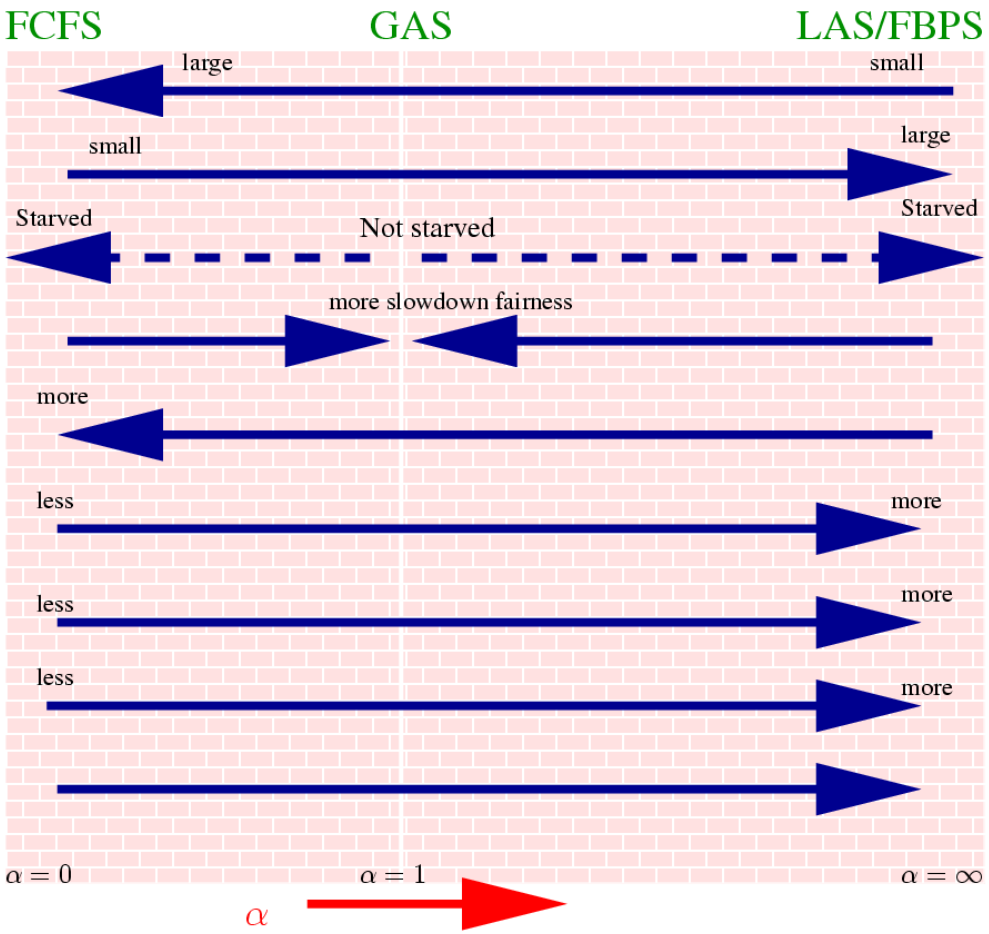
Some properties of PBS proved in the paper

- A new task immediately receives service after arrival.
  - Small CPU fraction for $\alpha < 1$
  - Large CPU fraction for $\alpha > 1$.
- **Seniority**: Earlier tasks get more attained service.
- **Time-shared**: CPU may be shared by two or more tasks.
  - **Hospitality**: A new task always gets a CPU share.
- **Convergence**: Converge to PS in a long run for long jobs.
  - Converge to DPS with an offset to log formula,
- **No Starvation**: Priority values of temporarily blocked tasks increase towards infinity, and will become highest-priority task.
  - For α close to 0 (FCFS) or ∞ (LAS), tasks may be blocked for a long time.

PBS Tunability: A Graphical Conclusion

PBS is monotonic in many aspects

- Guidelines for tuning α manually.



Monotonicity of PBS with respect to α in terms of ...

Mean response time for DHR

Mean response time for IHR

Starvation

Slowdown Fairness

Seniority Fairness

Attained Service Fairness (Variability)

