A QoS Aware Non-work-conserving Disk Scheduler

Pedro E. Rocha
Luis C. E. Bona
Federal University of Paraná
Brazil

IEEE MSST'12

Why QoS Aware?

- To be able to share a disk among users whilst providing tight QoS guarantees.
- Enforce *performance isolation:*

the performance experienced by an application (VM) should not suffer due to variations in the workload from other applications.

- Important in virtualized systems

Why QoS Aware?

- We consider QoS aware different from proportional share:
  - QoS aware can specify tight and independent QoS guarantees in terms of:
    - Bandwidth
    - Delay
    - Bursts
  - Proportional share
    - I/O Priorities
    - Disk share (in %) → disk performance is very difficult to predict

Why Non-work-conserving?

- Work-conserving schedulers suffer with *deceptive idleness*
- Non-work-conserving schedulers prevent deceptive idleness by predicting future requests:

A request that is soon to arrive might be closer to the current disk head position than other pending requests

- **The solution**: after serving a (*synchronous and sequential*) request, keep the disk idle

Previous Work

- CFQ – Complete Fairness Queuing
  - Non-work-conserving ✓
  - Proportional share (I/O priorities) ✗
- BFQ – Budget Fair Queuing
  - Non-work-conserving ✓
  - Proportional share (a disk share per application) ✗
- pClock
  - Work-conserving ✗
  - QoS aware (tags per request) ✓

HTBS

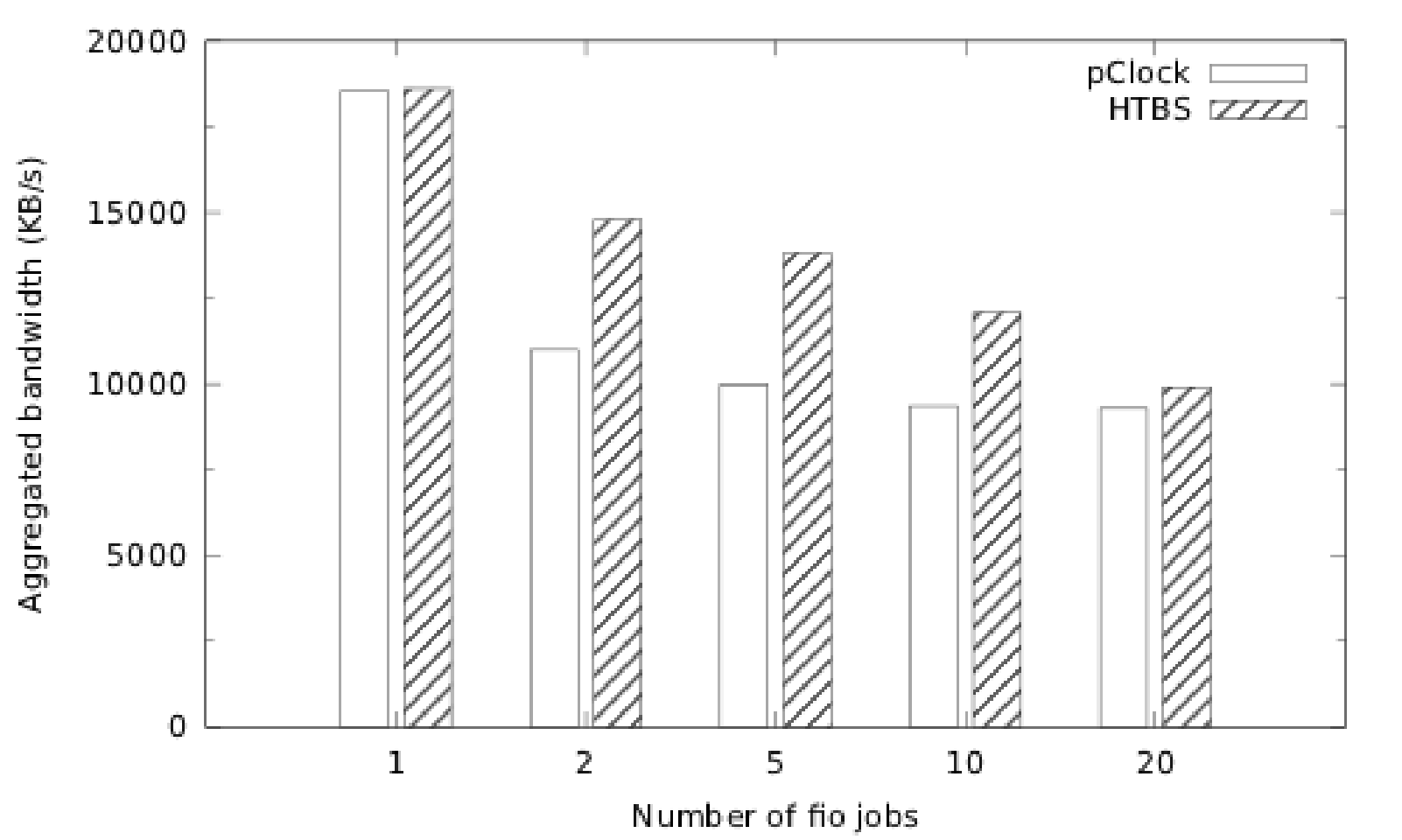
- High-throughput Token Bucket Scheduler
- Assigns tags per request in a fair-queuing like fashion (similar to pClock):
  - Request queue per application
  - Start and finish tags per request
- Non-work-conserving dispatch order.

