Managing the performance of large, distributed storage systems

Scott A. Brandt
and
Carlos Maltzahn, Anna Povzner, Roberto Pineiro,
Andrew Shewmaker, and Tim Kaldewey
Computer Science Department
University of California Santa Cruz
and
Richard Golding and Ted Wong, IBM Almaden Research Center

LANL ISSDM Talk — July 10, 2008

Outline

- Problem: managing the performance of large, distributed storage systems
- Approach: end-to-end performance management
- General model: RAD
- Applying RAD to other resources: disk, network, and buffer cache
- Moving forward: data center performance management

Distributed systems need performance guarantees

- *Many* distributed systems and applications need (or want) I/O performance guarantees
  - Multimedia, high-performance simulation, transaction processing, virtual machines, service level agreements, real-time data capture, sensor networks, ...
  - Systems tasks like backup and recovery
  - Even so-called best-effort applications
- Providing such guarantees is difficult because it involves:
  - Multiple interacting resources
  - Dynamic workloads
  - Interference among workloads
  - Non-commensurable metrics: CPU utilization, network throughput, cache space, disk bandwidth

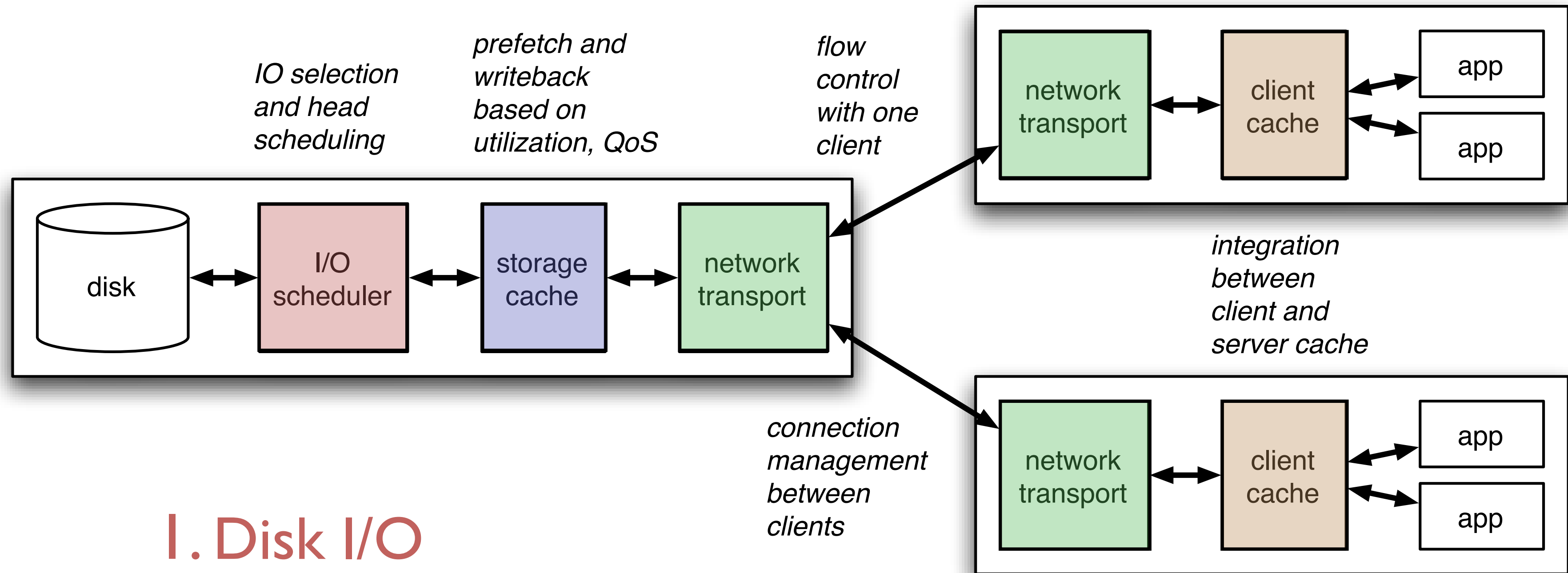
End-to-end I/O performance guarantees

- Goal: Improve end-to-end performance management in large distributed systems
  - Manage performance
  - Isolate traffic
  - Provide high performance
- Targets: High-performance storage (LLNL), data centers (LANL), satellite communications (IBM), virtual machines (VMware), sensor networks, ...
- Approach:
  1. Develop a uniform model for managing performance
  2. Apply it to each resource
  3. Integrate the solutions

Our current target

- High-performance I/O
  - From *client*, across *network*, through *server*, to *disk*
  - Up to hundreds of thousands of processing nodes
  - Up to tens of thousands of I/O nodes
  - Big, fat, network interconnect
  - Up to thousands of storage nodes with cache and disk
- Challenges
  - Interference between I/O streams, variability of workloads, variety of resources, variety of applications, legacy code, system management tasks, scale

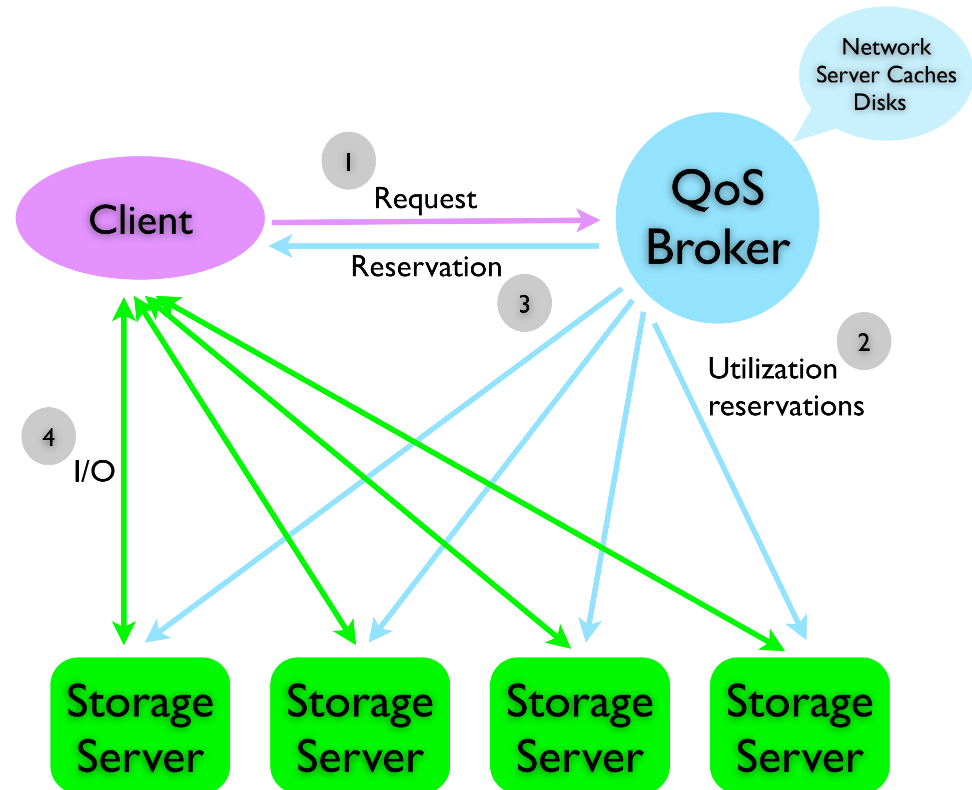
Stages in the I/O path



1. Disk I/O
2. Server cache
3. Flow control across network
   - Within one client's session *and* between clients
4. Client cache

System architecture

- Client: Task, host, distributed application, VM, file, ...
- Reservations made via broker
  - Specify workload: throughput, read/write ratio, burstiness, etc.
- Broker does admission control
  - Requirements + workload are translated to utilization
  - Utilizations are summed to see if they are feasible
  - Once admitted, I/O streams are guaranteed (subject to workload adherence)
- Disk, caches, network controllers maintain guarantees



Achieving robust guaranteeable resources

- Goal: Unified resource management algorithms capable of providing
  - Good performance
  - Arbitrarily hard or soft performance guarantees with
    - Arbitrary resource allocations
    - Arbitrary timing / granularity
  - Complete isolation between workloads
  - All resources: CPU, disk, network, server cache, client cache

➡Virtual resources indistinguishable from "real" resources with fractional performance

