Improving Disk I/O Performance on Linux

Carl Henrik Lunde, Håvard Espeland, Håkon Kvale Stensland, Andreas Petlund, Pål Halvorsen

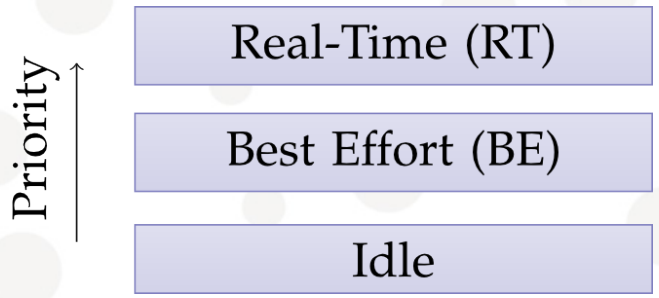
UNIVERSITETET I OSLO

Completely Fair Queuing

Default scheduler on Linux

Ensures complete fairness among I/O-requests in the same class by assigning time slices to each process

Provides some level of QoS by assigning processes to different classes.



Completely Fair Queuing

Gives relatively high throughput by waiting for close requests from the same process (spatial locality)

Latency is kept proportional to system load by scheduling each process periodically

Work-conserving scheduler

- If the active process is considered idle, CFQ will prematurely end the time slice, store the residual time, and move on the the next process

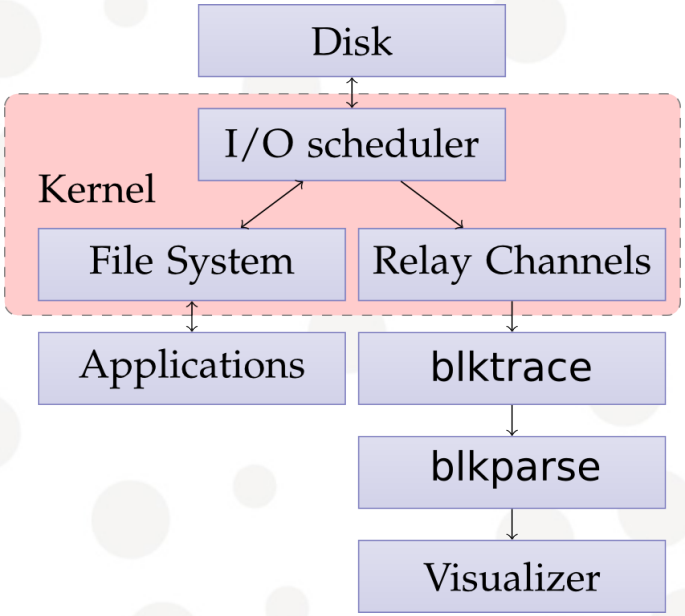
What happens at the different layers

```
Blockdev CPU Sequence Time        PID  Event Sector + blocks
  8 ,48  3        0   2.004027488 8593 m   N enter vfs_read
  8 ,48  3      657   2.004047581 8593 A   R 831107423 + 256 <- (8 ,49) 831107360
  8 ,48  3      658   2.004047704 8593 Q   R 831107423 + 256 [ iosched - bench ]
  8 ,48  3      659   2.004048864 8593 G   R 831107423 + 256 [ iosched - bench ]
  8 ,48  3      660   2.004049761 8593 I   R 831107423 + 256 [ iosched - bench ]
  8 ,48  3      661   2.004155593 8593 A   R 831107679 + 256 <- (8 ,49) 831107616
  8 ,48  3      662   2.004155696 8593 Q   R 831107679 + 256 [ iosched - bench ]
  8 ,48  3      663   2.004156385 8593 M   R 831107679 + 256 [ iosched - bench ]
  8 ,48  3      664   2.004157215 8593 U   N [ iosched - bench ] 7
  8 ,48  2      599   2.010858781 8598 C   R 33655111 + 256 [0]
  8 ,48  2      600   2.010983458 8598 A   R 33655623 + 256 <- (8 ,49) 33655560
  8 ,48  2      601   2.010983593 8598 Q   R 33655623 + 256 [ iosched - bench ]
  8 ,48  2      602   2.010984420 8598 G   R 33655623 + 256 [ iosched - bench ]
  8 ,48  2      603   2.010985094 8598 I   R 33655623 + 256 [ iosched - bench ]
  8 ,48  2      604   2.010988047 8598 D   R 33655623 + 256 [ iosched - bench ]
  8 ,48  2      605   2.010990666 8598 U   N [ iosched - bench ] 7
  8 ,48  1      530   2.012061299 8598 C   R 33655367 + 256 [0]
  8 ,48  1      531   2.012182151 8598 A   R 33655879 + 256 <- (8 ,49) 33655816
  8 ,48  1      532   2.012182705 8598 Q   R 33655879 + 256 [ iosched - bench ]
  8 ,48  1      533   2.012183893 8598 G   R 33655879 + 256 [ iosched - bench ]
  8 ,48  1      534   2.012184605 8598 I   R 33655879 + 256 [ iosched - bench ]
  8 ,48  1      535   2.012187585 8598 D   R 33655879 + 256 [ iosched - bench ]
  8 ,48  1      536   2.012190294 8598 U   N [ iosched - bench ] 7
  8 ,48  3      665   2.012834865 8598 C   R 33655623 + 256 [0]
  (...)
```