Service Interruption

Responsiveness

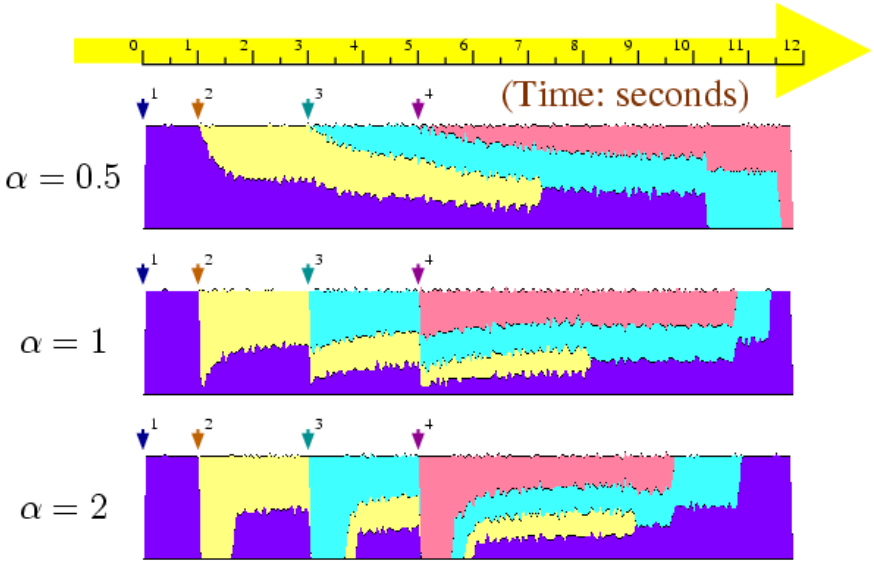
Preference to Small Tasks

Implementation

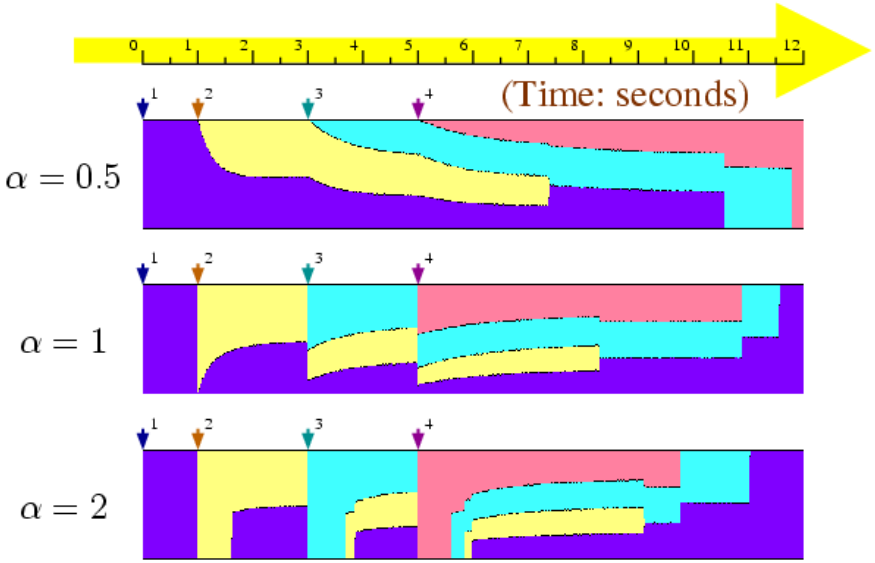
Implementation in Linux Kernel

CPU utilization measurement

- Discrete time implementation in Linux 2.6.15.
- 50ms moving average of measured CPU utilization per task.
- Measurement results are close to simulation results.
- Difference is the roughness on small time scales.



(Measured in Modified Linux System)

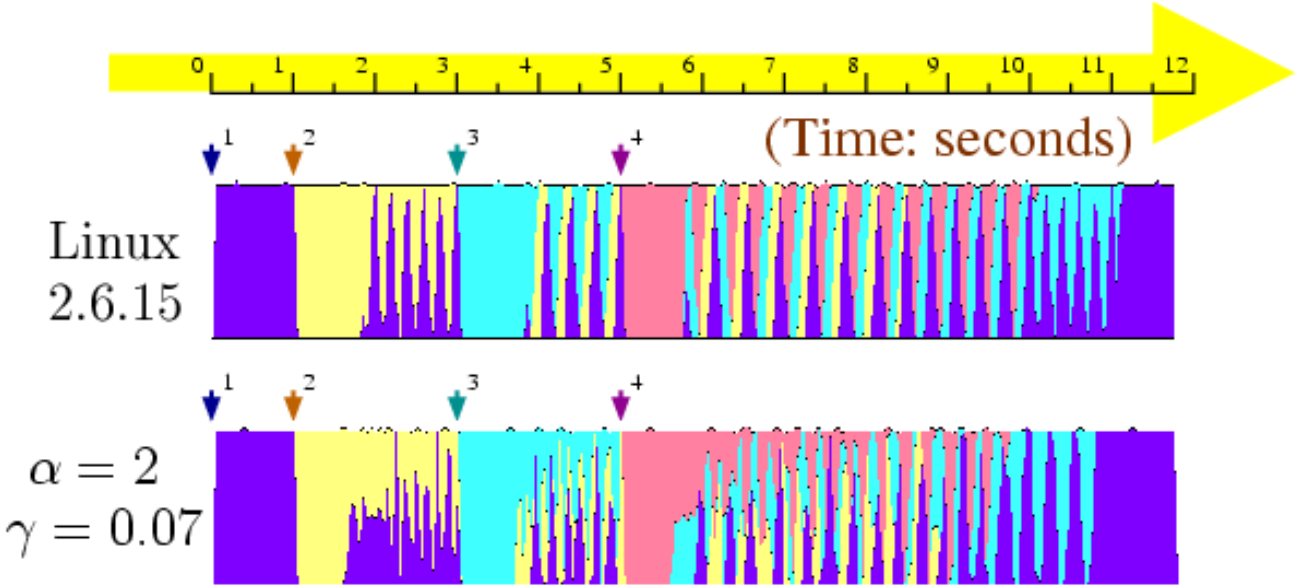


(Simulation Results)

