ORACLE®

CFQ IO Scheduler

Jens Axboe <jens.axboe@oracle.com>
Consulting Member of Staff

Outline

- Drive characteristics
- History of CFQ
- CFQ design
  - Work load simulations
  - Data structures
  - Algorithms
- Some benchmark results

Drive performance
Characteristics

- How the drive can help us
  - Command queuing (NCQ, TCQ)
    - Optimal seek pattern
    - Eliminate/reduce rotational latency
- Where the drive is mostly helpless
  - Associated/dependent requests
  - Competing IO streams
  - Fairness

Current drive performance

	Avg seek	Avg rot. delay	Transfer speed
5400RPM	14 msec	5.6 msec	40 MiB/s
7200RPM	9 msec	4.2 msec	60 MiB/s
15000 RPM	4 msec	2.0 msec	90 MiB/s

	4K xfer	4K random read	64K random read
5400RPM	98 μsec	203 K/sec	3020 K/sec
7200RPM	65 μsec	301 K/sec	4490 K/sec
15000 RPM	43 μsec	662 K/sec	9600 K/sec

Linux IO Scheduling

Schedulers

- Currently includes 4 IO schedulers
  - noop
    - No sorting, does request merging
  - deadline
    - Assigns deadlines to requests
    - Otherwise CSCAN with a few twists
      - Direction batching
  - anticipatory ("as")
    - Basic functional algorithm is like deadline
    - Adds request anticipation
  - CFQ
    - We'll get to that

History of CFQ
CFQ v1

- Inspired by SFQ, stochastic fair queuing
  - Fixed number of buckets
  - Per-process buckets (tgid, really) → Complete FQ
- First IO scheduler to tie process and IO
  - Linux IO model async
- Round robin of busy processes
  - Work conserving
- Approx 700 lines of code
- Merged in 2.6.6

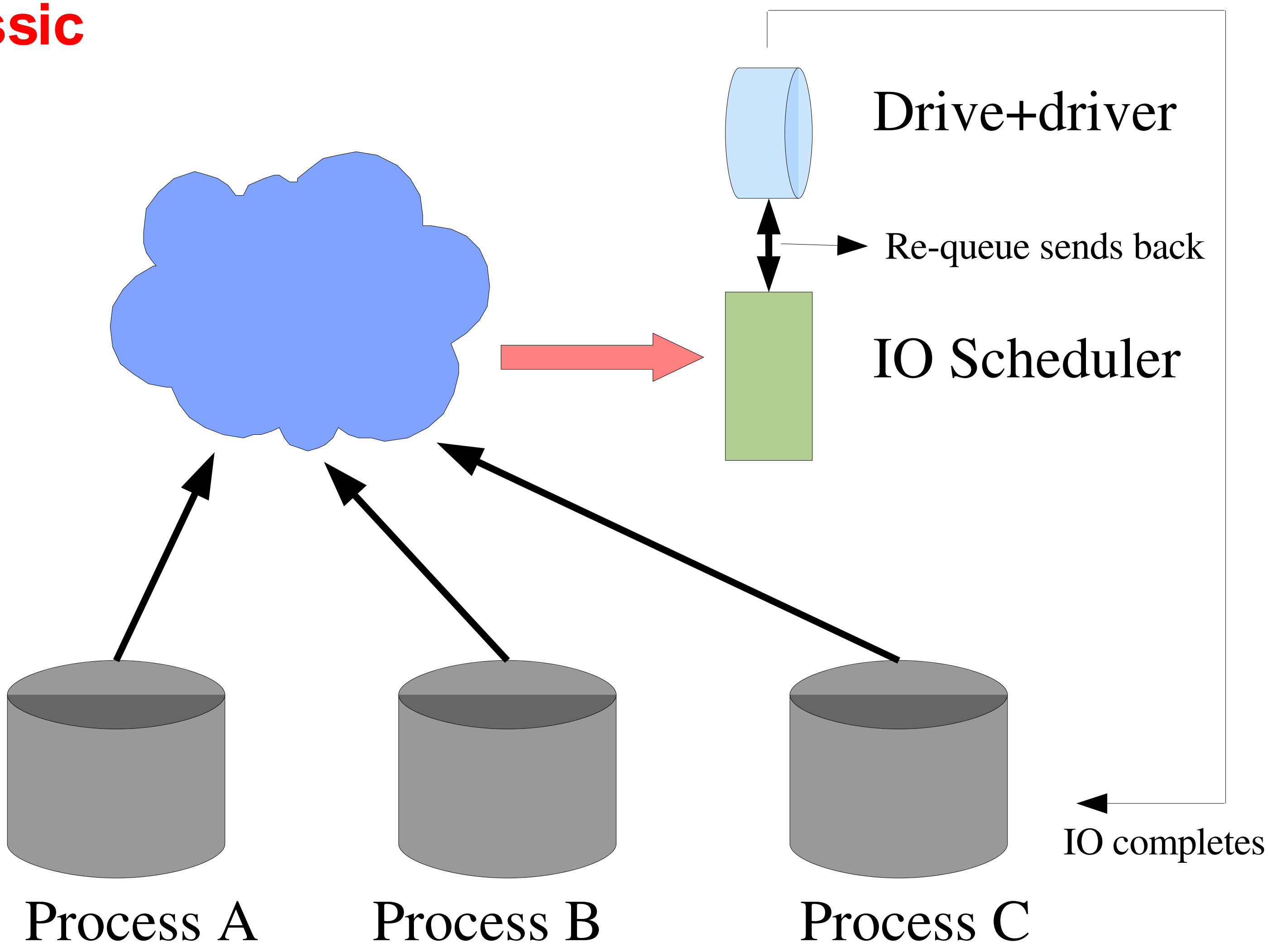
History of CFQ
CFQ v2

- Added persistent process contexts
  - Built on the 'as' introduced process io contexts
  - Fairness across process life time
- Addressed inter-queue fairness
- Approx 1800 lines of code
- Merged in 2.6.10