Isolation is key

- CPU
  - 20% of a 3 Ghz CPU should be indistinguishable from a 600 Mhz CPU
  - Running: compiler, editor, audio, video
- Disk
  - 20% of a disk with 100 MB/second bandwidth should be indistinguishable from a disk with 20 MB/second bandwidth
  - Serving: 1 stream, *n* streams, sequential, random

Epistemology of virtualization

- Virtual Machines and LUNs provide good HW virtualization
- Question: Given perfect HW virtualization, *how can a process tell the difference between a virtual resource and a real resource?*
- Answer: By not getting its **share** of the resource **when** it needs it

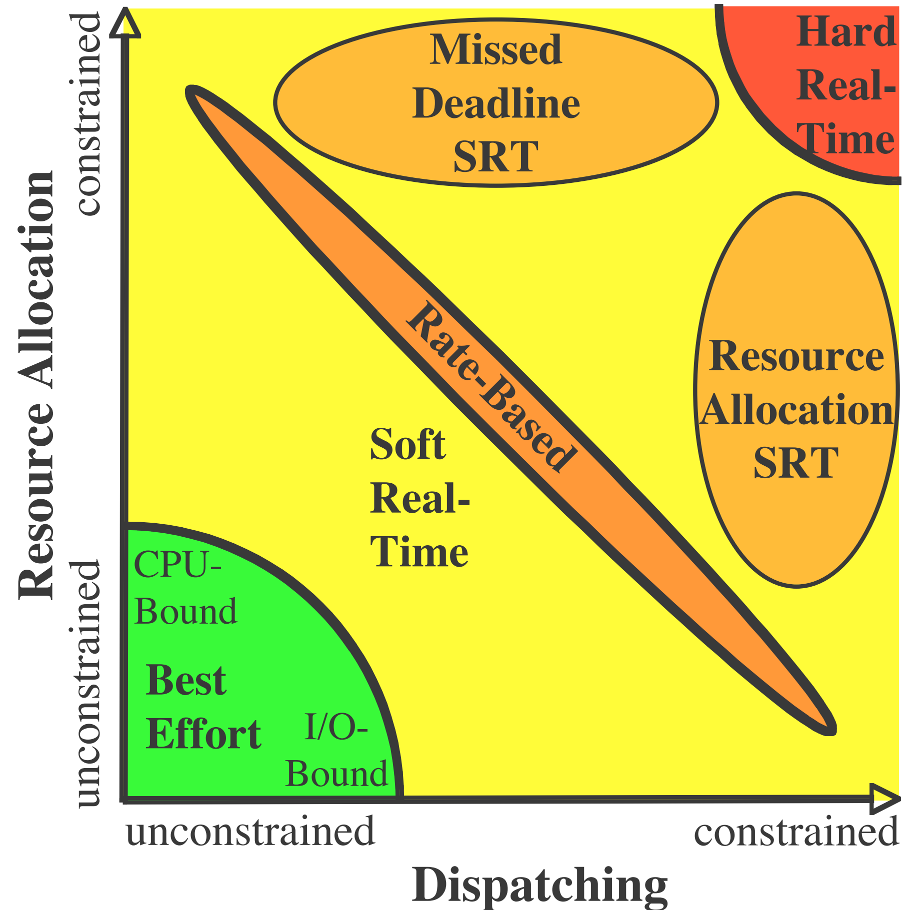
Observation

- Resource management consists of two distinct decisions
  - Resource Allocation: *How much* resources to allocate?
  - Dispatching: *When* to provide the allocated resources?

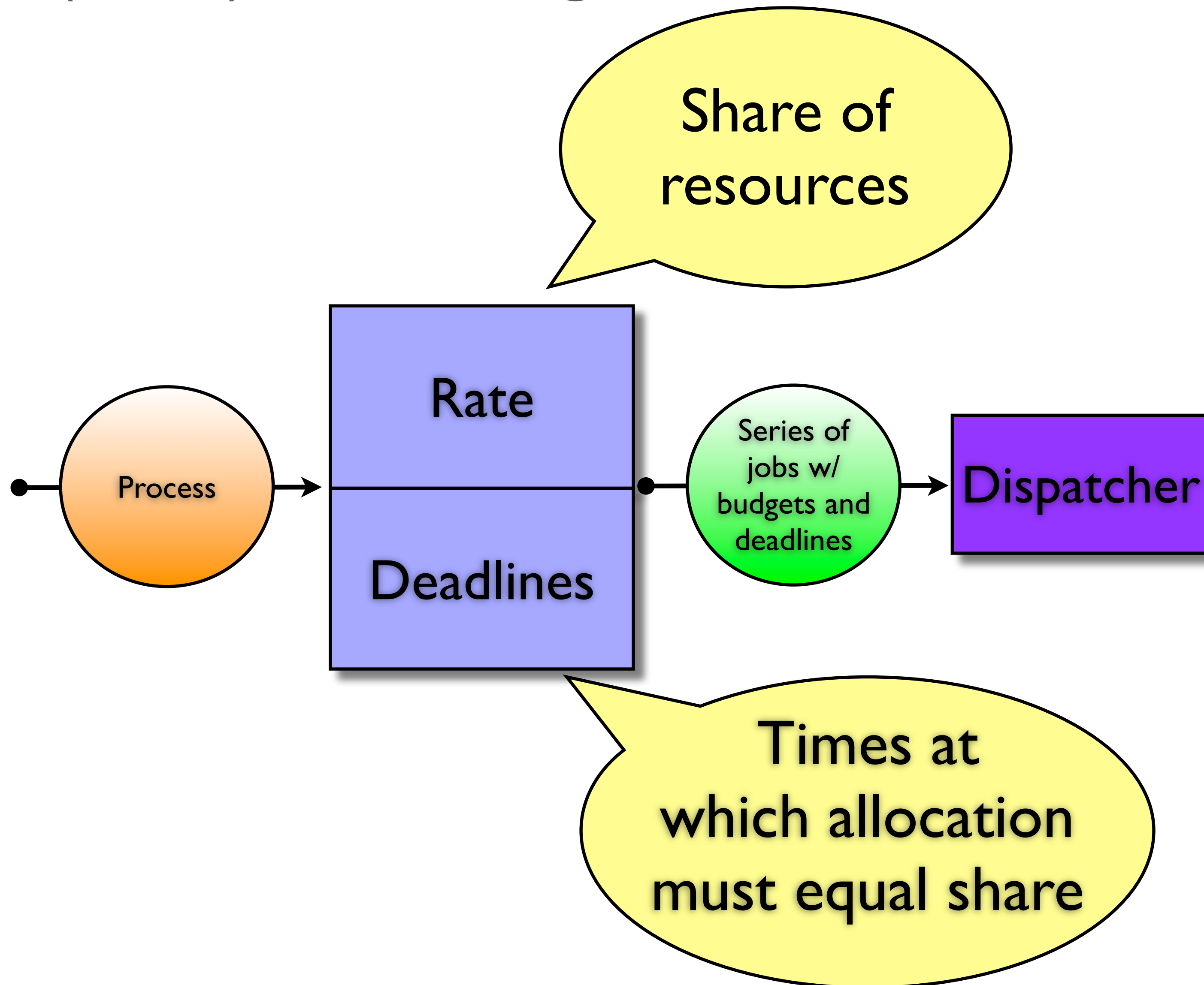
- Most resource managers conflate them
  - Best-effort, proportional-share, real-time

Separating them is powerful!