- More details in the paper...

Experimental Setup

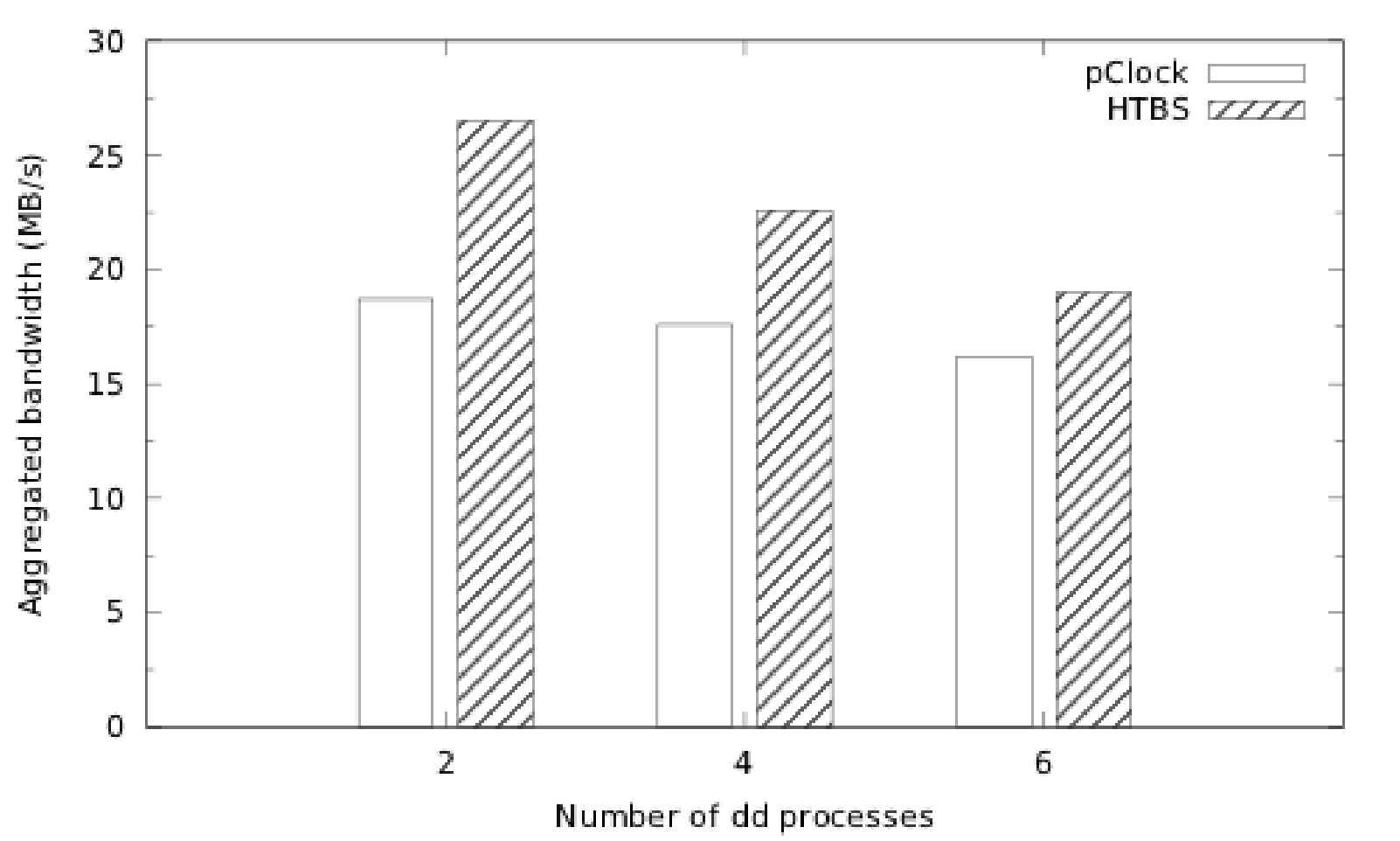
- AMD Athlon 2800 MHz *dual-core*, 4 GB RAM
- Low-end HDD → Samsung HD080HJ SATA, 80GB, 7200 rpm
- We implemented HTBS and pClock for Linux
- Microbenchmark → fio and dd
- Two experiments:
  - Measure how future request prediction increases throughput
  - Show that a QoS-aware work-conserving scheduler misses QoS guarantees in synchronous workloads