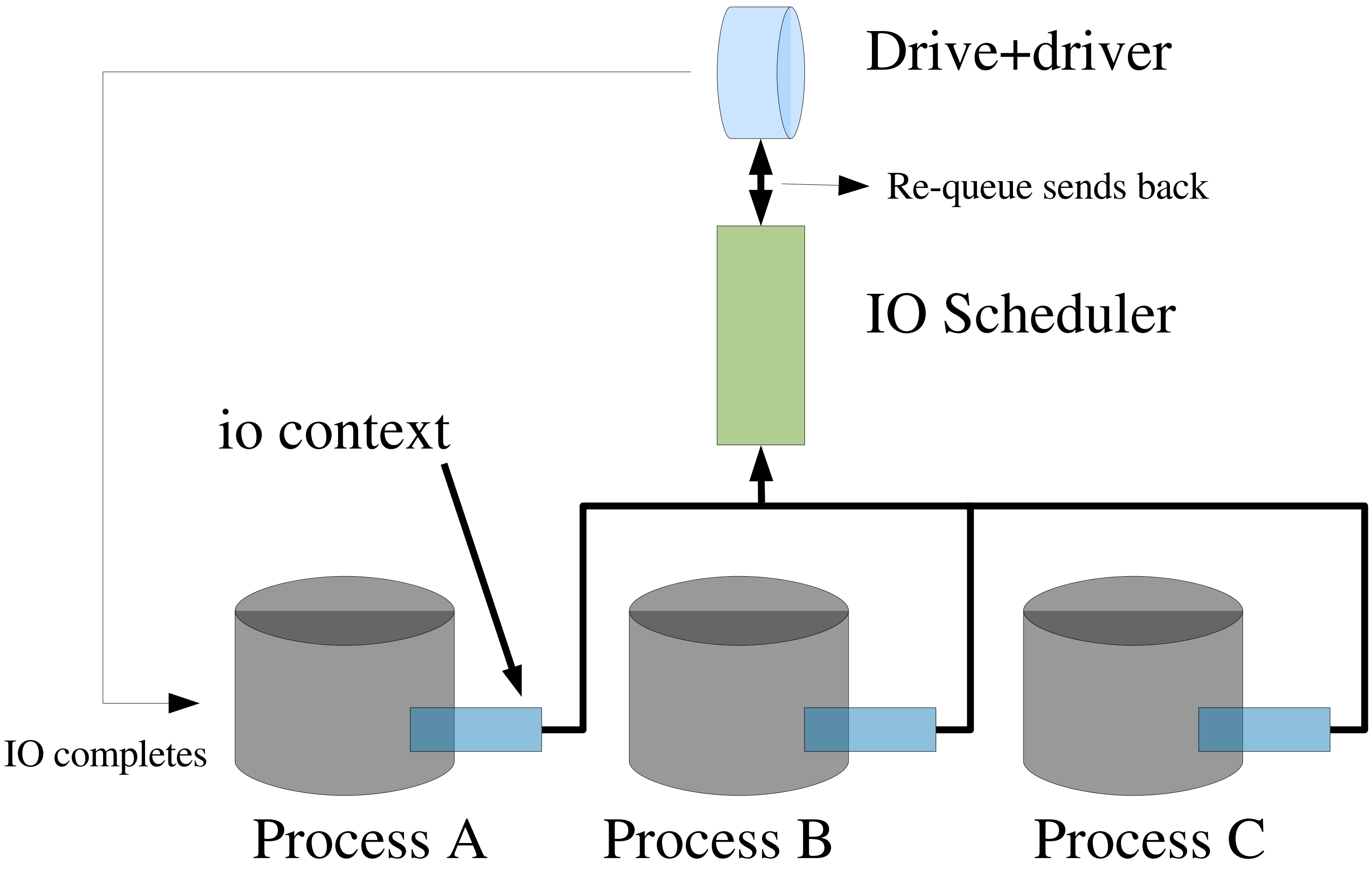
History of CFQ
CFQ v3

- Time slice design
  - Time based accounting, not request
  - Fairness across different io patterns
- Support for IO priorities
- Approx. 2200 lines of code
- Merged in 2.6.13