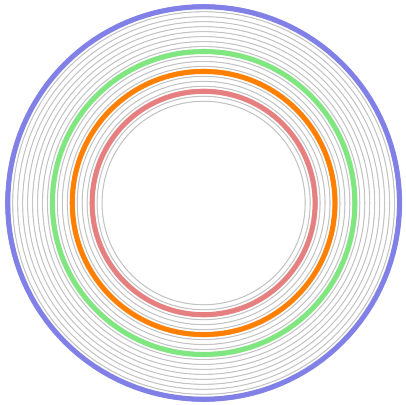
Performance of I/O-schedulers

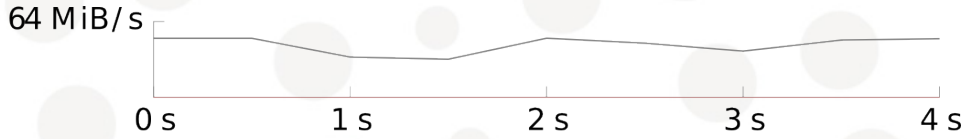
To evaluate I/O-schedulers, we wrote a tool to visualize the behaviour

I/O requests are captured using *blktrace*, and we can analyse what happens at each level in great detail

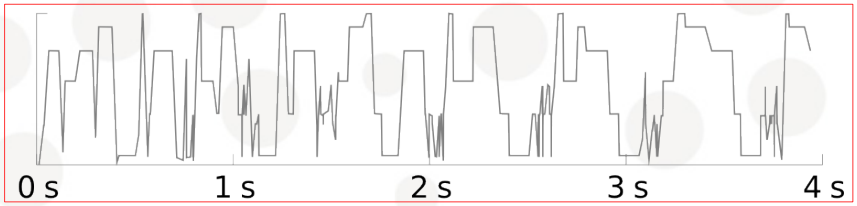


CFQ Behaviour Visualized



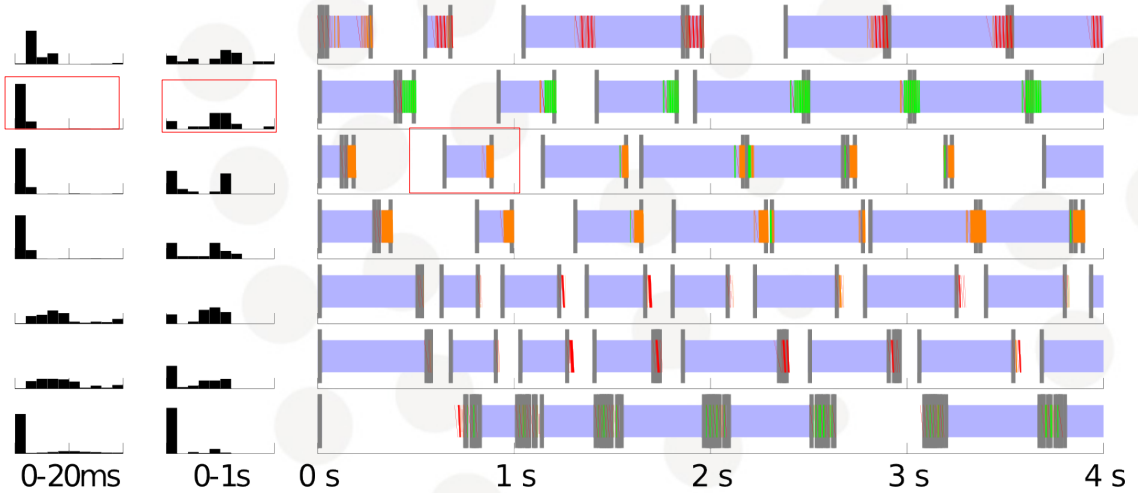


Aggregate bandwidth



Estimated disk head position