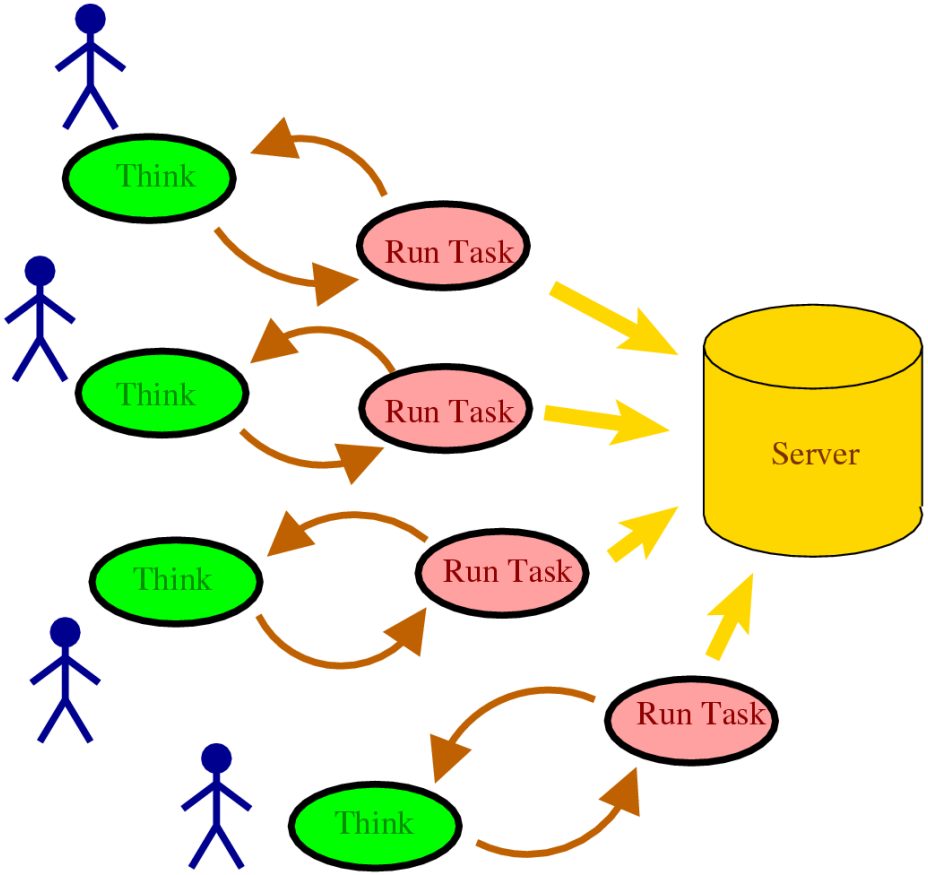
Emulating Existing Linux Scheduler

A small tweak

- Add a bonus priority γ to the current task in order to limit context switch.
- With $\alpha = 2$ and $\gamma = 0.07$, PBS looks close to Linux native scheduler.