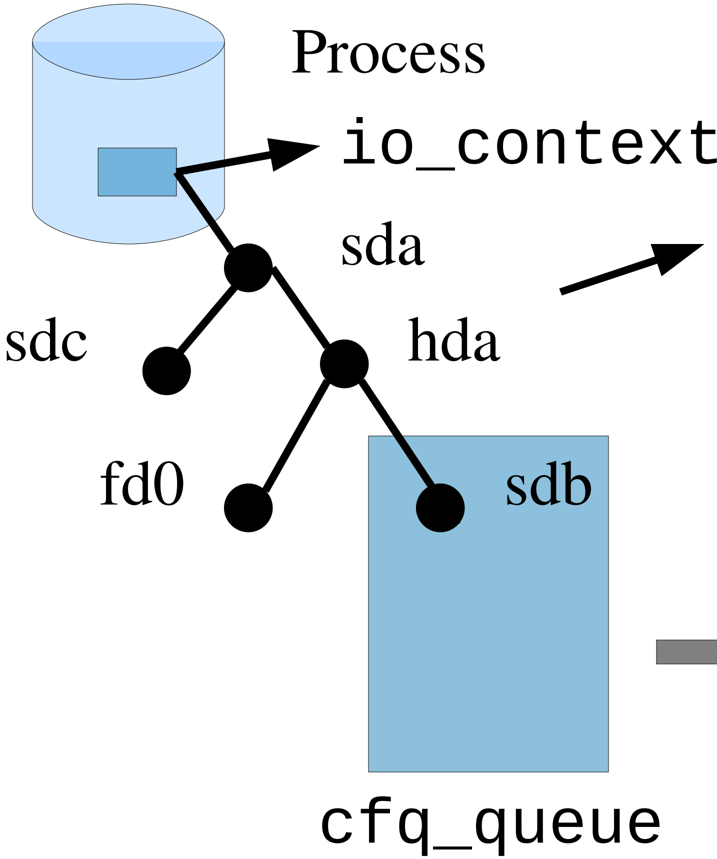
IO Hierarchy
Classic



IO Hierarchy
CFQ

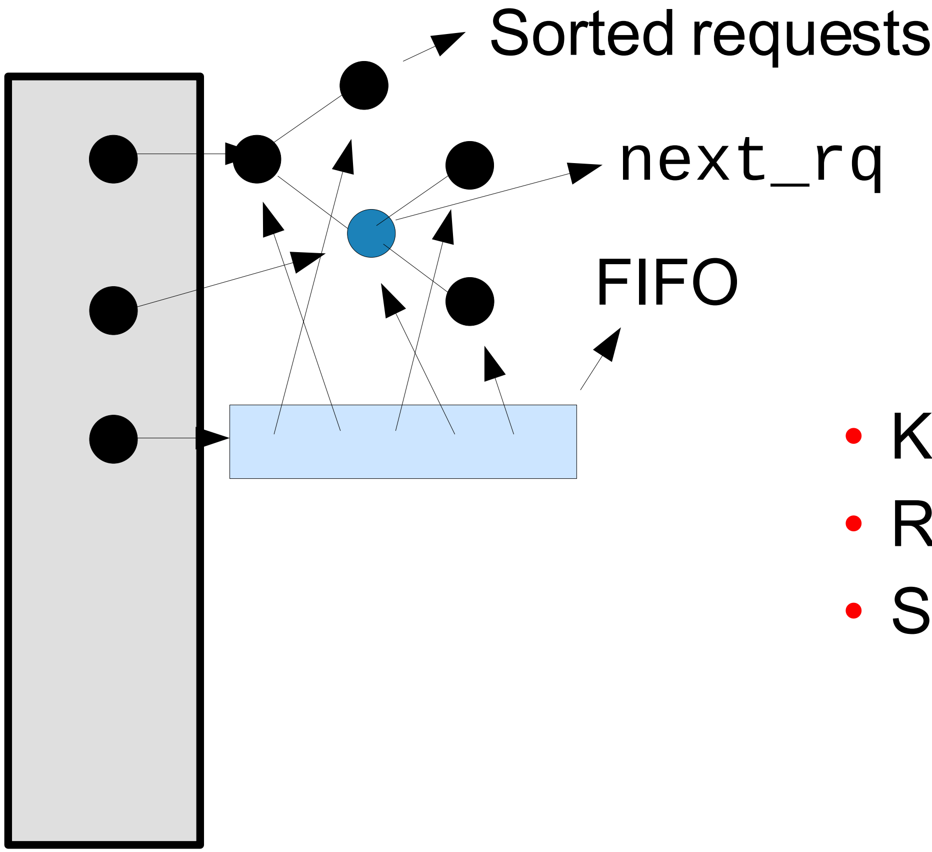


Process <-> CFQ mapping



- Request sorting
- FIFO
- `->next_request`
- `cfq_queue` hash
- Priority information, state

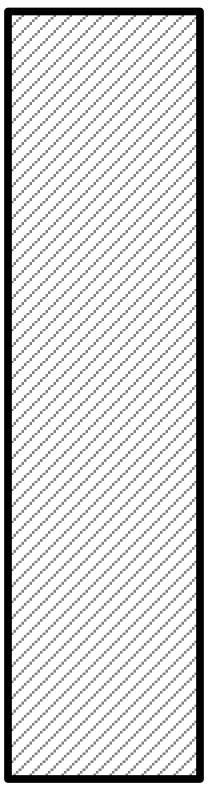
struct cfq_queue



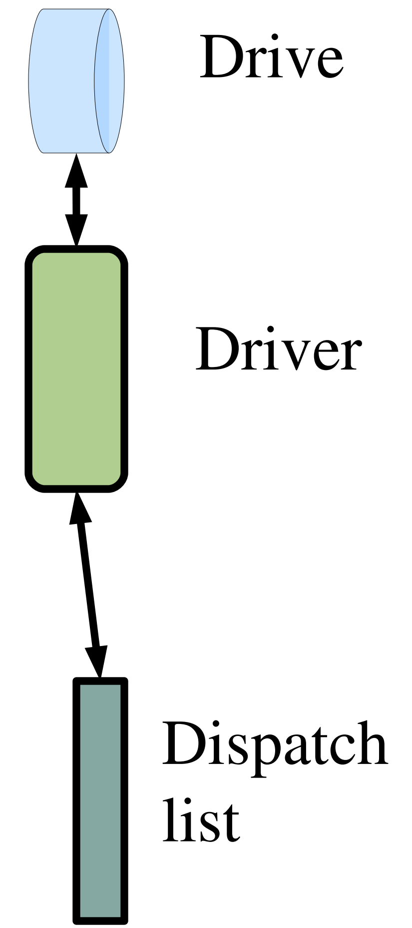
- Key
- References
- State information

CFQ per-queue data

`cfq_data`



- best-effort lists
- busy list
- current list
- idle list
- hash of `cfq_queue`
- `io_context` pointer
- state and settings