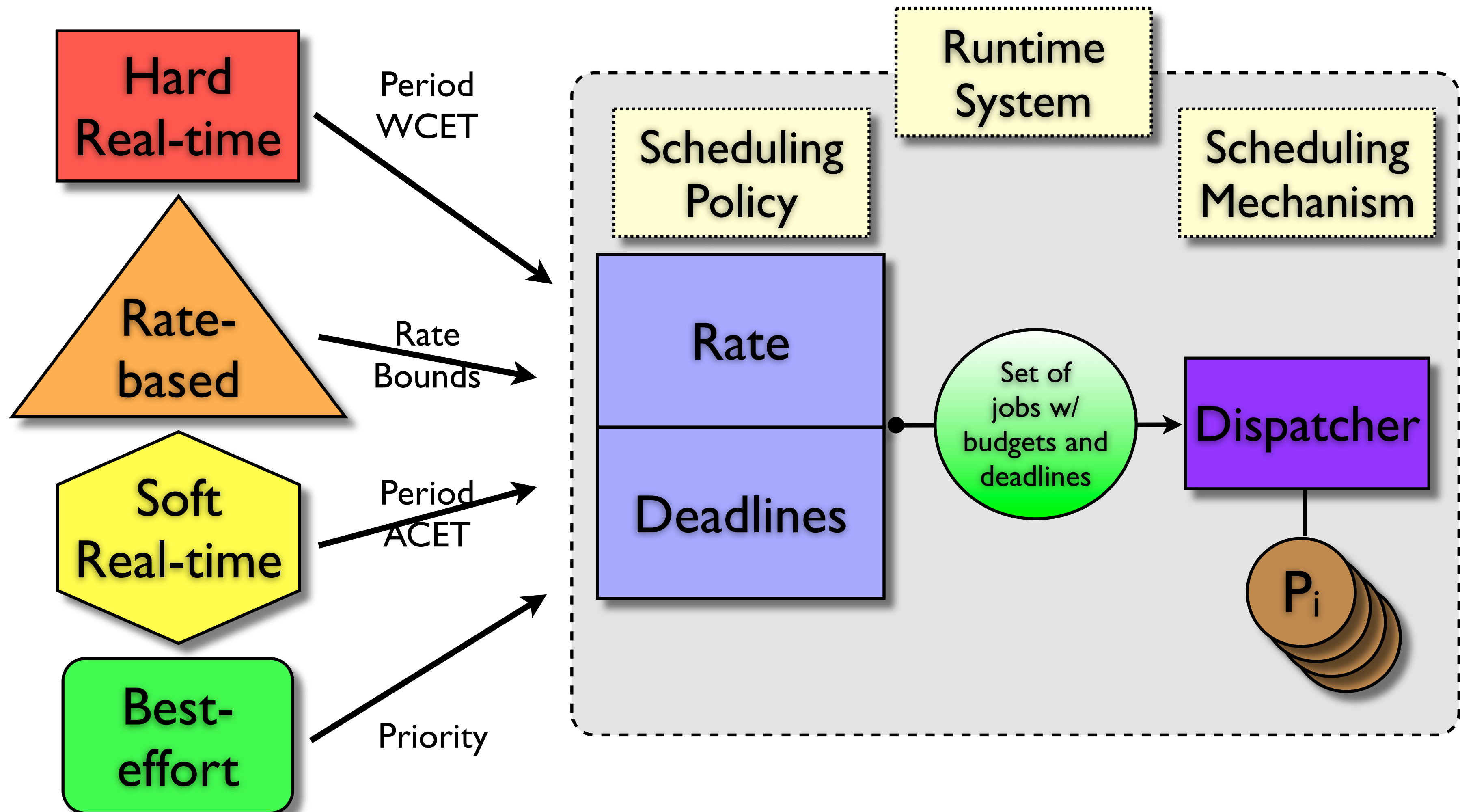
- Separately managing resource allocation and dispatching gives direct control over the delivery of resources to tasks
- Enables direct, integrated support of all types of timeliness needs



The resource allocation/dispatching (RAD) scheduling model

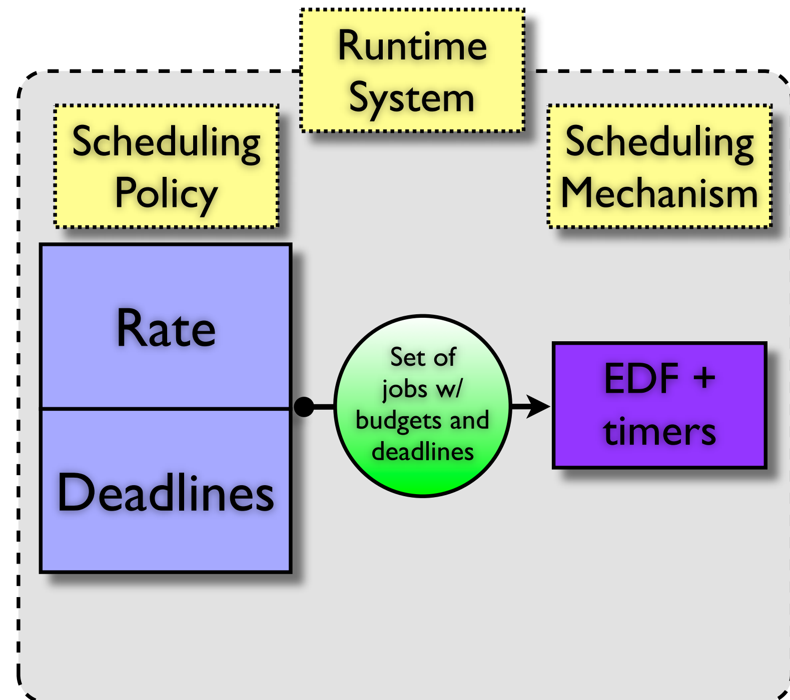


Supporting different timeliness requirements with RAD



Rate-Based Earliest Deadline (RBED) CPU scheduler

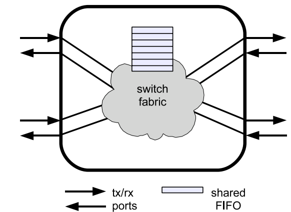
- Processes have rate & period
  - $\sum$rates $\leq$ 100%
  - Periods based on processing characteristics, latency needs, etc.
- Jobs have budget & deadline
  - budget = rate * period
  - Deadlines based on period or other characteristics
- Jobs dispatched via Earliest Deadline First (EDF)
  - Budgets enforced with timers
  - Guarantees all budgets & deadlines = all rates & periods



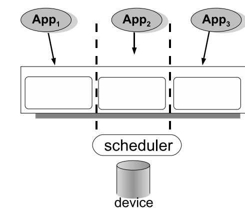
Adapting RAD to disk, network, and buffer cache

- Guaranteed disk request scheduling
  Anna Povzner

- Guaranteeing storage network performance
  Andrew Shewmaker (UCSC and LANL)



- Buffer management for I/O guarantees
  Roberto Pineiro

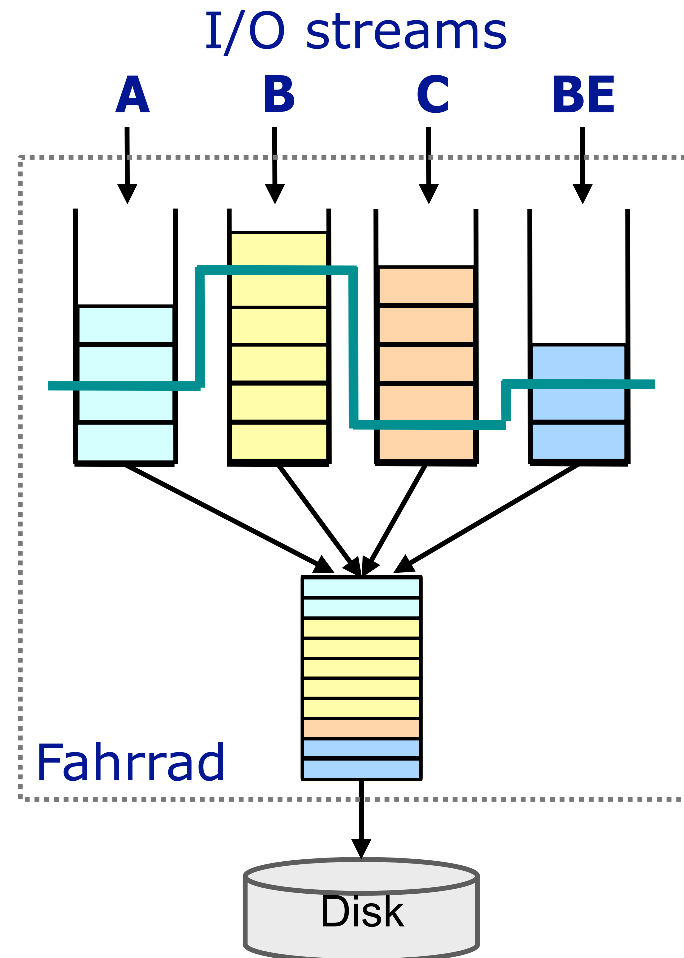


Guaranteed disk request scheduling

- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Good I/O performance
- Challenging because disk I/O is:
  - Stateful
  - Non-deterministic
  - Non-preemptable, and
  - Best- and worst-case times vary by 3–4 orders of magnitude