Experimental model

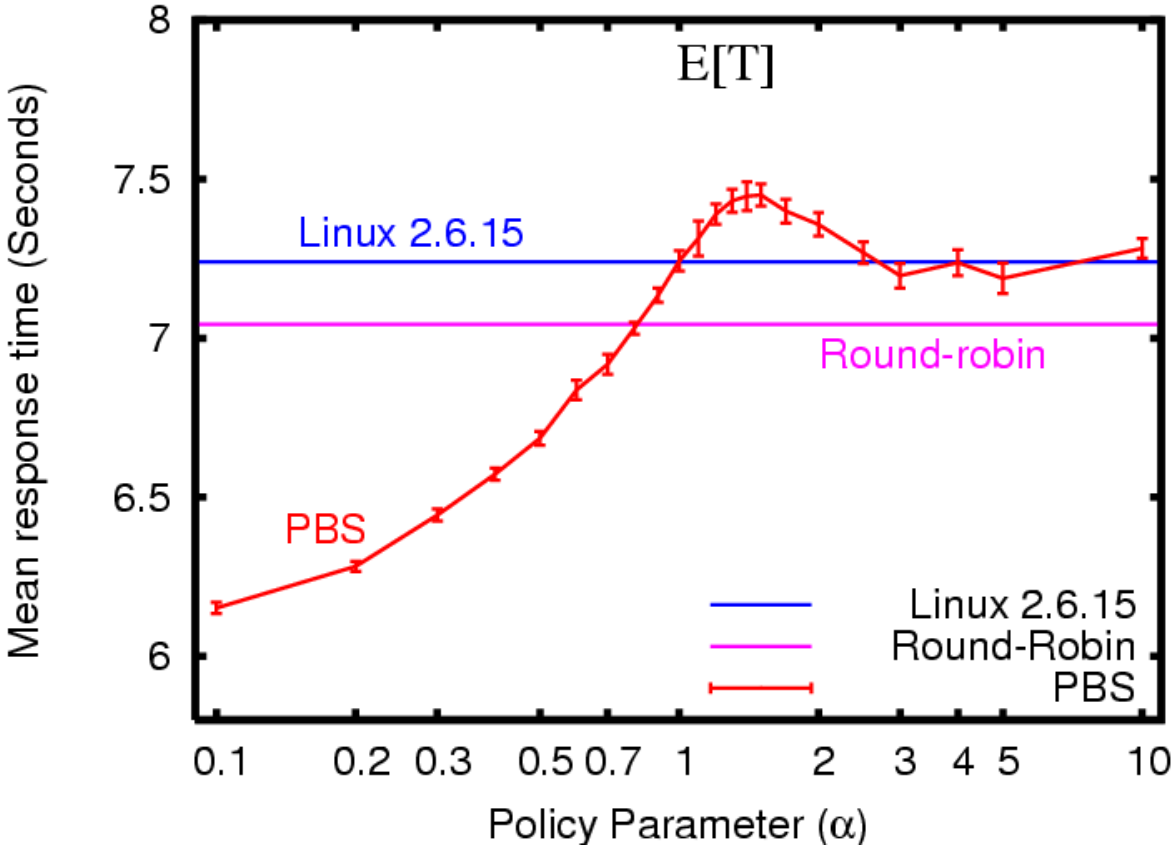


A closed model

- A fixed number of users.
- Each user submits a task after thinking.
- Exponentially distributed thinking time.
- Response time of every task is measured.

Experimental Results (Set A)

- Computational tasks with almost deterministic CPU usage.
- About 3-second processing for each task.
- 8 users, 25s average thinking time.

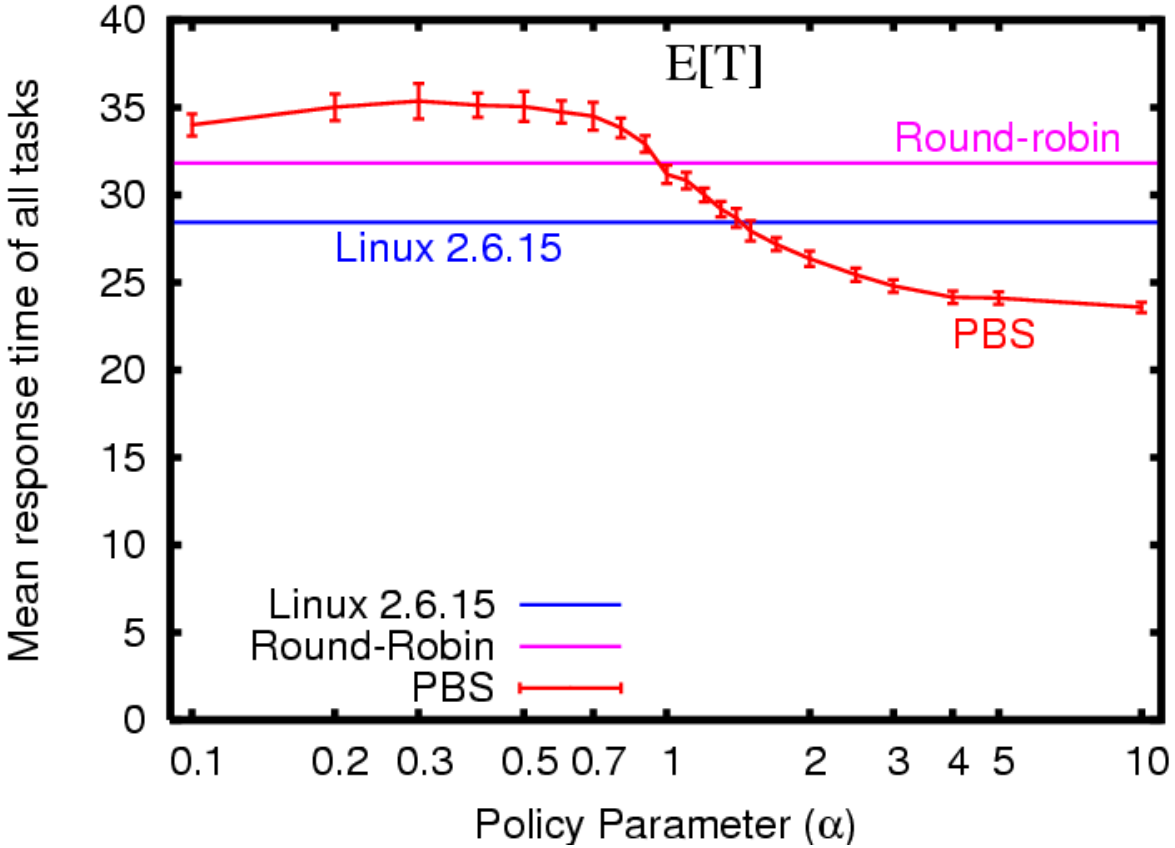


For this work load,

- small α works best.
- PBS ($\alpha < 0.7$) outperforms Linux and Round-robin.

Experimental Results (Set B) (1/2)

- Apache web server 2.0, dynamic pages with heavy processing.
- Overloaded with 30 users, 10s average thinking time.
- Processing time is heavy-tailed.