Experimental Results



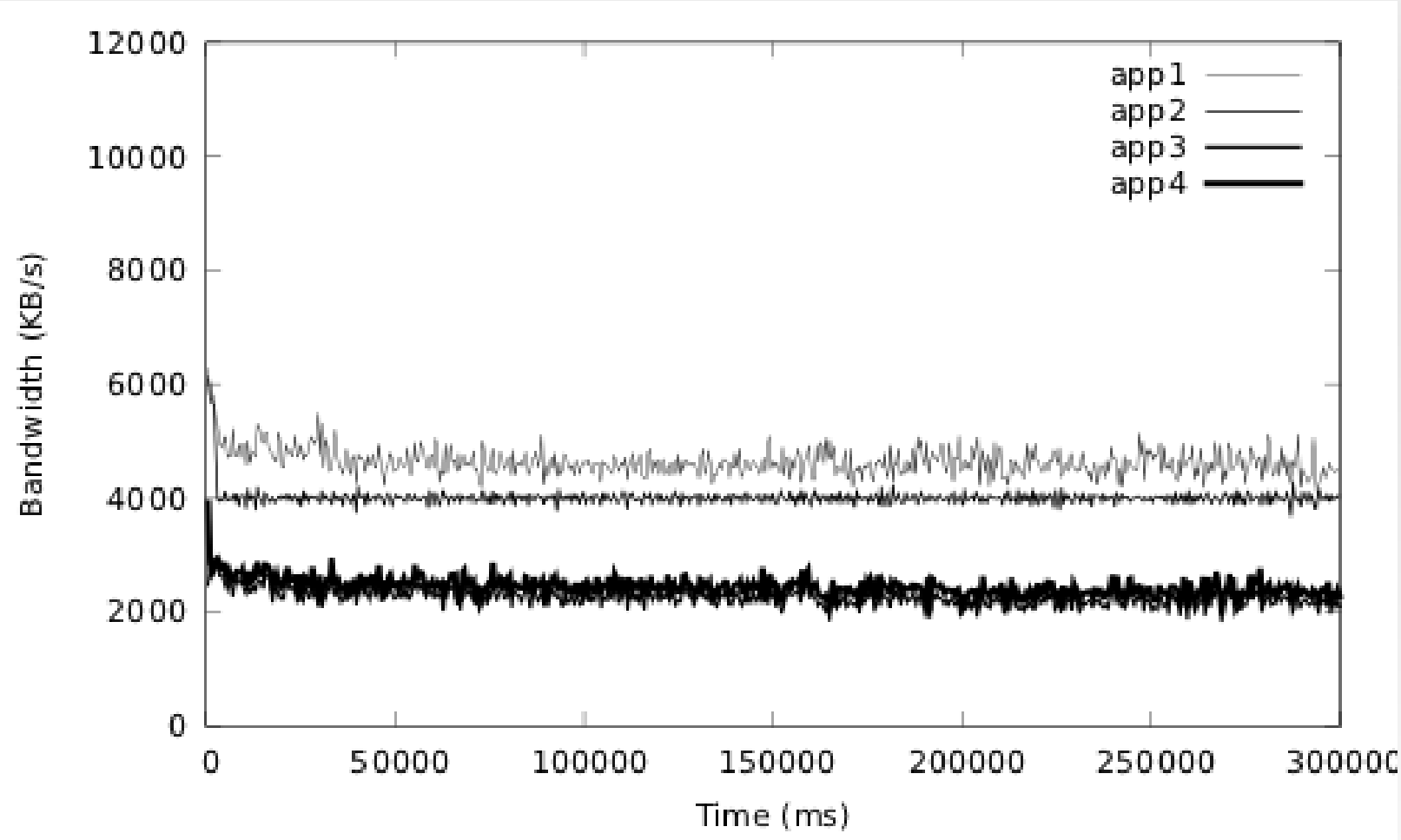
Aggregated bandwidth achieved by pClock and HTBS using fio benchmark