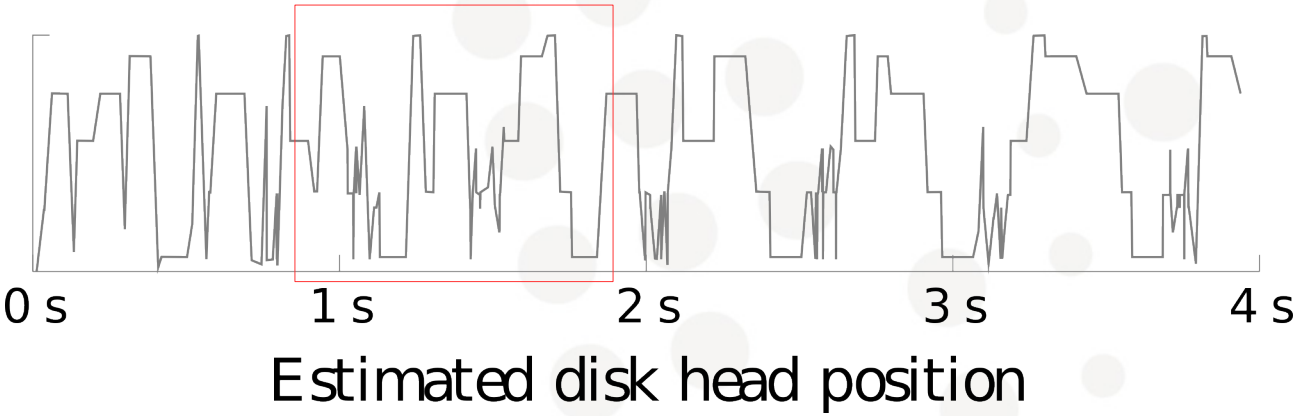
S_3 14.3/16 MiB/s 8192 KiB
S_2 15.6/16 MiB/s 8192 KiB
S_1 8.0/8 MiB/s 4096 KiB
S_0 8.1/8 MiB/s 4096 KiB
R_1 39.0/40 KiB/s 4 KiB
R_0 40.0/40 KiB/s 4 KiB
MF_0 388.0 KiB/s 446 files



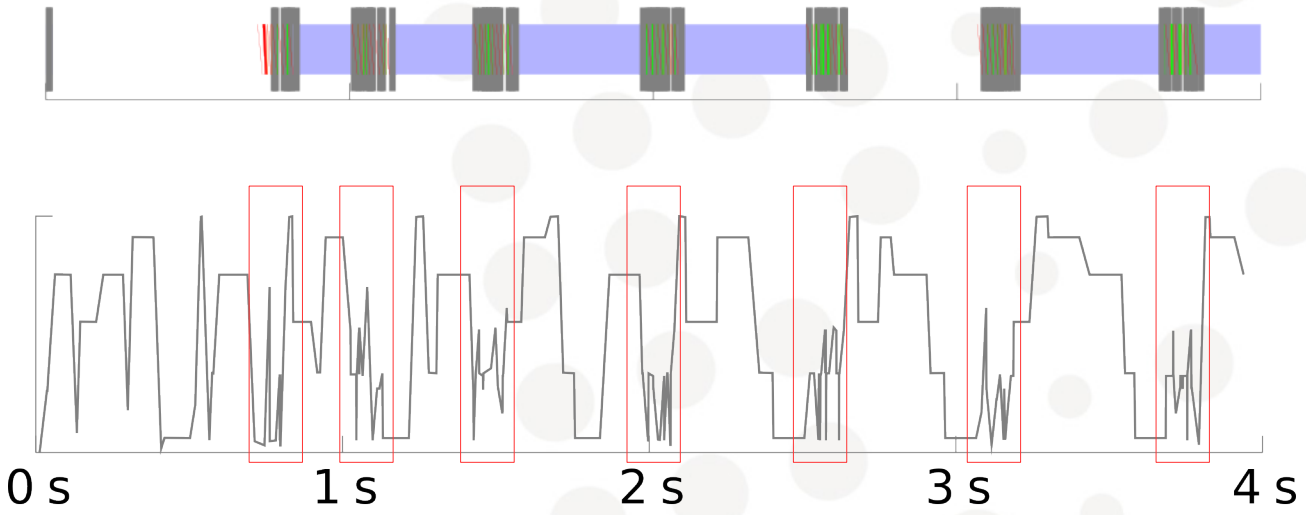
D2C read() Reads and I/Os

Suggested improvements

Add a top level elevator for better performance



Suggested improvements



Improve the seeking application

Queuedepth=1 I/O scheduler has no choices!