For this workload,

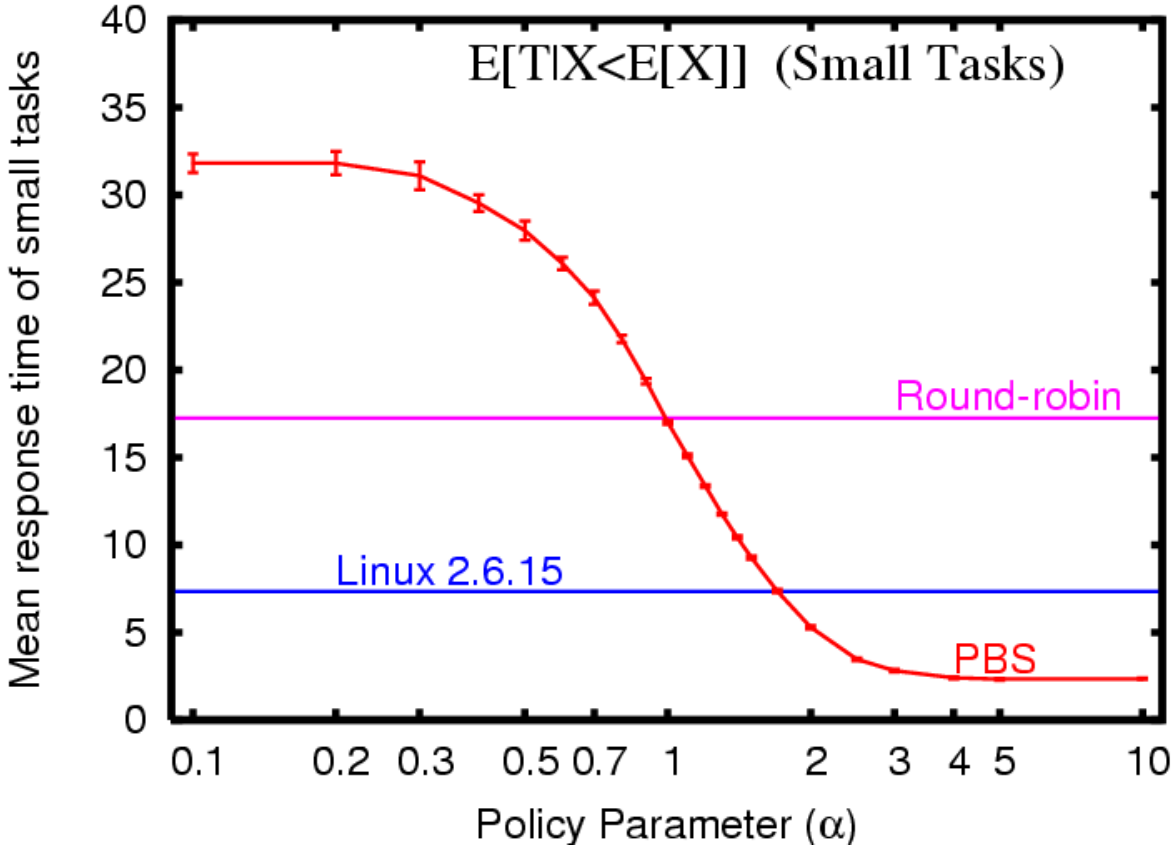
- big α works best.
- PBS ($\alpha > 2$) outperforms Linux and Round-robin.

Conclusion

Different α's are better for different workloads.