Fahrrad

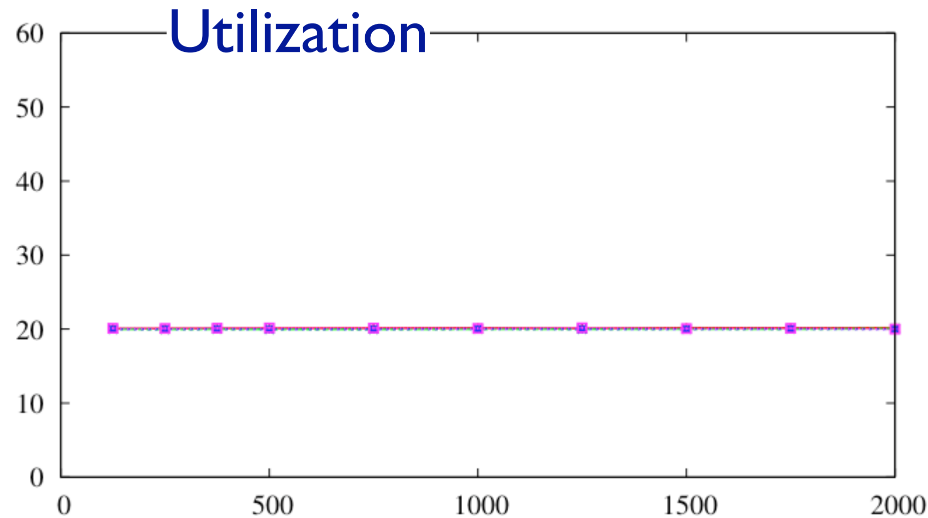
- Manages disk *time* instead of disk *throughput*
- Adapts RAD/RBED to disk I/O
- Reorders aggressively to provide good performance, without violating guarantees

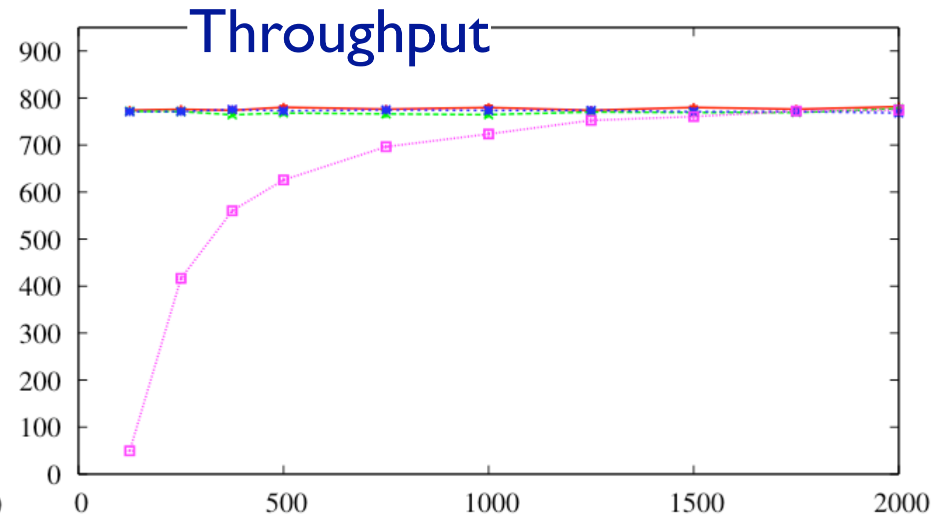


A bit more detail

- Reservations in terms of *disk time utilization* and *period* (granularity)
- All I/Os feasible before the earliest deadline in the system are moved to a Disk Scheduling Set (DSS)
- I/Os in the DSS are issued in the most efficient way
- I/O charging model is critical
- Overhead reservation ensures exact utilization
  - 2 WCRTs per period for "context switches"
  - 1 WCRT per period to ensure last I/Os
  - 1 WCRT for the process with the shortest period due to non-preemptability

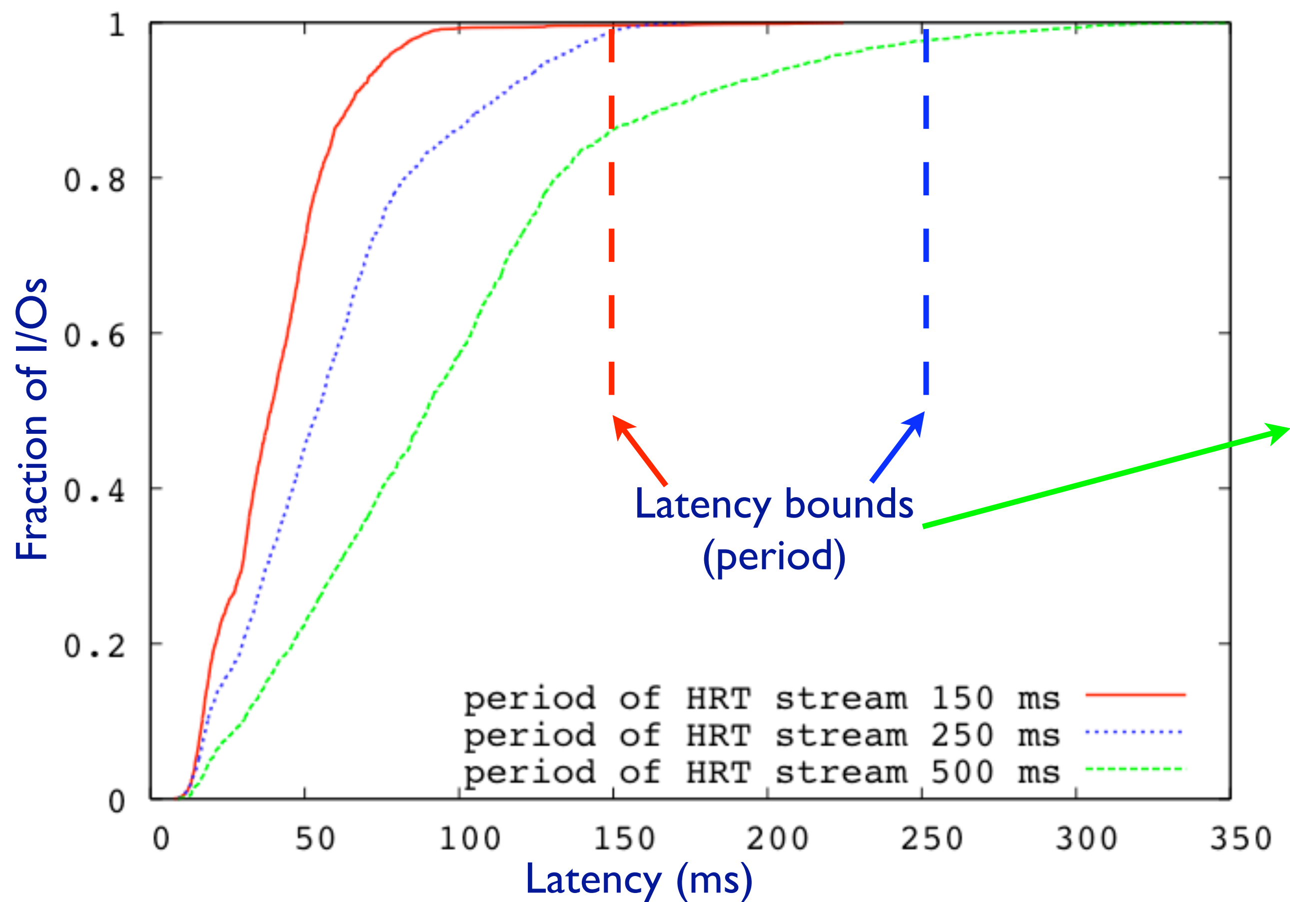
Fahrrad guarantees utilization, isolation, throughput