Time slices
Concept

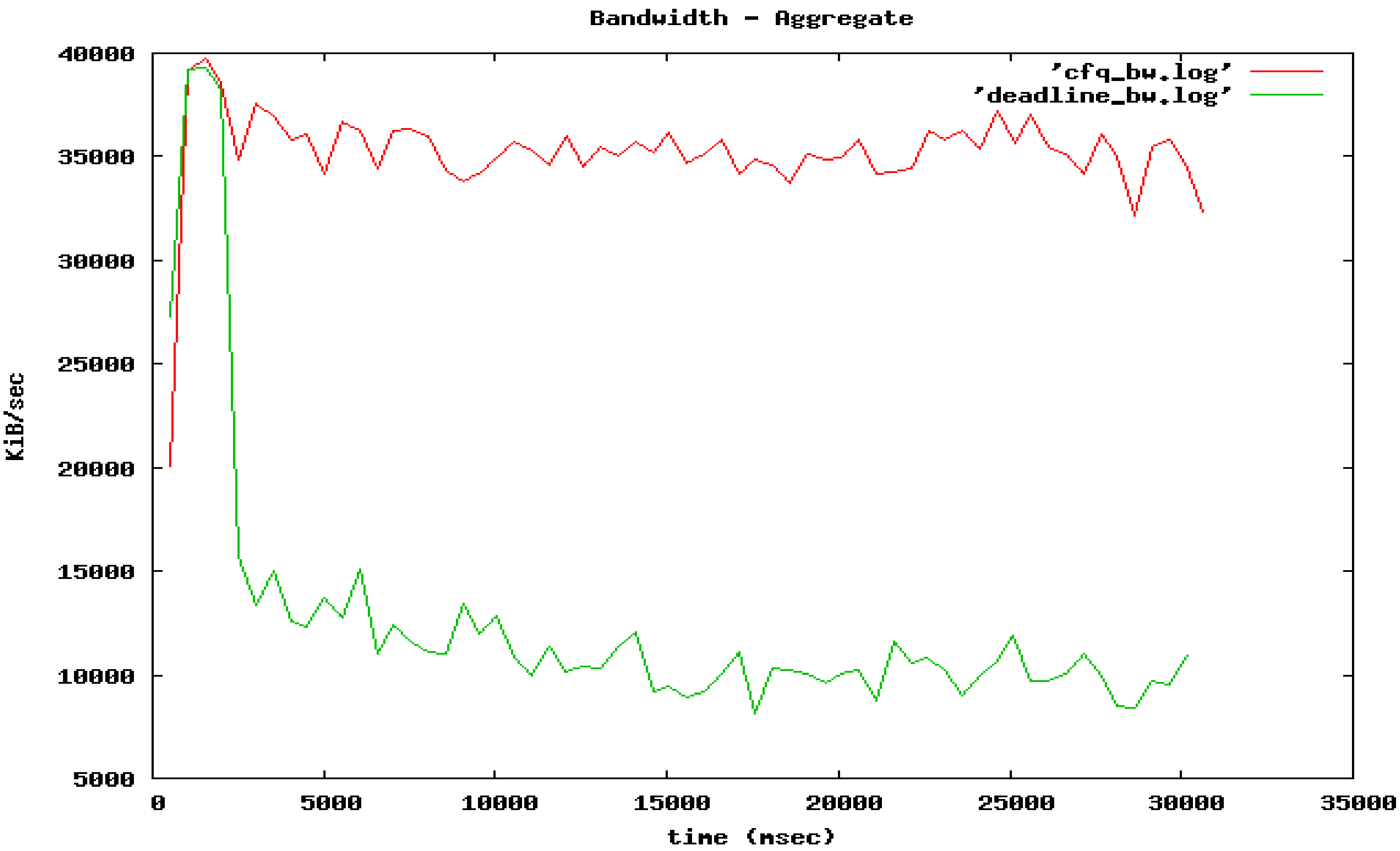
- Simple to understand
  - Each process gets priority access to the disk for a given period of time
- Fair
  - Occasional/sync issuer gets as much time as queue flooder
  - Defined latency
- Synchronous slices
  - Time bounded only (100 msec at prio 4)
  - May idle
- Asynchronous slices
  - Time bounded (40 msec at prio 4)
  - Request bounded (2 at prio 4)
  - May not idle

Time slices
Idling

- Not a work conserving scheduler
  - May decide to idle drive, even with work pending
- Why?
  - Expect close request
  - Seek time reduction
  - Several processes
- When not to idle
  - Process IO pattern is seeky
  - Process “waits” for too long between requests
  - Command queuing
  - Slice left is less than expected wait and service time

Streamed readers

Example, CFQ vs DEADLINE



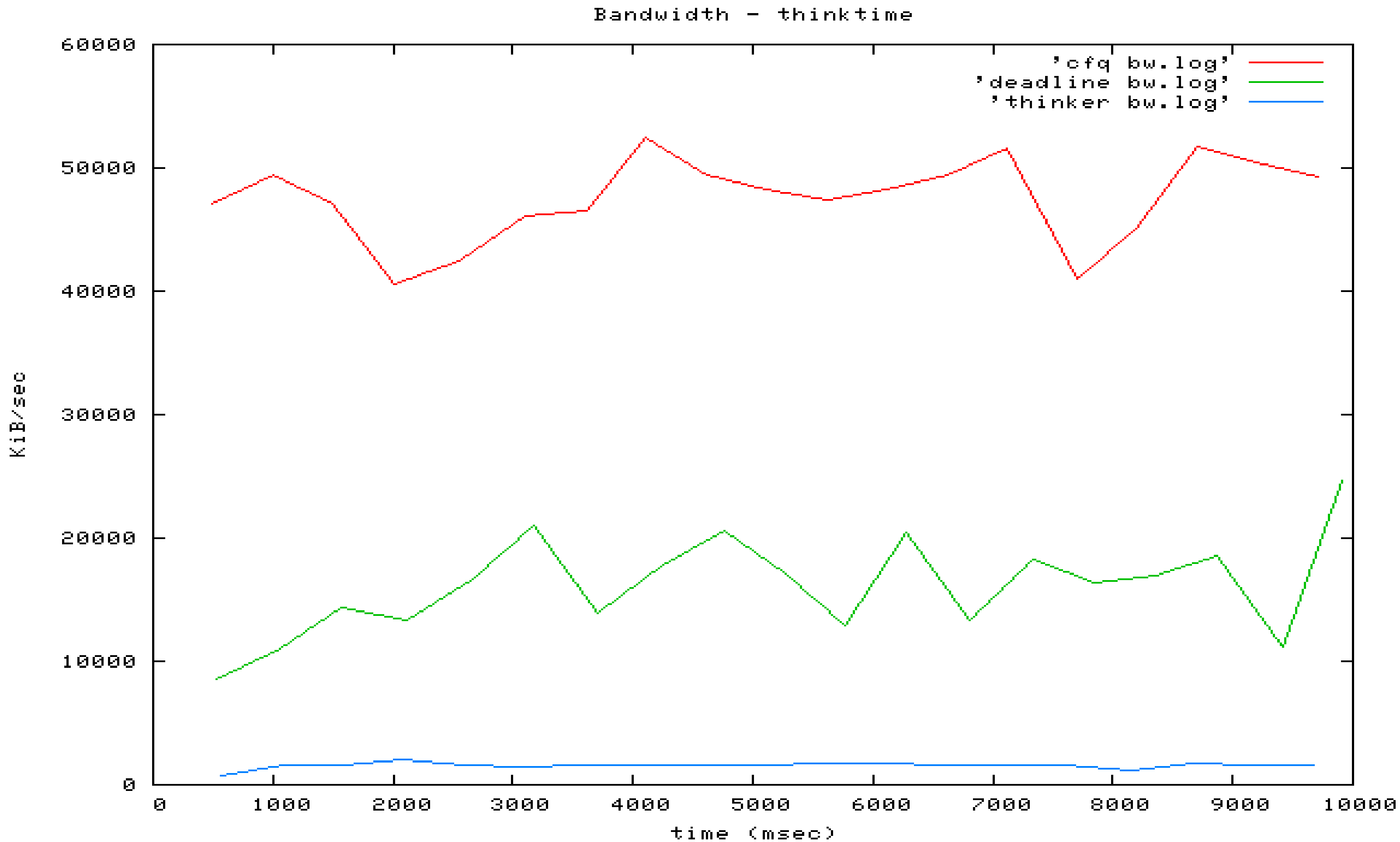
Dependent reads

How many reads to open a 300 byte file?