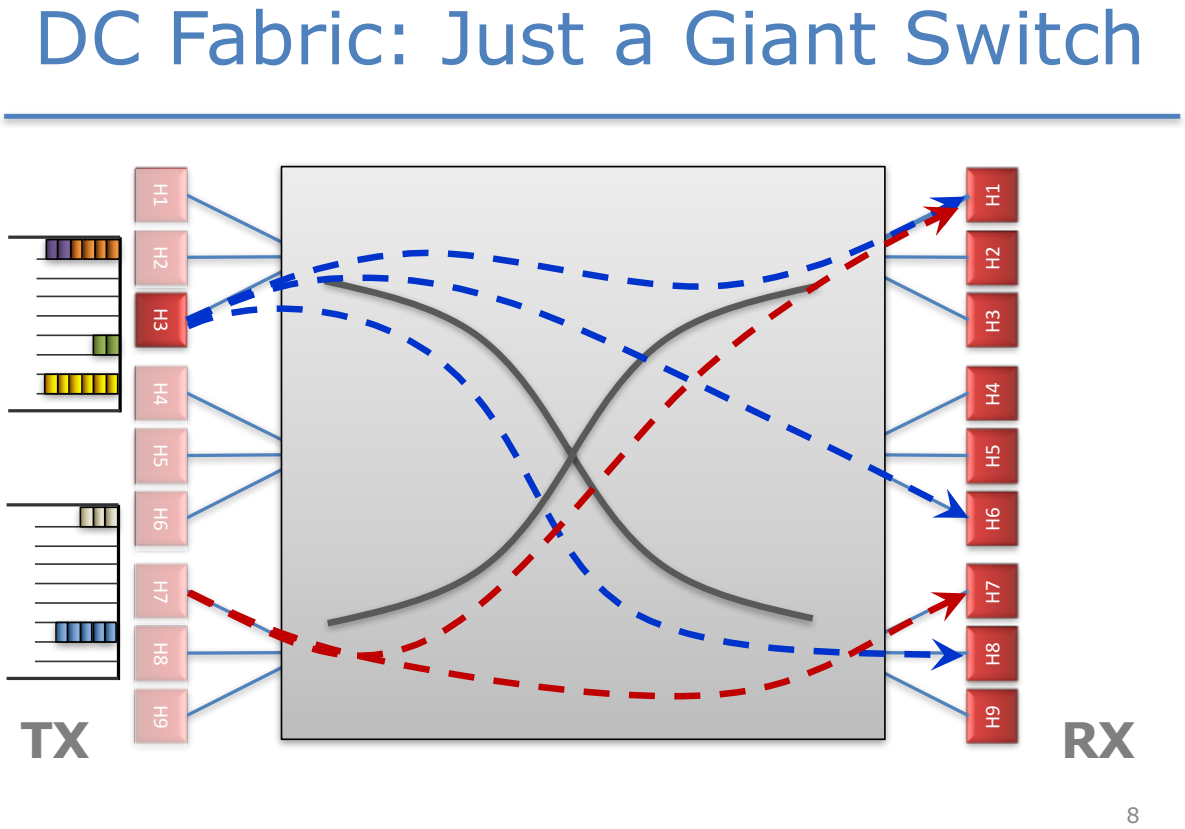
Experimental Results (Set B) (2/2)

- Apache web server 2.0, dynamic pages with heavy processing.
- Overloaded with 30 users, 10s average thinking time.
- Processing time is heavy-tailed.



Data center

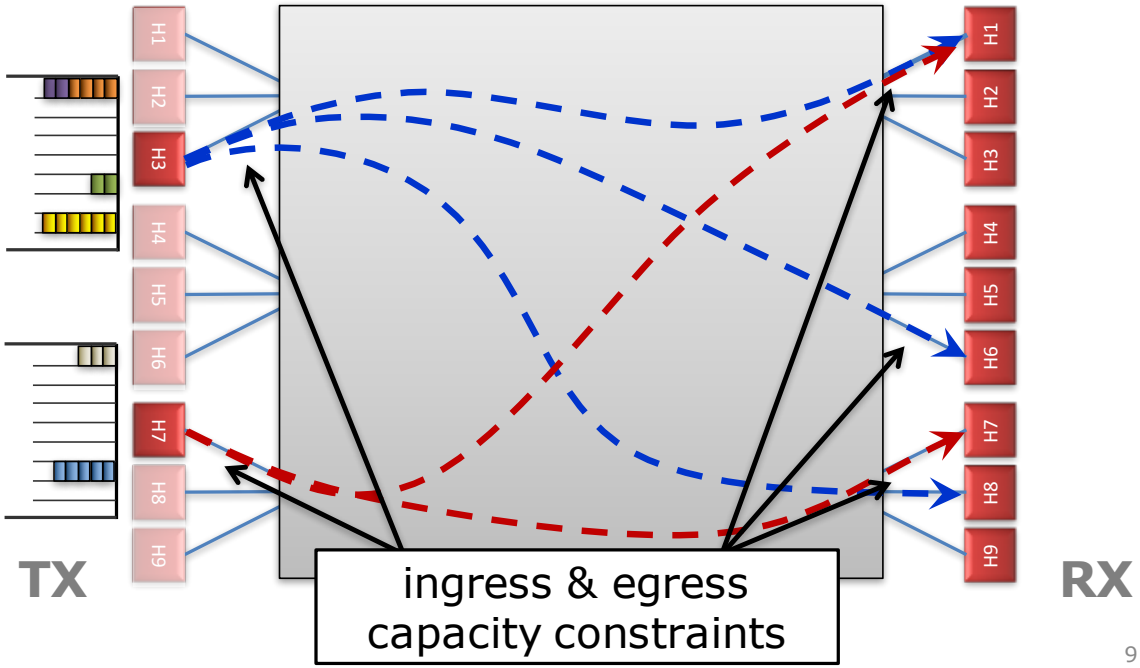
Data center fabric: A giant switch



Transport in Data Centers

DC transport = Flow scheduling on giant switch

Objective?
- Minimize avg FCT



9

pFabric

pFabric: Minimal Near-Optimal Datacenter Transport (Alizadeh et al. Sigcomm 2013)

Goal: Complete Flows Quickly

- Requires scheduling flows such that:
  - High throughput for large flows
  - Fabric latency (no queuing delays) for small flows