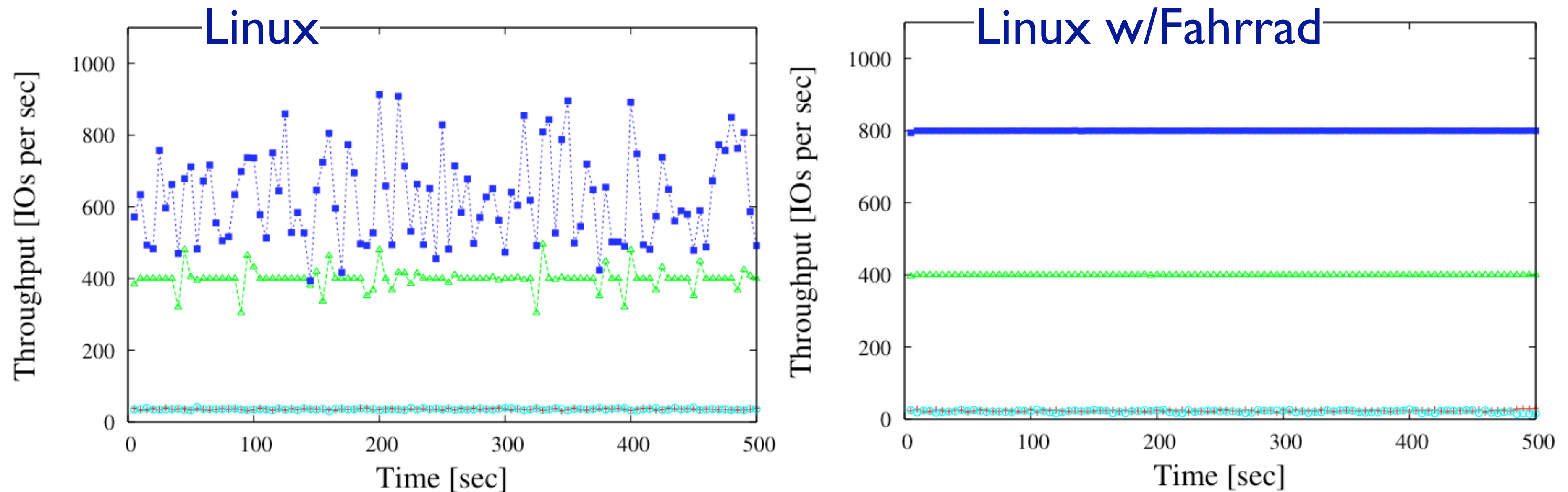
- 4 SRT processes, sequential I/Os, reservations = 20%
- 3 w/period = 2 seconds
- 1 with period 125 ms to 2 seconds
- Result: perfect isolation between streams

Fahrrad also bounds latency



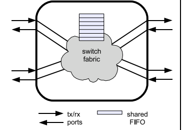
- Period bounds latency

Fahrrad outperforms Linux



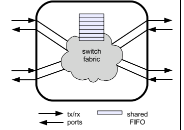
- Workload
  - Media 1: 400 sequential I/Os per second (20%)
  - Media 2: 800 sequential I/Os per second, (40%)
  - Transaction: short bursts of random I/Os at random times (30%)
  - Background: random (10%)
- Result: Better isolation AND better throughput

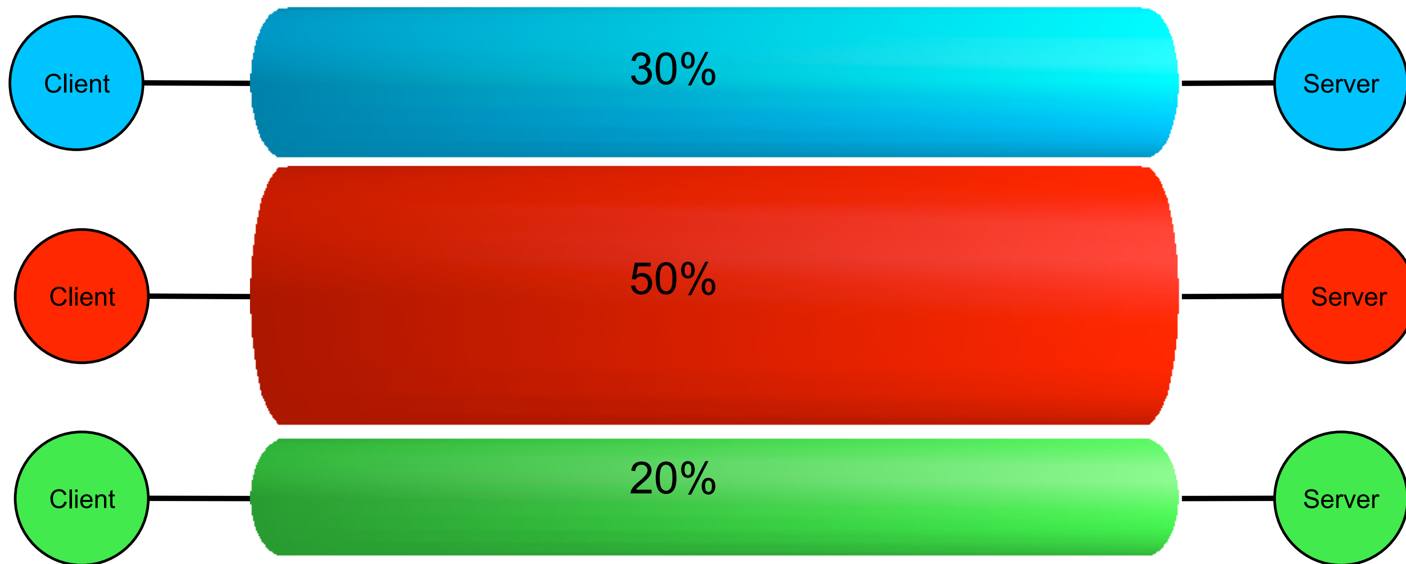
Guaranteeing storage network performance



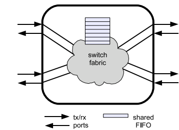
- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Good I/O performance
- Challenging because network I/O is:
  - Distributed
  - Non-deterministic (due to collisions or switch queue overflows)
  - Non-preemptable
- Assumption: closed network

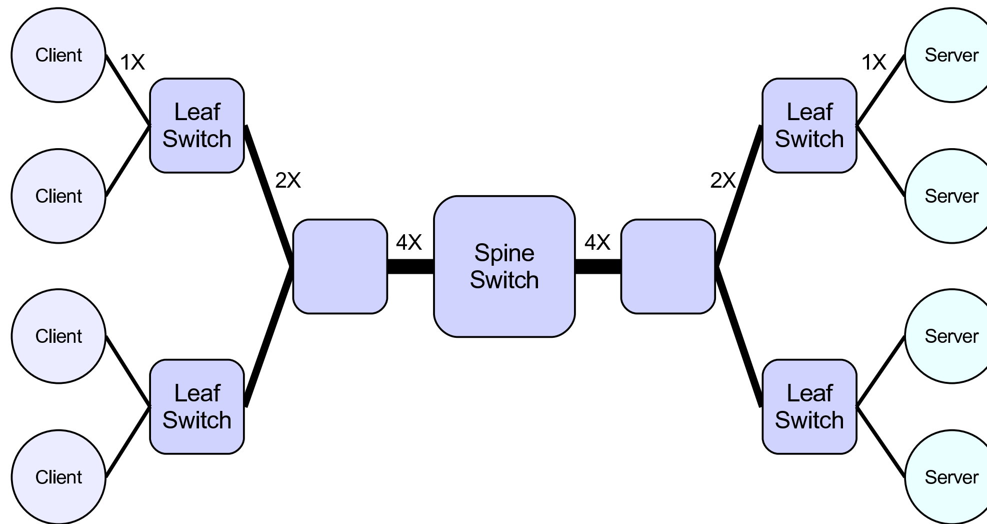
What we want

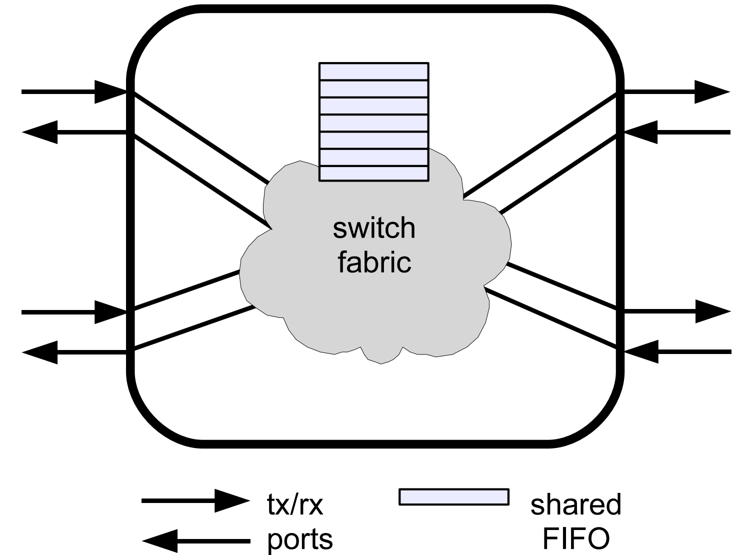




What we have

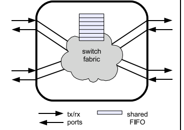






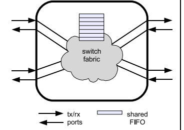
- Switched fat tree w/full bisection bandwidth
- Issue 1: Capacity of shared links
- Issue 2: Switch queue contention