Experimental Results



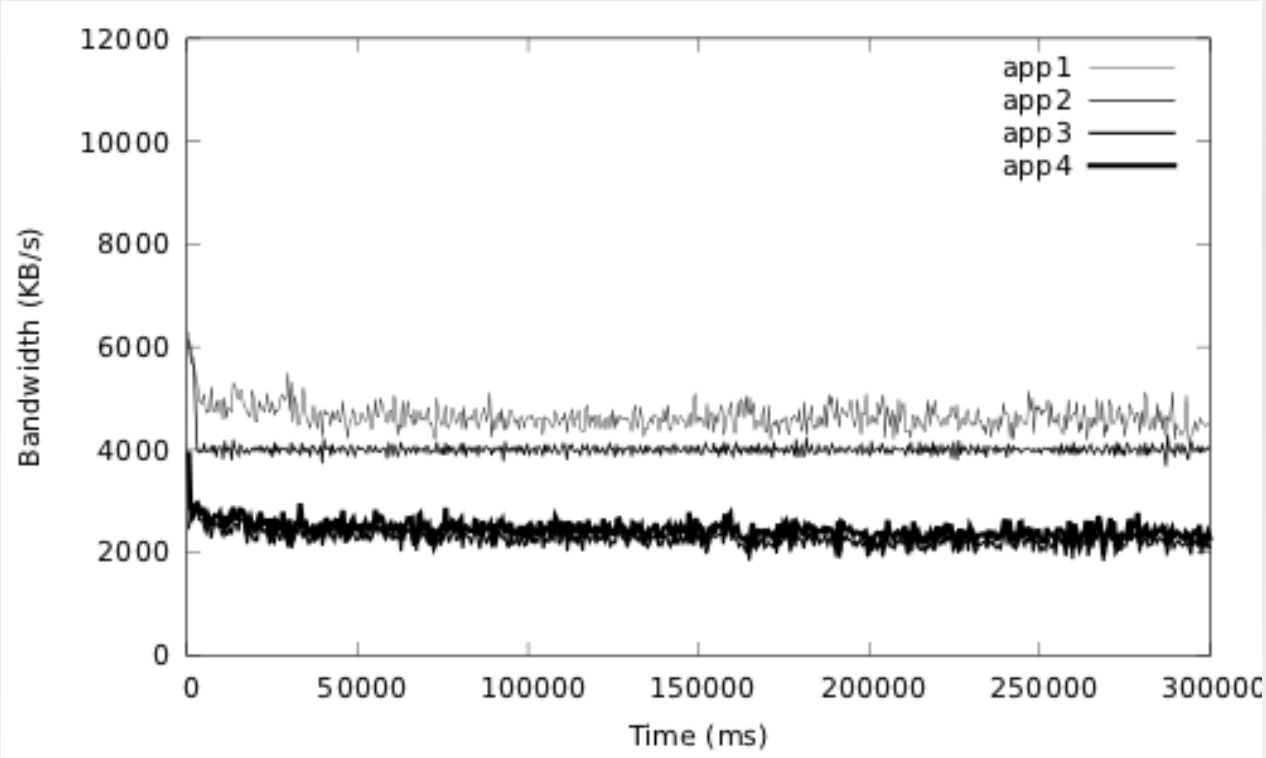
Aggregated bandwidth achieved by pClock and HTBS using *dd* processes