Prior work: use rate control to schedule flows

- vastly improve performance, but complex

pFabric in one slide

pFabric Packets

Packets carry a single priority number, e.g., prio = remaining flow size

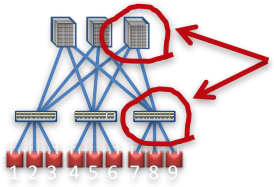
pFabric Switches

Very small buffers (20-30KB for 10Gbps fabric)
Send highest priority / drop lowest priority pkts

pFabric Hosts

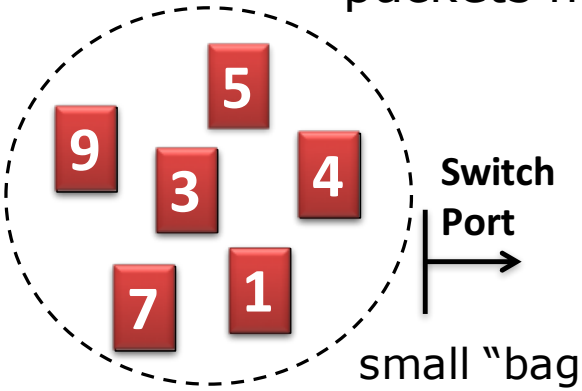
Send/retransmit aggressively
Minimal rate control: just prevent congestion collapse

pFabric Switch



pFabric Switch

- **Priority Scheduling**
  send highest priority packet first
- **Priority Dropping**
  drop lowest priority packets first

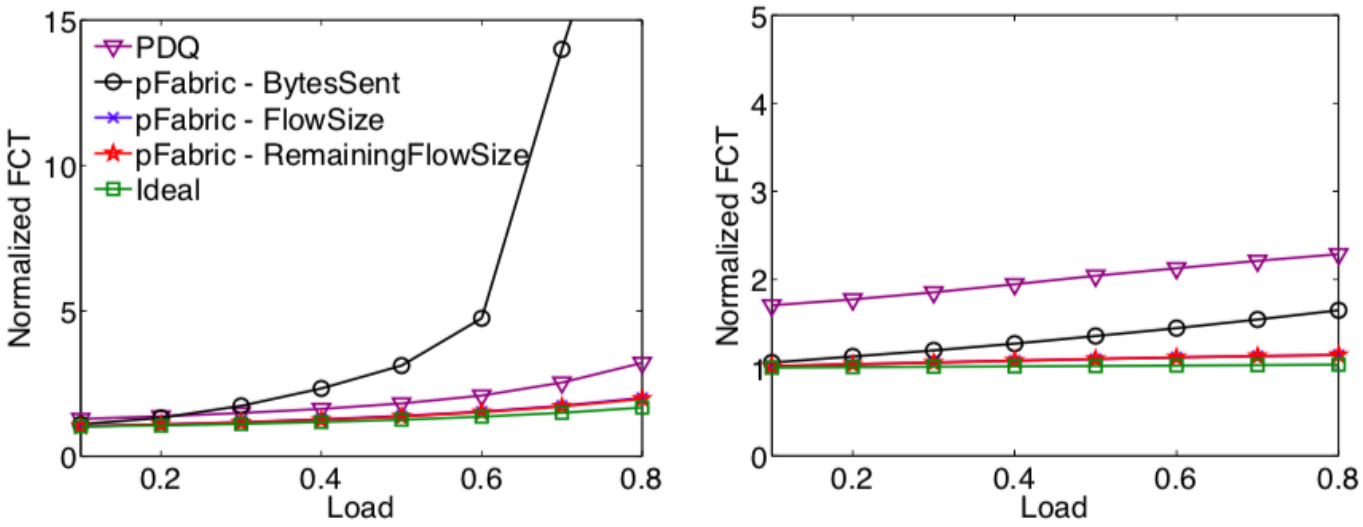


small "bag" of packets per-port

prio = remaining flow size

13

pFabric results



(a) Web search workload (b) Data mining workload

Figure 14: Overall average FCT for different priority assignment schemes. Note the different y-axis range in these plots.

Summary of results

- SJF and SRPT achieve nearly equal performance (for large flows SRPT $\approx 15\%$ better than SJF).
- LAS (BytesSent) for DataMining nearly as good as optimal/SRPT, for WebSearch performance breaks down at high load.
- Many jobs of similar sizes keep getting pre-empted at high loads, until the new job "catches up", and then it starts again