Radon—RAD on the Network



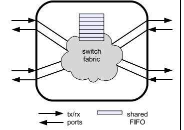
- RAD-based network reservations ~ disk reservations
  - *Network utilization* and *period*
- Fully decentralized
- Two controls:
  - Window size w = # of network packets per transmission window
  - Offset = delay before transmitting window
- Two pieces of information:
  - Dropped packets (TCP)
  - Forward delay (TCP Santa Cruz)

Flow control and congestion control

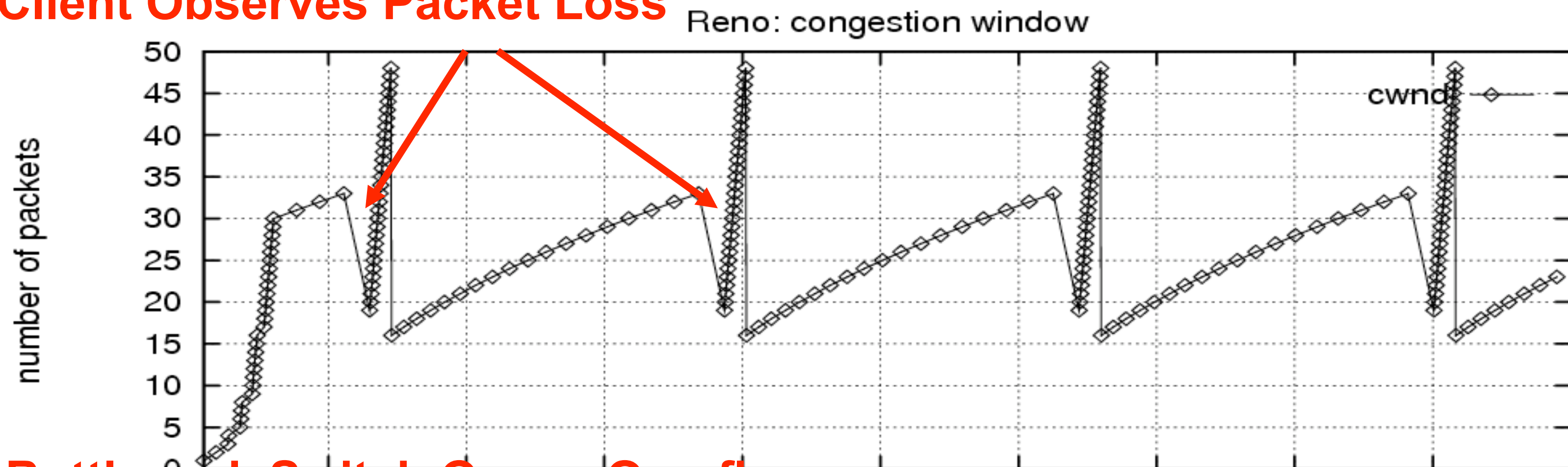


- Flow control determines how much data is transmitted
  - Fixed window sizes (TCP)
  - Adjust window size based on congestion (TCP Santa Cruz)
- Congestion control determines what to do when congestion is detected
  - Packet loss ➔ backoff then retransmit (TCP)
  - Δ Forward delay ➔ Δ window size (TCP Santa Cruz)

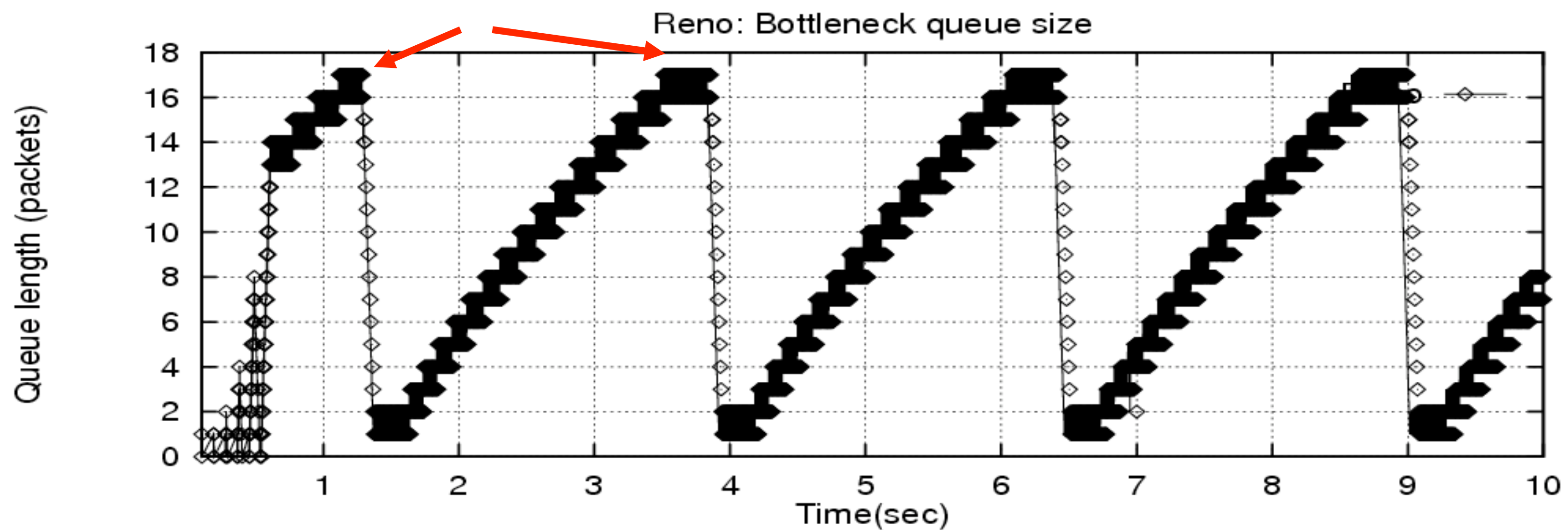
Packet loss based congestion control (TCP)