Experimental Results



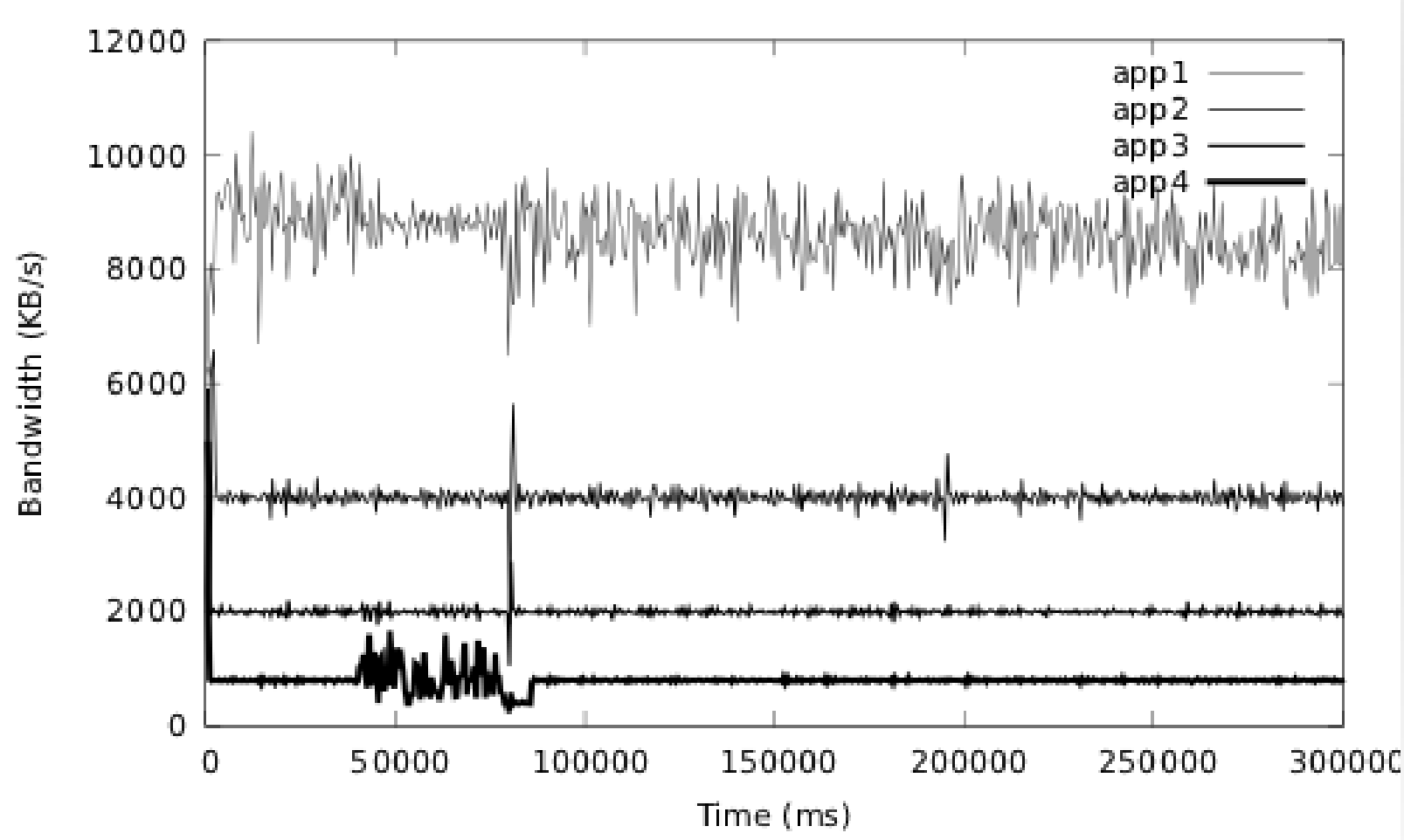
pClock: four synchronous jobs with different bandwidth attributes (8800, 4000, 2000 and 800 KB/s)