CFQ

Current best-effort proportional share “QoS” is not sufficient for applications with bandwidth requirements

- How many MiB/s is 200 ms/s?

Real-time priority can starve other processes

- Filling buffers can leave other processes with a missed deadline

Hard to do work conservation in RT class

Adding QoS to CFQ

Introduce a bandwidth class that can coexist with other CFQ classes

Reserves bandwidth in terms of bytes per second in contrast to proportional disk time

Do not change behaviour of existing classes

CFQ Bandwidth Class Proof-of-Concept

Implemented as Token Bucket to allow bursty behaviour, like VBR video streaming

Avoid seeks by introducing a top-level elevator for BW readers which improves global throughput

Work conservation enables BW readers to utilize the disk in excess of the reserved bandwidth if no best effort-readers are pending

Unused reserved bandwidth are distributed among best effort readers

Evaluation of BW class

Benchmarks use videos in a streaming scenario

- HD video at 3 MiB/s (25 Mbit/s)
- SD video at 1 MiB/s

In some tests additional greedy readers consumes data at a rate faster than the disk can provide

Deadline Misses

Find the maximum number of reserved readers that we can add while still maintaining deadlines

Evaluate effect of simultaneous best-effort readers

“Background noise” readers

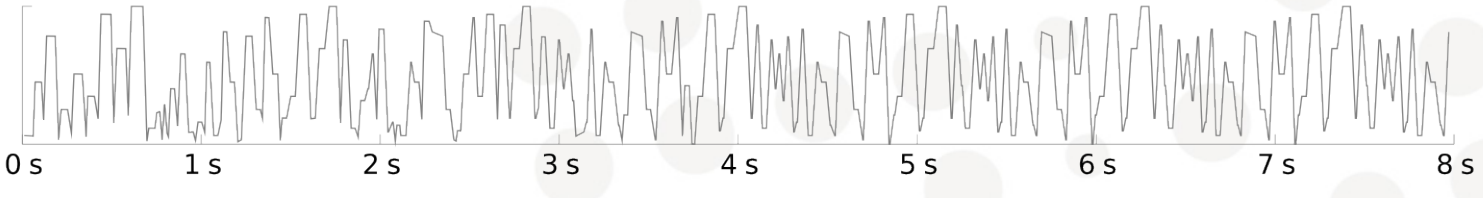
- 10 HD stream readers in Bandwidth class
- 3 random BE, reading 4 KB 20 times per second
- 5 sequential BE, reading 4 MiB/s

Deadline Misses

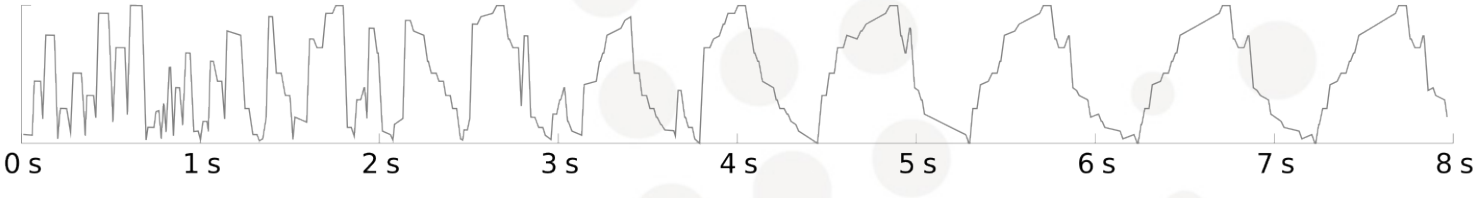
Streams		CFQ			AS	Deadline	noop
1 MiB/s	3 MiB/s	BW SSF	BW C-SCAN	RT			
16	10	0	0	0	479	345	281
17	10	0	0	0	482	330	398
18	10	0	0	0	492	300	783
19	10	0	0	0	512	276	1061
20	10	0	0	179	505	288	1123
21	10	0	0	186	531	381	1075
22	10	0	0	276	517	947	1061
23	10	0	0	N/A	549	1233	1064
24	10	0	0	N/A	553	1276	1054
25	10	182	188	N/A	536	1279	1033