Client Observes Packet Loss

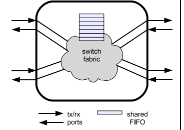


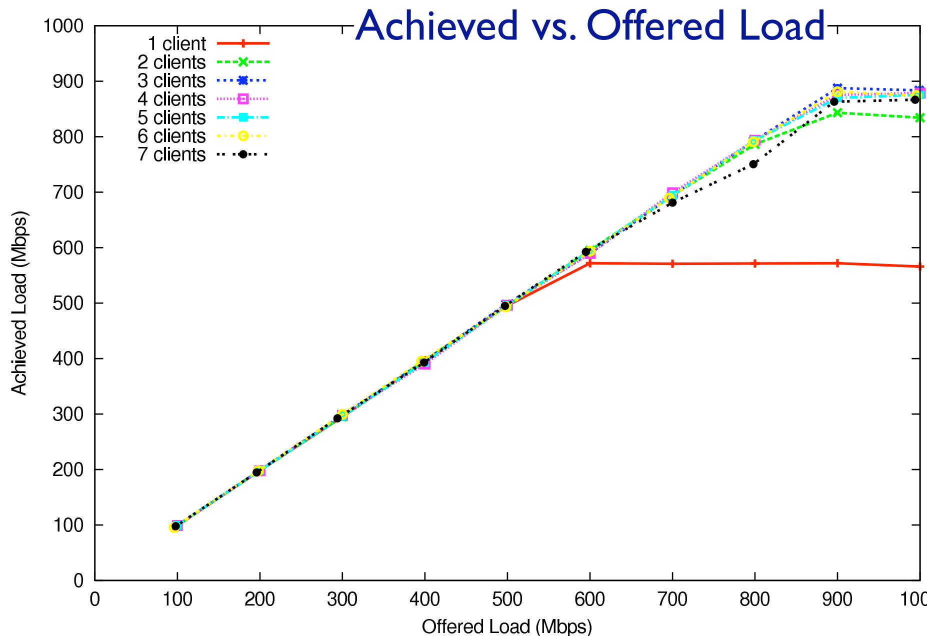
Bottleneck Switch Queue Overflows

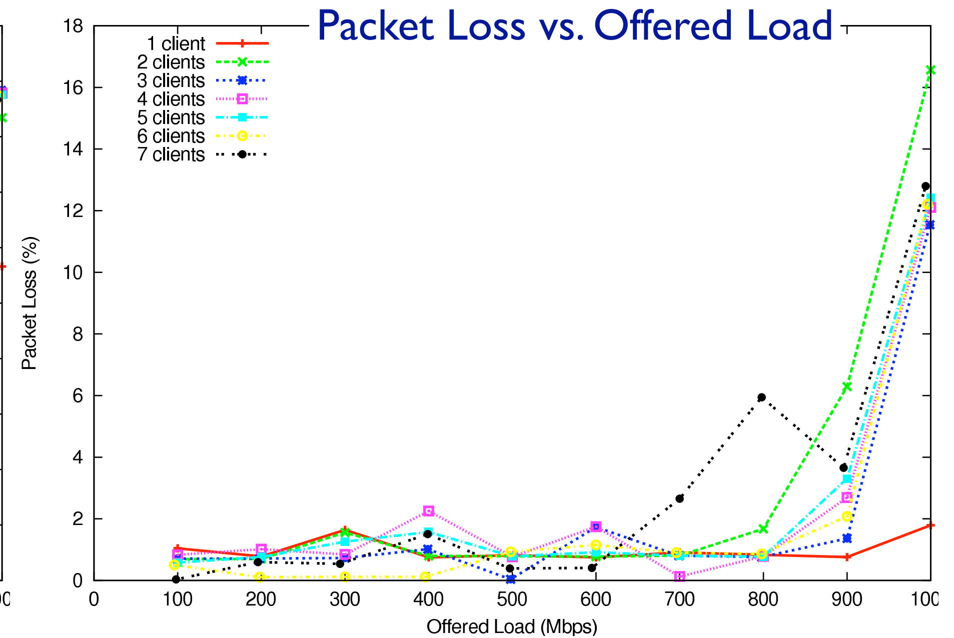


Improving TCP Congestion Control Over Internets with Heterogeneous Transmission Media (1999)
Christina Parsa, J.J. Garcia-Luna-Aceves. Proceedings of the 7th IEEE ICNP

Performance vs. offered load (w/nuttcp)

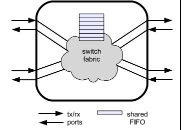




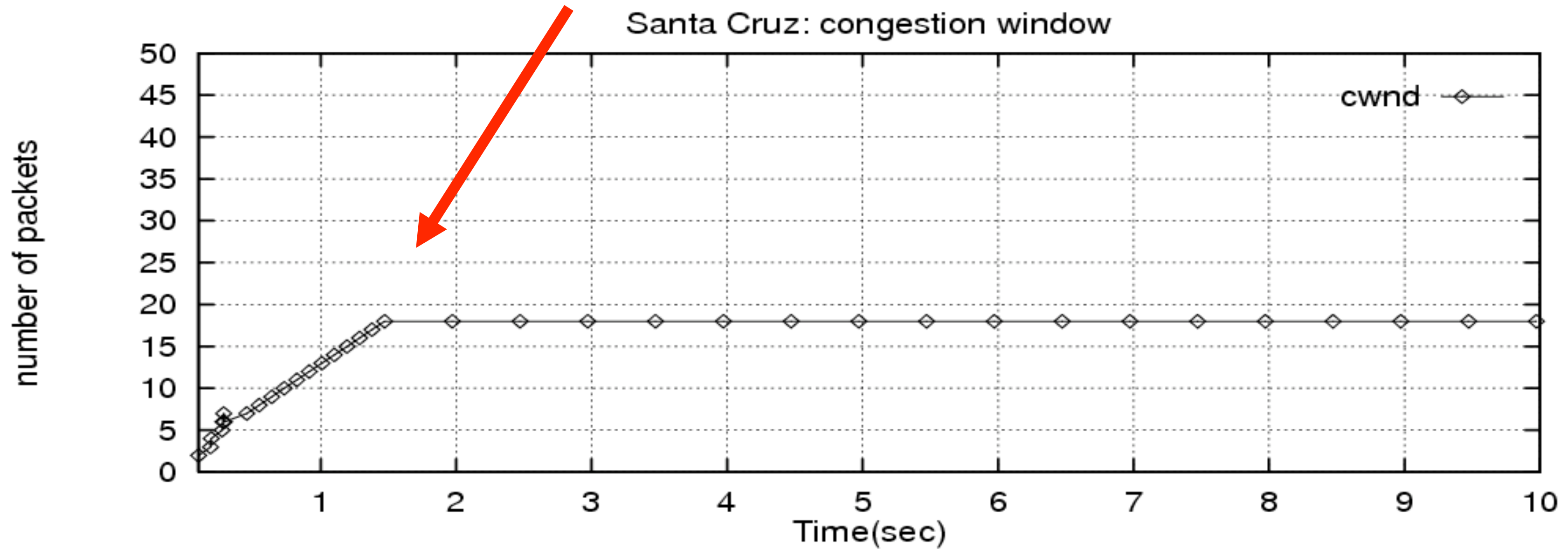


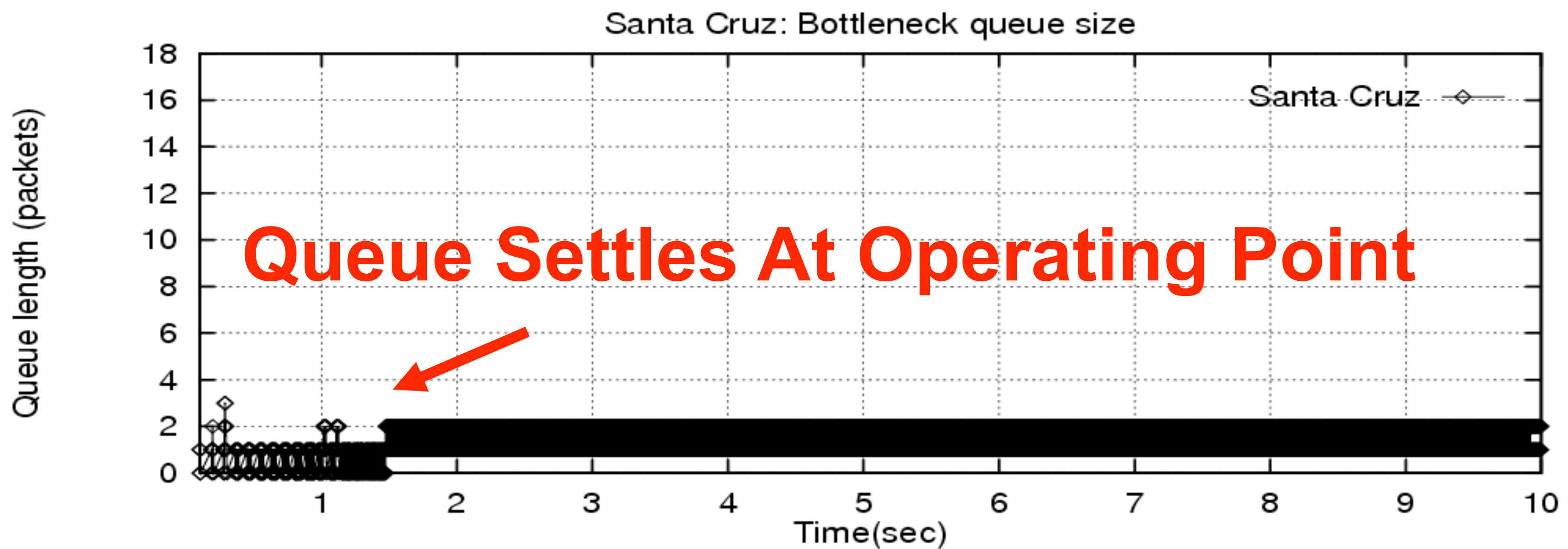
- The switch performs well below 800 Mbps
    - Regardless of the # of clients
    - Each client is limited to 600 Mbps due to SW/NIC issues