Experimental Results



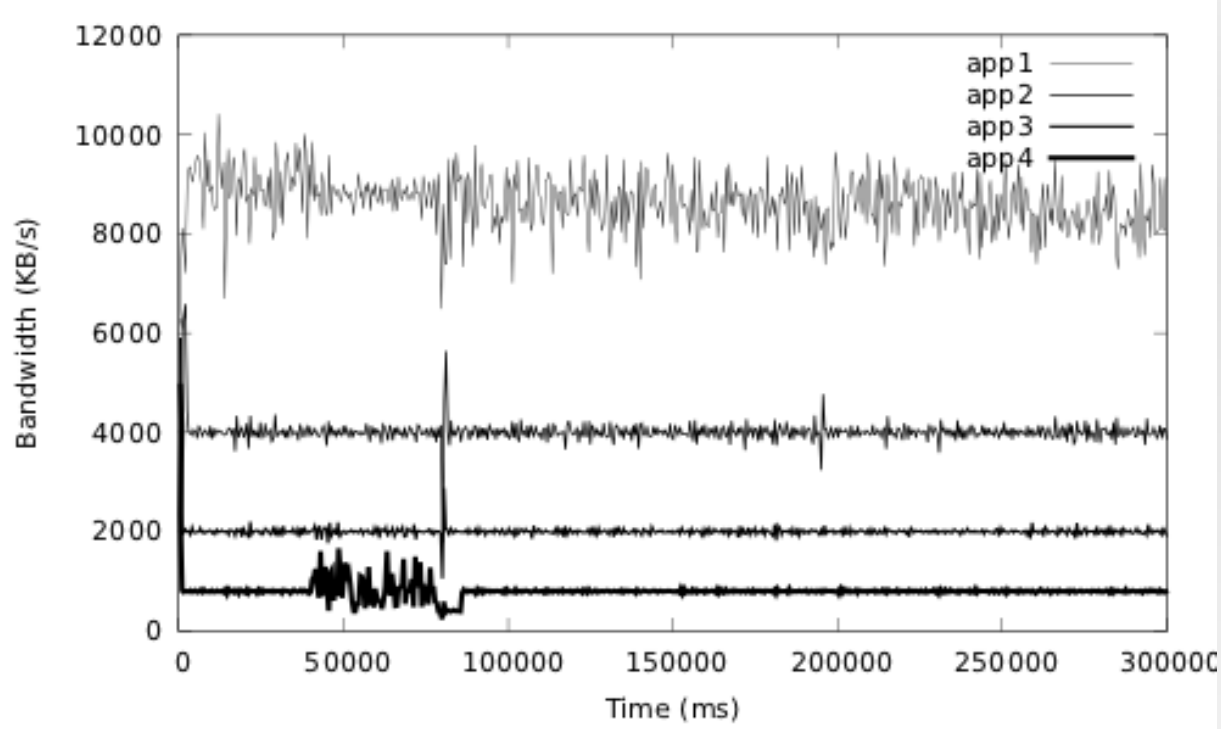
- Since there is a short amount of time between synchronous requests, a work-conserving scheduler cannot dispatch several requests from the same application, even if its guarantees are higher.
- QoS guarantees are missed

Experimental Results



HTBS: four synchronous jobs with different bandwidth attributes (8800, 4000, 2000 and 800 KB/s)

Experimental Results



- HTBS could meet QoS guarantees, since it can dispatch several requests from the same application (depending on its QoS guarantees, and up to B_{max})
- QoS guarantees are met

Future Directions

- Do some more experimentation using macrobenchmarks
  - Filebench
  - TCP
  - DVDStore
- Integrate with VMMs to provide QoS guarantees to VMs without decrease system overall throughput

Conclusions

- We presented HTBS, a new non-work-conserving QoS aware disk scheduler
- Through experiments with a Linux implementation, we showed that:
  - HTBS increases throughput when compared to other QoS-aware schedulers
  - HTBS can provide QoS guarantees even with synchronous workloads, unlike previous work

Thanks!

Questions?