Deadline misses with 1 s deadline / round

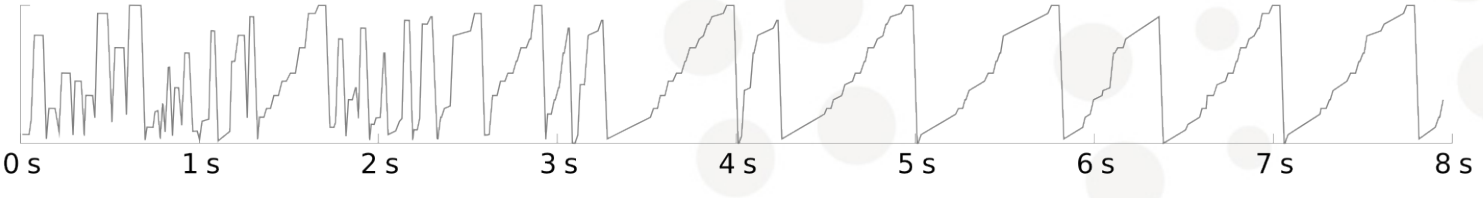
Disk head movement



CFQ RT (no elevator)



CFQ BW SSF



CFQ BW C-SCAN

Isolation

Best-effort readers should not affect reserved readers

Greedy reserved readers should not get more bandwidth than reserved if it would affect any best effort reader

Work Conservation of BW class

	Reserved	Requested	Result	Latency
Greedy	32 MiB/s	∞	63.5 MiB/s	138 ms max (31 ms avg)
Video A	BE4	8 MiB/s	8.0 MiB/s	676 ms max (48 ms avg)
Video B	BE4	8 MiB/s	8.0 MiB/s	

Two Best Effort streams, requesting 8 MiB/s each

One BW class stream with 32 MiB/s requested

The reserved reader requests as much as possible, and receives more than the reservation allows

Optimizing Multi-file Performance

The I/O-scheduler can not improve the multi-file reader.

- Only a limited number of requests to choose from before deadline expires

How fast would it be without disk seeking?

Simulation

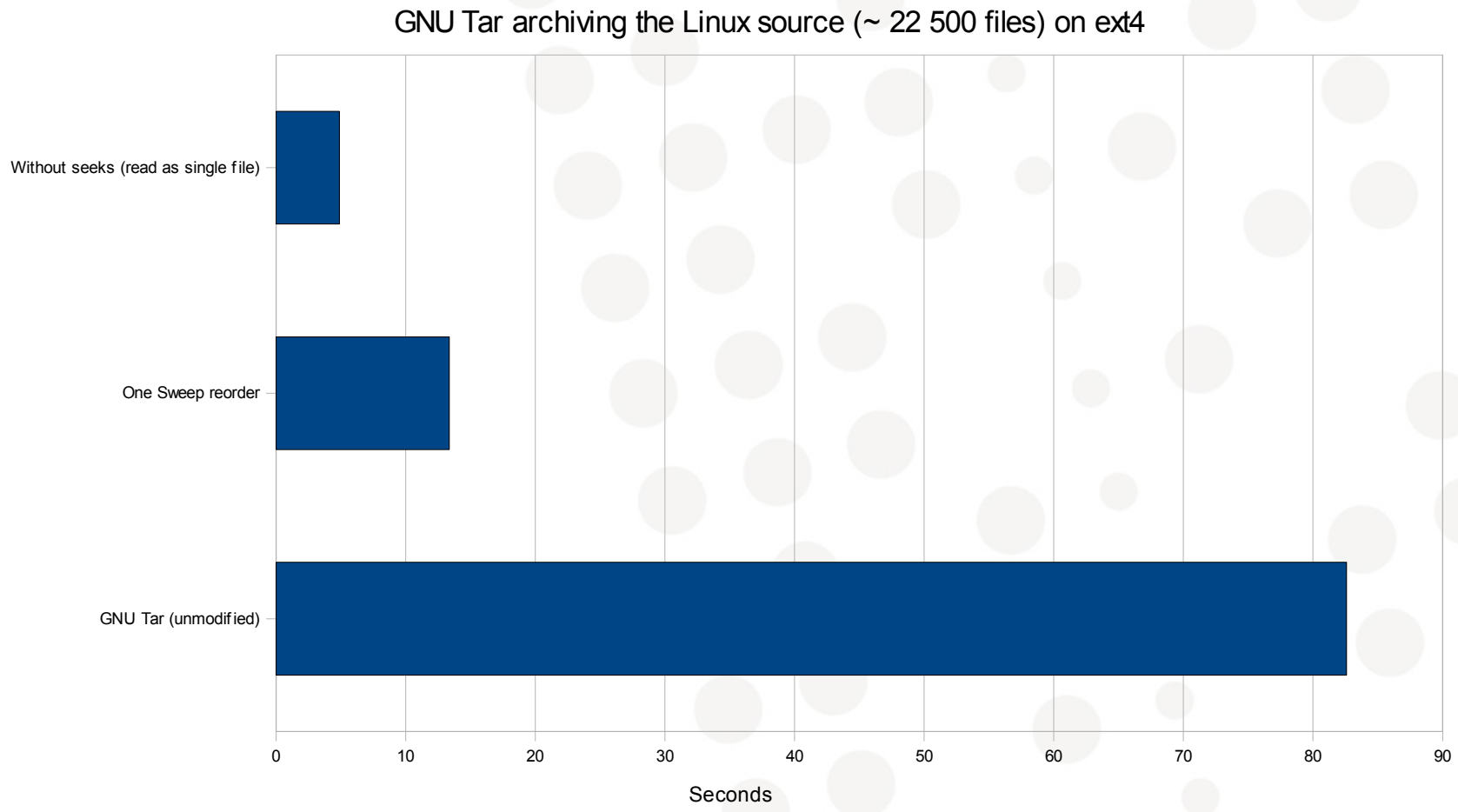
Created an utility to reorder traces of disk activity