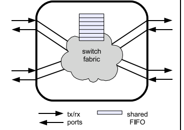
Delay based congestion control (TCP SC)



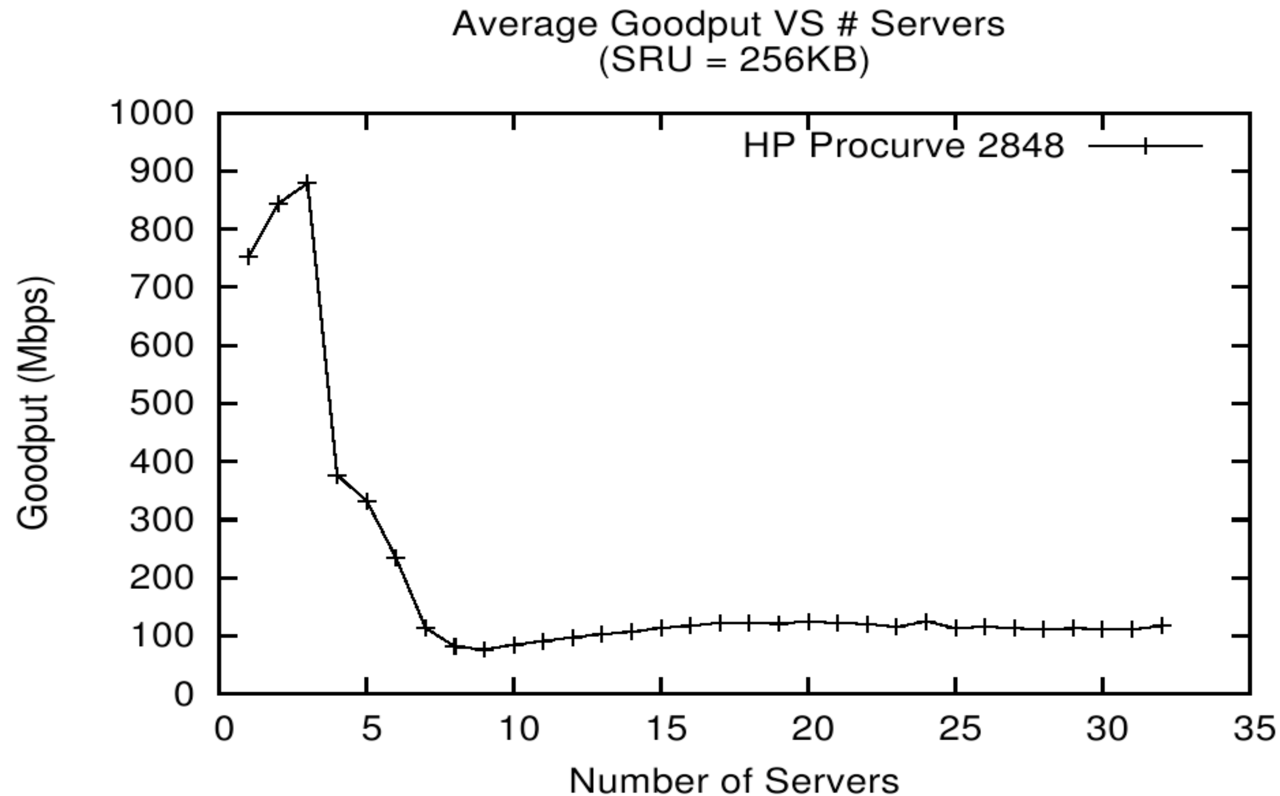
Client Observes Increased Delay





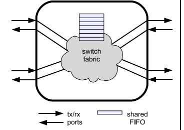


The incast problem



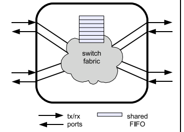
- **On Application-level Approaches to Avoiding TCP Throughput Collapse in Cluster-based Storage Systems (2007)**. Elie Krevat, Vijay Vasudevan, Amar Phanishayee, David G. Andersen, Gregory R. Ganger, Garth A. Gibson, Srinivasan Seshan. *Proceedings of the PDSW*

Three approaches



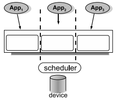
1. Fixed window size w/out offsets
   - Clients send data as it is available, up to per-period budget
   - Expected result: lots of packet loss due to incast/queue overflow
2. Fixed window size w/per-window offsets
   - Clients transmit each window with random offset in [0, *laxity* / *num_windows*]
3. Less Laxity More
   - Distributed approximation to Least Laxity First
   - No congestion ➔ increase window size
   - Congestion ➔ move toward $w_{opt} = (1\text{-}\%laxity)*w_{max}$

Early results

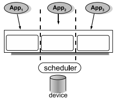


- Congestion detection
- Congestion response

Buffer management for I/O guarantees



- Goals
  - Hard and soft performance guarantees
  - Isolation between I/O streams
  - Improved I/O performance
- Challenging because:
  - Buffer is space-shared rather than time-shared
    - Space limits time guarantees
  - Best- and worst-case are opposite of disk
  - Buffering affects performance in non-obvious ways

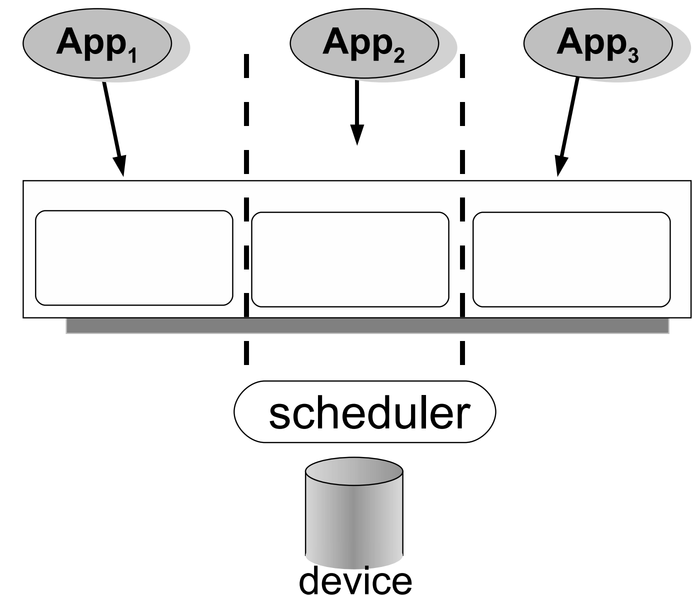


Buffering in storage servers

- Staging and de-staging data
  - Decouples sender and receiver
- Speed matching
  - Allows slower and faster devices to communicate
- Traffic shaping
  - Shapes traffic to optimize performance of interfacing devices
- Assumption: reuse primarily occurs at the client

Approach

- Partition buffer cache based on worst-case needs
  - 1–3 periods worth
- Use slack to prefetch reads and delay writes
- *Period transformation*: period into cache may be shorter than from cache to disk
- *Rate transformation*: rate into cache may be higher than disk can support



Data center performance management

- Data center perf. mgmt. is currently relatively *ad hoc* and reactive
- RAD gives us both tools and metrics for managing the performance of large distributed systems
  - CPU, client cache, network, server cache, disk
  - Predict, provision, guarantee, measure, monitor, mine
  - By job, node, resource, I/O stream, etc.

Goals

1. A first-principles model for data center perf. mgmt.
2. Full-system metrics for measuring performance in client processing nodes, buffer cache, network, server buffer cache, and disk
3. Performance visualization by application, client node, reservation, or device
4. Application workload profiling and modeling
5. Full system performance provisioning and management based on all of the above
6. Online machine-learning based performance monitoring for real-time diagnostics

Conclusion

- Distributed I/O performance management requires management of many separate components
- An integrated approach is needed
- RAD provides the basis for a solution
- RAD has been successfully applied to several resources: CPU, disk, network, and buffer cache
- We are on our way to an integrated solution
- There are many useful applications: Data centers, TorMesh, virtualization, ...