- Monitor IO activity with blktrace
- exec: `vi /path/to/some/file`
- Total of 5 reads (28KiB)
  - Meta data + file data
  - Takes 66 msecs unloaded
- Undisturbed
  - Imagine a delay between each operation

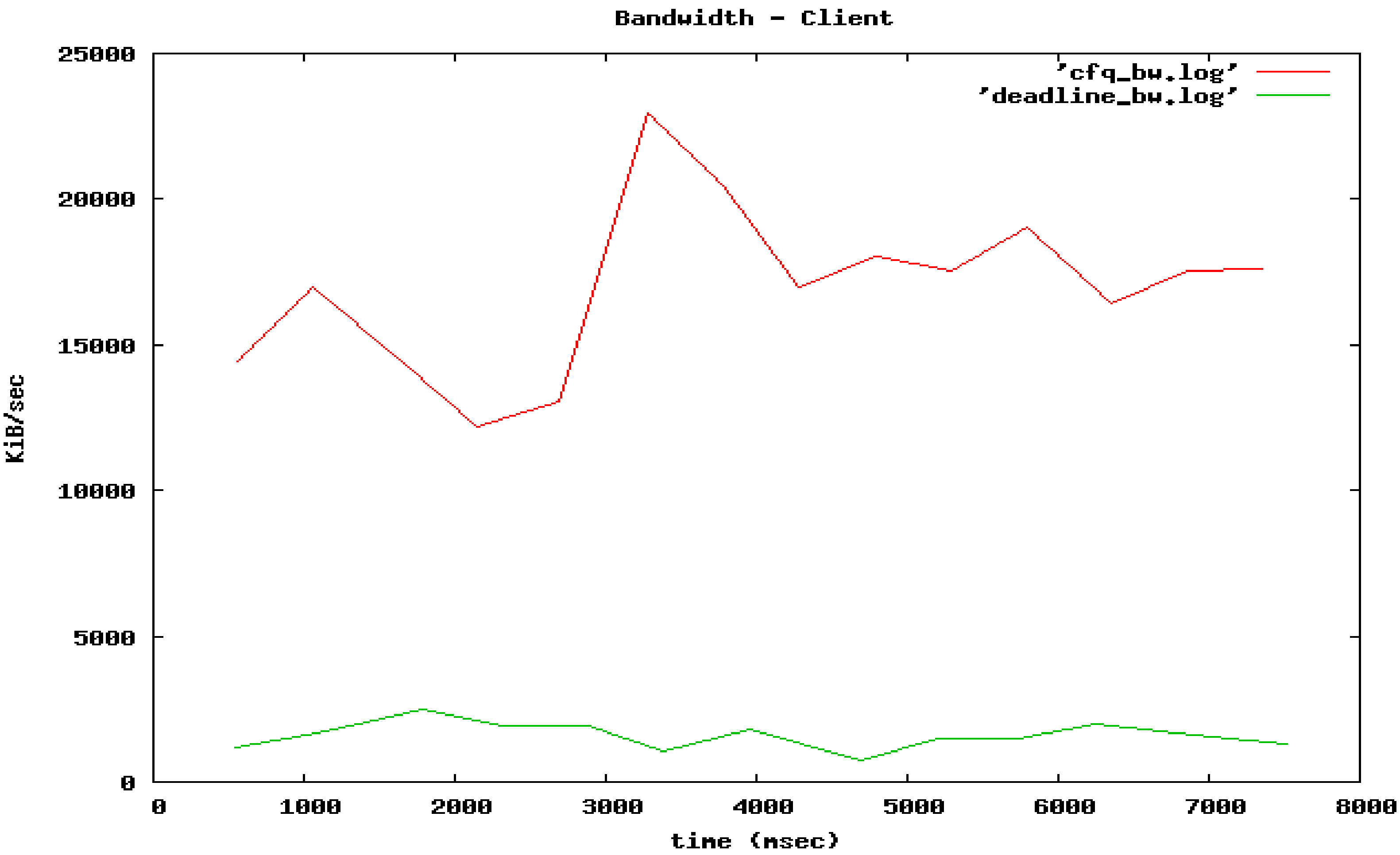
Dependent reads

Example, CFQ/DEADLINE vs thinker



Reader vs writer

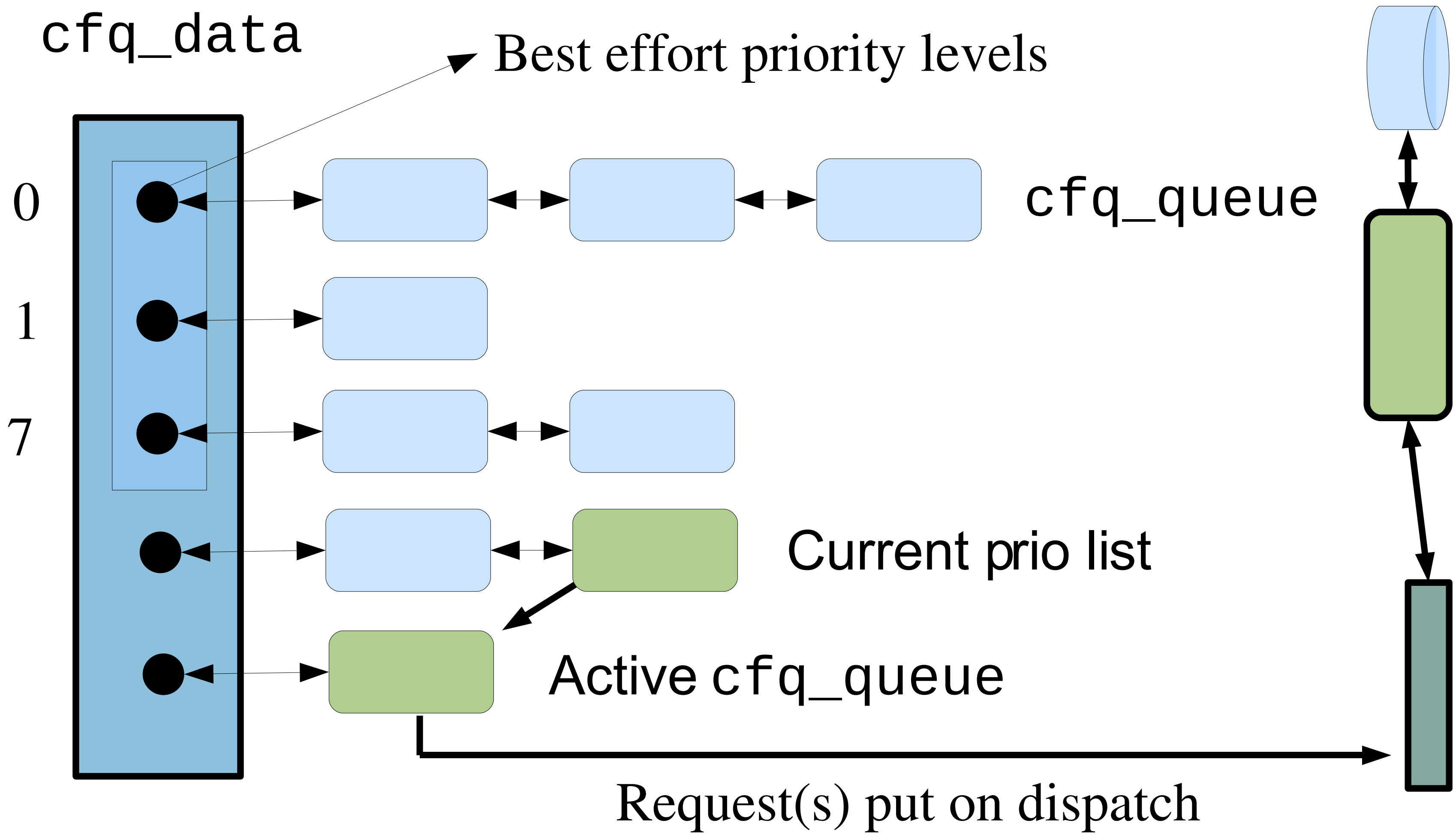
Example, CFQ vs DEADLINE



Queue distribution

- Per-process queues
  - Directly issued IO
  - Synchronous requests
    - Often dependent, process needs request completed before being able to proceed
    - reads, direct writes
  - Latency important
- Asynchronous queues
  - Writes
  - (mainly) indirectly issued IO
    - pdflush
    - kswapd
  - One per priority level
  - Latency less important

CFQ per-queue data
Continued



Queue selection algorithm

- If active queue
  - If slice expired
    - new queue
  - If requests pending
    - dispatch
  - If queue is sync
    - If arm idle timer
      - return and wait
  - Expire slice
- Set new active queue

Queue request dispatch

- If FIFO contains an expired element
  - Sort that request into dispatch list
- Select next request from queue
  - Sort that request into dispatch list
- Repeat until quantum met or queue empty
- Assign slice time to queue
- Expire queue for various criteria
  - Queue is idle class
  - Async queue, and request number met

IO Priority Classes

- 3 default IO scheduling classes
  - Each with 8 sub levels, [0 – 7]
- Idle
  - Only gets access to disk, when nobody else uses it
  - Grace period
  - Currently root only
- Best effort
  - Default class
- Real time
  - Always gets priority access to disk (goes straight to `cur_rr`)