Sort requests by block number to minimize seeks

Dispatch requests without delay (ignore CPU-time)

Replay trace on real hardware

Simulation of tar



Result of simulation

Potential improvement in applications that accesses many files on rotational disks

For example: cp, zip, rsync, tar, scp, rm, find, dpkg (database), Tracker, File Manager and many others.

Proposed solution: Order I/O requests relative to physical location on disk before they hit the kernel scheduler

Userspace I/O Scheduling

We know the relative location of meta data by using inode number (at least on ext{3,4})

inode(a) < inode(b) ⇒ block(inode(a)) <= block(inode(b))

On ext4, any user can ask for position of file data

FIEMAP ioctl

Use this knowledge to avoid seeks

Implementation

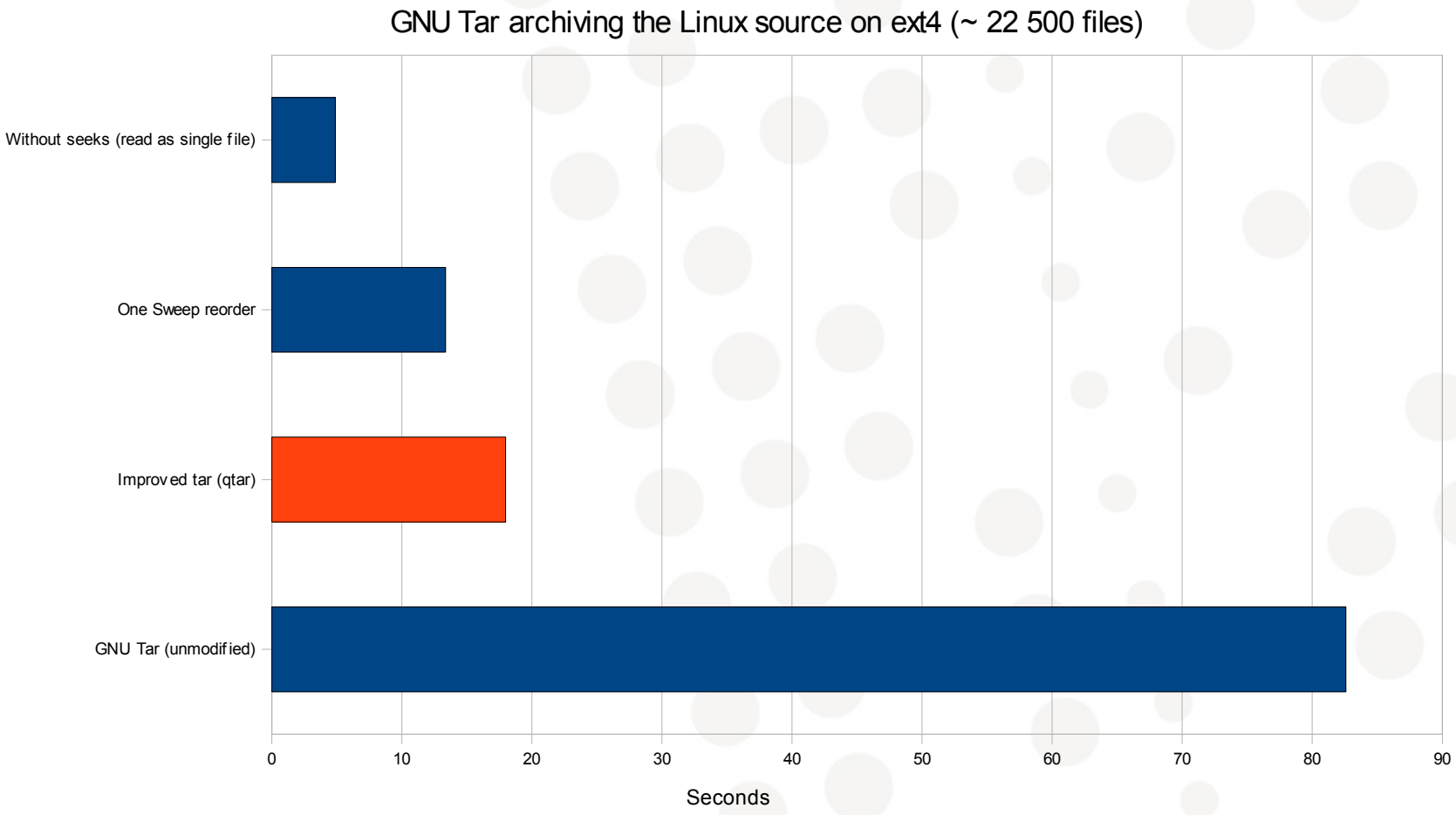
First pass:

- Read all meta data ordered by inode number (inodes and directories)
  - New directories are discovered during traversal, so we cannot sort all meta data in advance
  - Use a C-SCAN elevator for meta-data

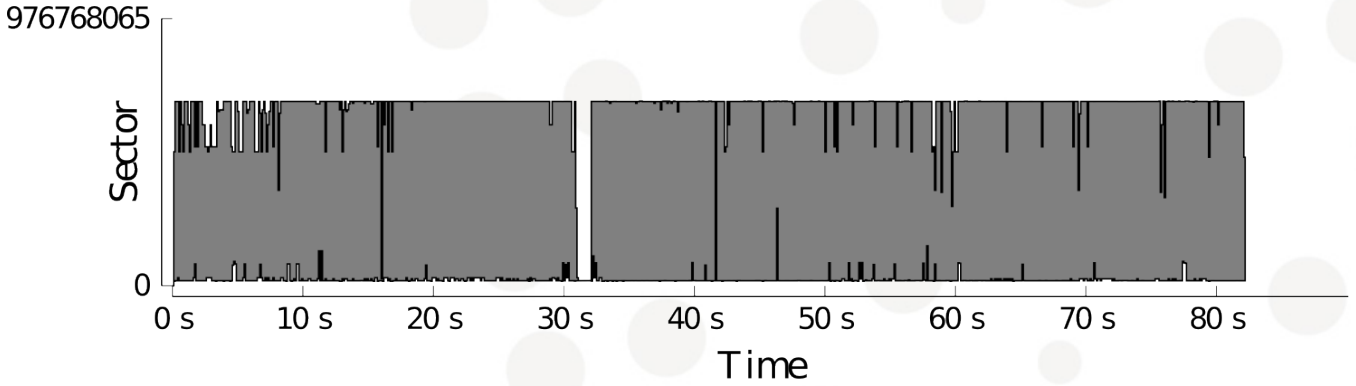
Second pass:

- Read file data ordered by file data position

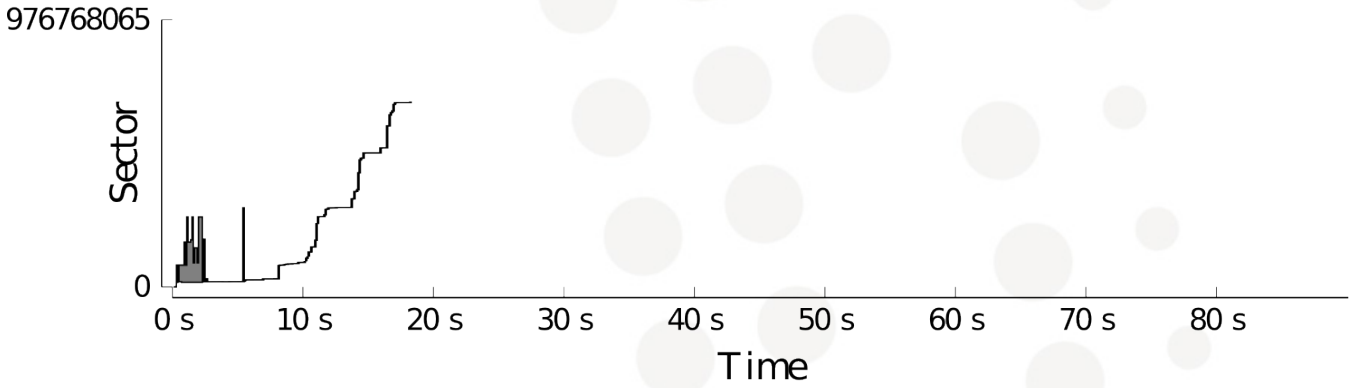
Improved tar results



Visualization of seek footprint (ext4)

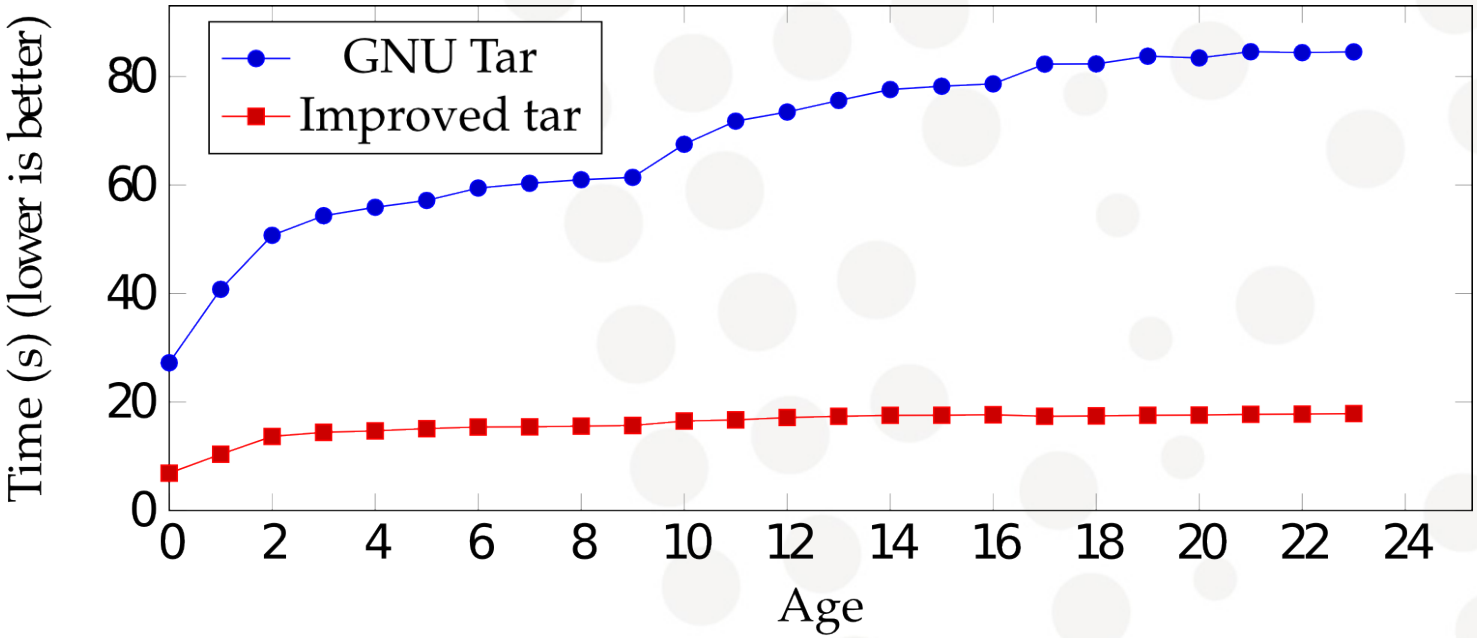


(a) GNU Tar



(b) Improved tar

Filesystem Aging



Archiving the Linux kernel (22 500 files) on ext4

Video

Video of GNU Tar vs. modified tar (qtar)

Aged ext3 file system

Around 5000 files

Some read errors, the disk is dying...

Notice the two passes of the qtar algorithm

Conclusion

Visualizing blktraces can be very useful

CFQ benefits from a top-level elevator

QoS should be available for programs with bandwidth requirements

- A simple bandwidth class added to CFQ works fairly well

The kernel I/O scheduler can only optimize requests that are pending. Ordering requests in user space mitigates this.

Questions?