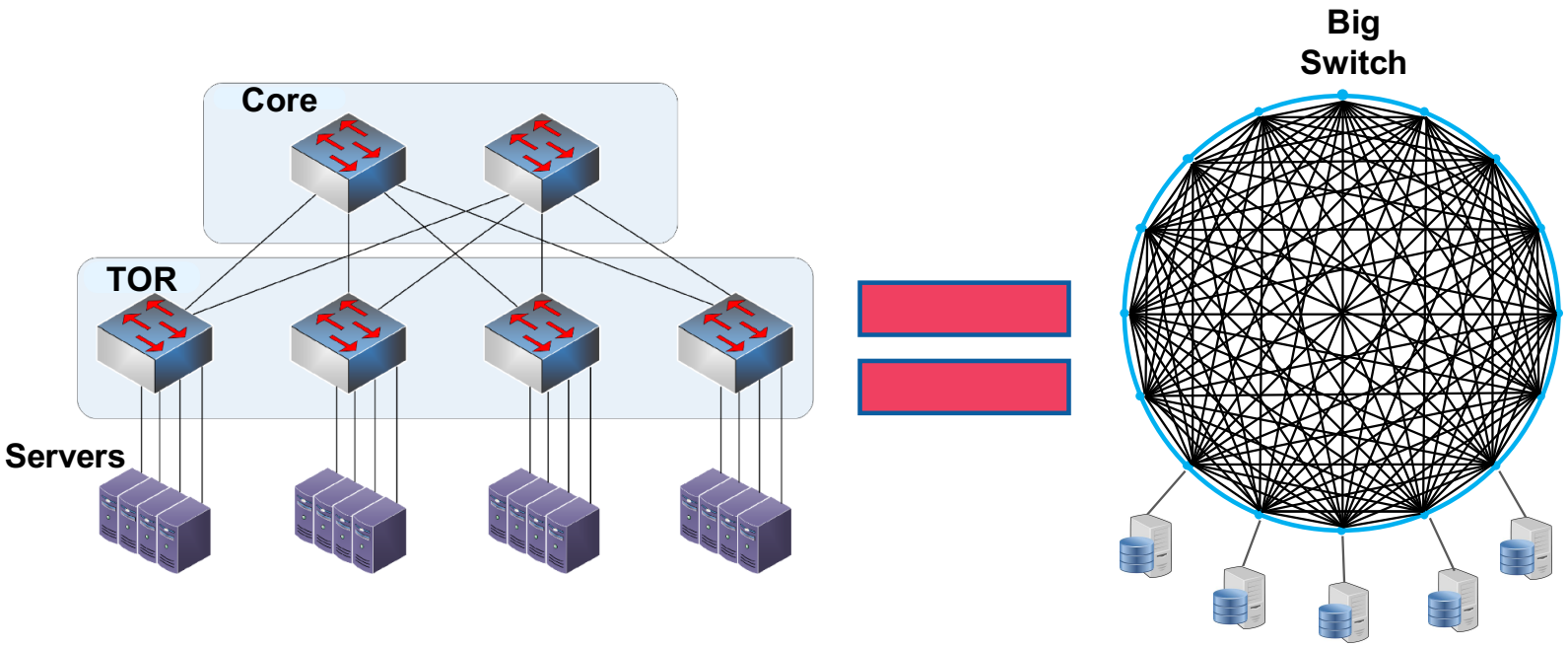
Tradeoffs

- SJF/SRPT work across workload distributions, but *require* Job Size information
  - Job size often not availabe ahead of time
- LAS requires *no* job size information, but doesn't work well with non-heavytailed job size distributions.
- PBS can achieve balance with right α

Homa: Practical Low Latency Datacenter Transport (Sigcomm 2018)

Congestion At The Edge

- **No persistent congestion in the core**



Approach and Performance

Schedule messages in shortest-remaining-first order (SRPT)
Near-optimal average latency & good tail latency for short messages

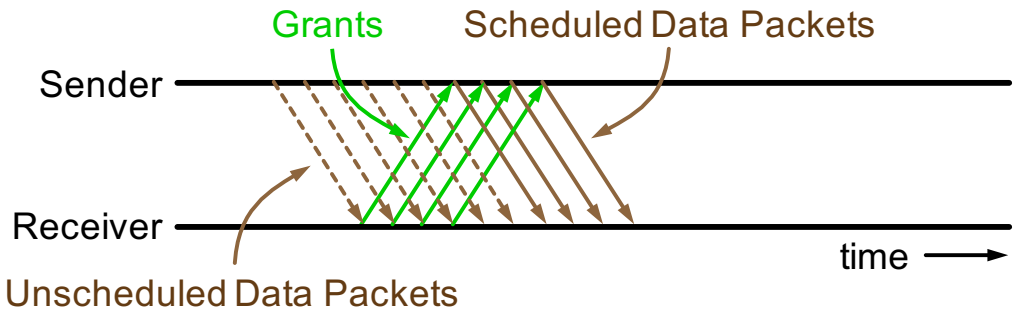
Key Ideas

- Receiver-driven congestion control and packet scheduling
- Reduce buffer occupancy & improve latency
- Use of network priorities dynamically assigned by receivers
- Bypass queues for short messages
- Controlled overcommitment on receivers downlink
- Avoid bandwidth waste, leads to high bandwidth utilization

Homa mechanism

SRPT to schedule packets

- **Homa receivers schedule incoming packets: one grant per packet**
- **Problem: 1 RTT additional latency for scheduling (size unknown)**
- **Solution: Transmit 1 RTT of packets per message blindly**



PBS for Homa

- No need to know size of flows
- First packet has natural high priority (unless $\alpha = 0$)

Conclusion

Conclusion and Future Work

Contributions

- We introduce a novel configurable policy, PBS.
- By varying the single parameter, we can tune for various performance and fairness requirements.
- Demonstrate properties and advantages of PBS by analysis, simulations, implementation, and experiments.

Current/Future work

- Closed form of mean response time in $M/G/1$ for any α.
- Design an automatic mechanism to dynamically adapt α to workload.
- Implement PBS with Data Center scheduling/transport systems
- Extend PBS to multi-core systems.

The End

The End