  - Otherwise like best effort (same slice lengths)
  - root only

IO Priorities

- 8 default levels
  - 0 the highest, 7 the lowest. Default is 4.
- Simple extension to time slices
  - Just scale slice with priority
- Can be set explicitly with ionice
  - $ ionice -n <level> -c <class> [-p <pid>] [command]
  - Inherited across forks
- Otherwise, follows cpu nice
  - Best effort class
  - nice -20...-16: ionice 0
  - nice 0...4 ionice 4
  - nice 15...19: ionice 7

IO Priorities

Priority selection

- If real time queues exist
  - Select next RR queue for service
- If best effort queues exist
  - 0
  - 0, 1
  - 0, 1, 2 ...
- If idle queues exist
  - If we are past the idle grace period
    - Select next idle queue for service
  - Arm idle grace timer

CFQ tunables

- `$ ls /sys/block/sda/queue/iosched/`
  - **back_seek_max, back_seek_penalty**
    - Modification of one-way scan
  - **fifo_expire_async, fifo_expire_sync**
    - Inter-queue fairness
  - **quantum**
    - Max dispatch number
  - **slice_sync, { slice_async, slice_async_rq }**
    - Controls slice management
  - **slice_idle**
    - Controls maximum idle time at the end of slice

Performance Results

Test setup

- 7200 RPM SATA drive
- ata_piix controller
- NCQ not used (controller not capable)
- Pentium D 3.0GHz, 2GB RAM
  - Not terribly important
- XFS file system

- fio tool used for benchmarks
  - Flexible IO tester

Benchmark, competing readers
fio job file

- 8 simultaneous readers
  - 128 MiB files
- 4 KiB block size
- Write bandwidth log
  - 500 msec window average

```
[global]
bs=4k
buffered=1
rw=read
ioengine=sync
iodepth=1
size=128m
write_bw_log

[files]
numjobs=8
```

Benchmark, competing readers

Results

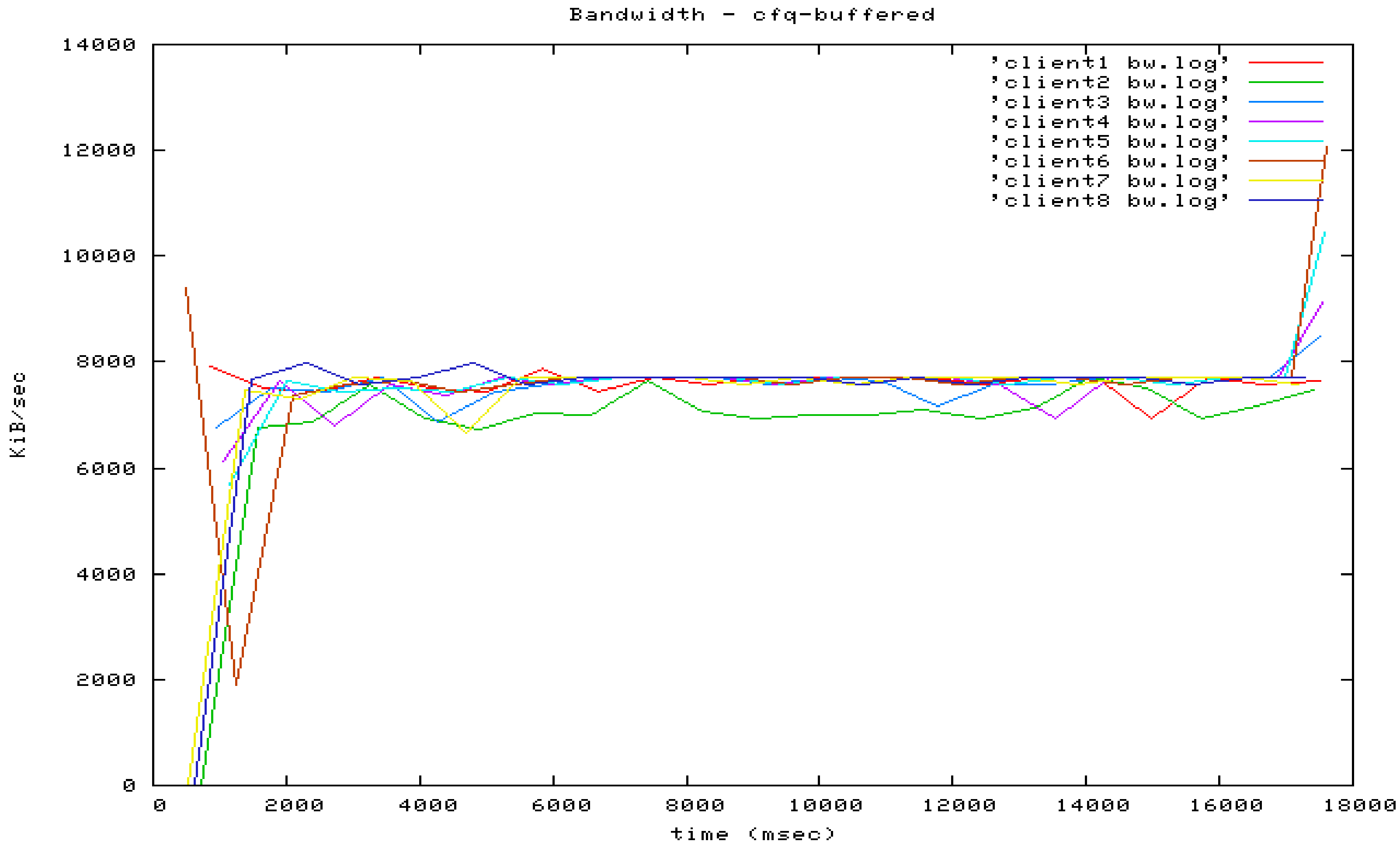
Single thread: ~62MiB/sec

	Min bw	Max bw	Aggr bw	% of max
CFQ	7535 K/s	7727 K/s	60295 K/s	94.9%
AS	7240 K/s	9451 K/s	57921 K/s	91.2%
DEADLINE	2462 K/s	2488 K/s	19700 K/s	31.0%

	Runtime	Max latency	Avg lat	Deviation
CFQ	17.8 secs	746 msec	0.5 msec	19 msec
AS	18.5 secs	890 msec	0.5 msec	17 msec
DEADLINE	54.5 secs	240 msec	1.7 msec	13 msec

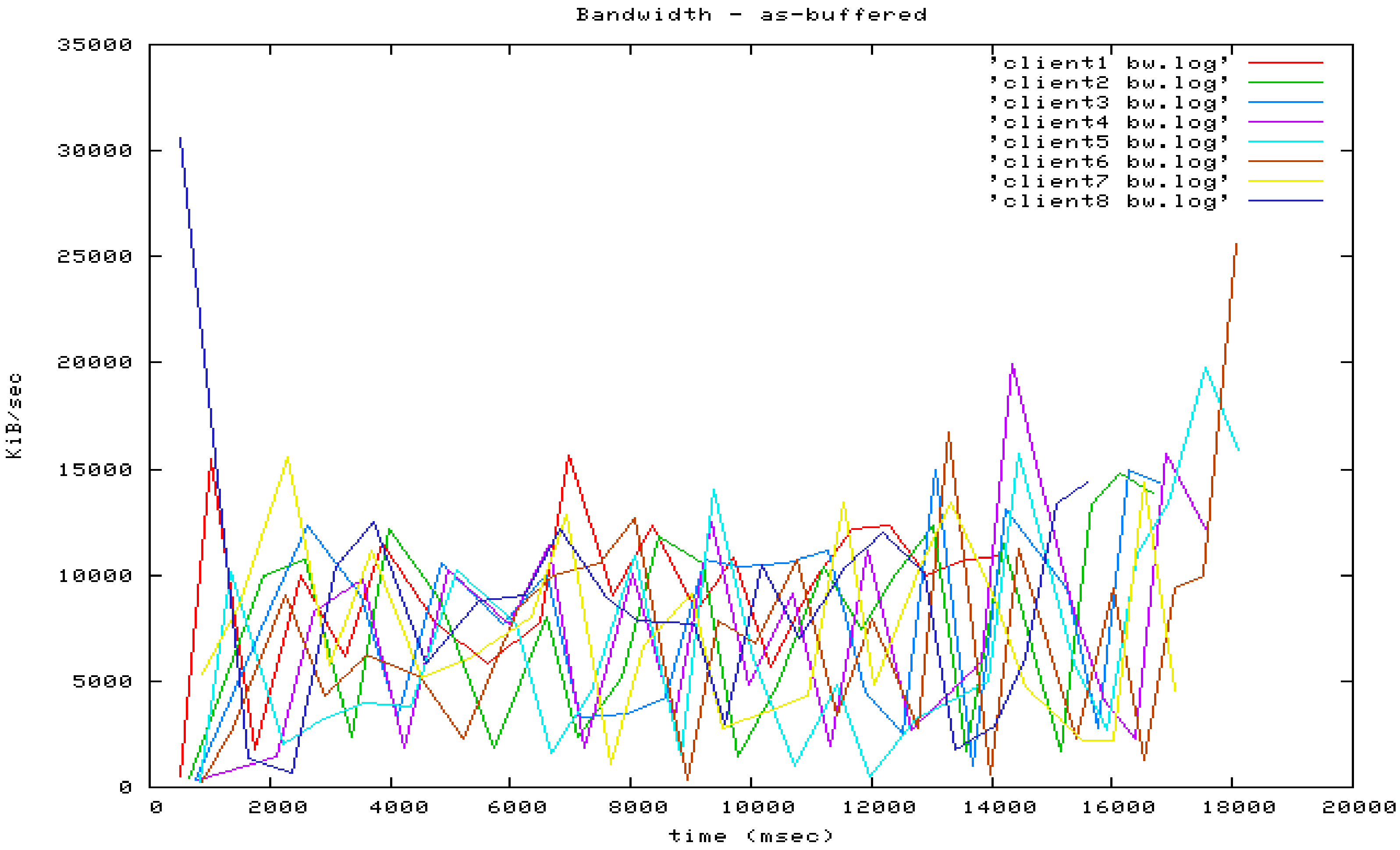
Benchmark, competing readers

Graphed bandwidth, CFQ



Benchmark, competing readers

Graphed bandwidth, AS



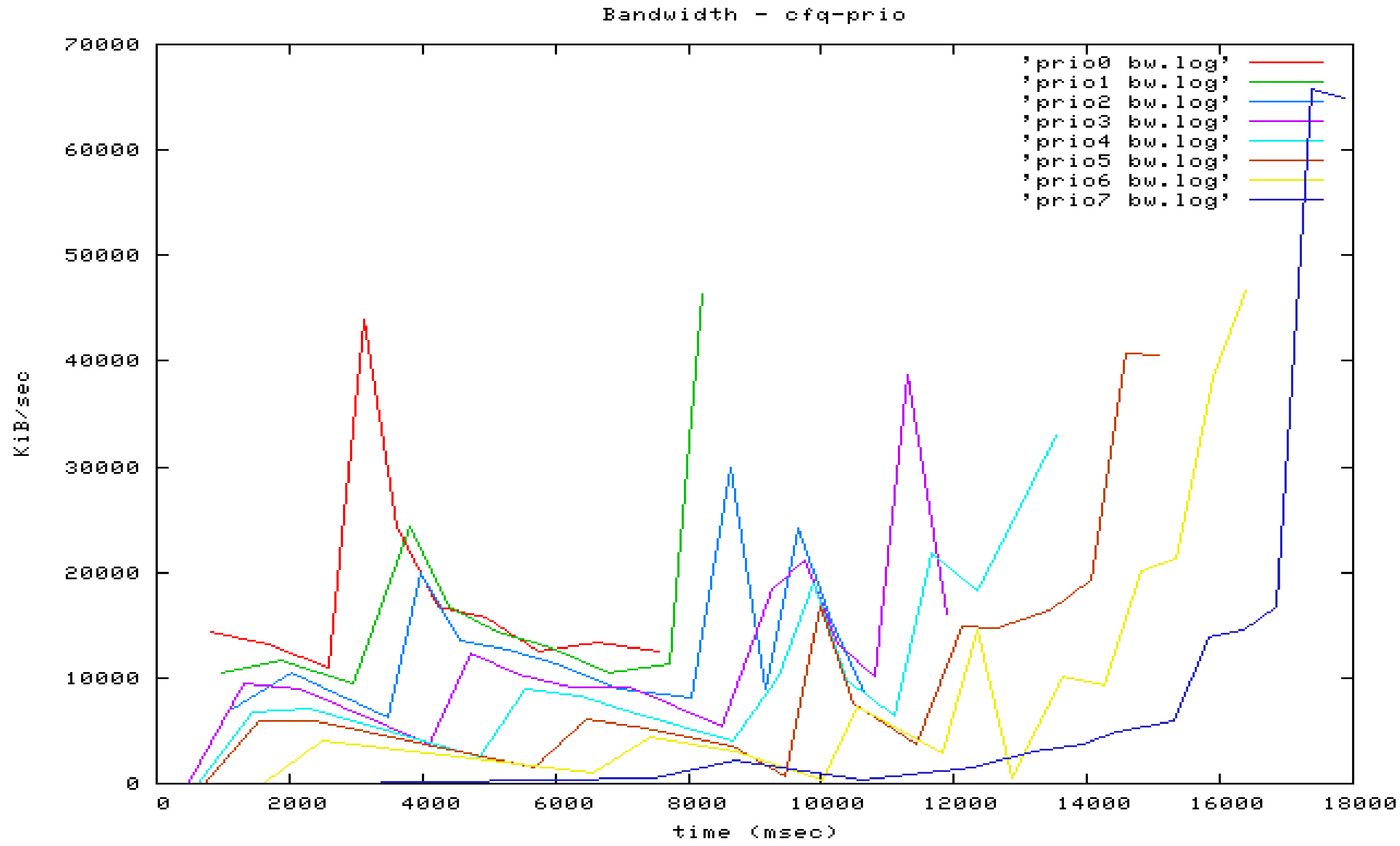
Benchmark, competing readers

CFQ IO priorities

- fio job file identical to previous
  - Files 1...8 uses priority 0...7
  - Best effort

Benchmark, competing readers

Graphed bandwidth, CFQ priorities



Resources

- Kernel files
  - `block/cfq-iosched.c`
  - `block/elevator.c, include/linux/elevator.h`
  - `block/ll_rw_blk.c, include/linux/blkdev.h`
- fio
  - git clone git://git.kernel.dk/data/git/fio.git
- blktrace
  - git clone git://git.kernel.dk/data/git/blktrace.git
  - Kernel parts merged since 2.6.17

Questions?

ORACLE IS THE INFORMATION COMPANY

Thanks!