Walking Through Walls

PostgreSQL ♥ FreeBSD

tmunro@freebsd.org
tmunro@postgresql.org
thomas.munro@enterprisedb.com

About me

- New to FreeBSD hacking
  Mentors: mjg, allanjude

- ~20 years work on proprietary C, C++, … applications on lots of different kinds of Unix

- Past ~4 years working on PostgreSQL at EnterpriseDB

- Past ~3 years dabbling in FreeBSD, beginning with the gateway drug of ZFS home storage boxes, now my main development and server environment

- Personal goal: make FreeBSD and PostgreSQL the best relational database stack

Berkeley

- INGRES: Developed at UC Berkeley, 197x-1985
  - Relational database ideas inspired by IBM's System/R (though using QUEL instead of SQL), developed on PDPs just as Unix arrived at Berkeley
  - First software released entirely under BSD licence (CSRG distribution still needed AT&T licence for parts)
- POSTGRES: Developed at UC Berkeley, 1986-1994
  - Entirely new system (but still using INGRES's QUEL query language)
  - Developed on SunOS (derived from 4.3BSD) and Dynix (derived from 4.2BSD, added SMP support for Sequent computers) and (probably) various other flavours of BSD
- PostgreSQL: Modern open source project, 1996-
  - We current claim to support Linux, {Open,Net,Free}BSD, macOS, AIX, HP/UX, Solaris, Windows; in the past we supported IRIX, Tru64, UnixWare, BSD/OS, BeOS, QNX, SunOS, SCO OpenServer

Michael Stonebraker

Latter day PostgreSQL hackers on a pilgrimage to Berkeley

How operating systems look to database hackers

- APIs, man pages, standards chiselled in stone
- Administration tools, tunables, monitoring tools
- Things get more interesting if you can actually influence the operating system!

Database hacker dilemmas

- Use thread/process per sessions and rely on kernel scheduling, or do own work scheduling over N threads (tuned for CPU topology)?
- Use OS page cache (as well as own cache!), or do direct IO? If buffered, what amount of user space IO scheduling (read ahead, write behind, write coalescing etc)?
- Use OS-supplied collation rules for text?
- Use our own userspace locking and IPC primitives?
- … more questions like this
- General theme: use OS facilities or do it ourselves?

- DB2 and Oracle switched to direct IO around the time of the following messages from Linux leadership (indicating that this was highly contentious)

- PostgreSQL is approximately the last RDBMS still using buffered IO (though others can as an option)

In summary, **O_DIRECT** is a potentially powerful tool that should be used with caution. It is recommended that applications treat use of **O_DIRECT** as a performance option which is disabled by default.

> "The thing that has always disturbed me about O_DIRECT is that the whole interface is just stupid, and was probably designed by a deranged monkey on some serious mind-controlling substances."—Linus

Date Wed, 10 Jan 2007 19:05:30 -0800 (PST)
From Linus Torvalds <>
Subject Re: O_DIRECT question

On Thu, 11 Jan 2007, Aubrey wrote:
>
> Now, my question is, is there a existing way to mount a filesystem
> with O_DIRECT flag? so that I don't need to change anything in my
> system. If there is no option so far, What is the right way to achieve
> my purpose?

The right way to do it is to just not use O_DIRECT.

The whole notion of "direct IO" is totally braindamaged. Just say no.

This is your brain: O
This is your brain on O_DIRECT: .

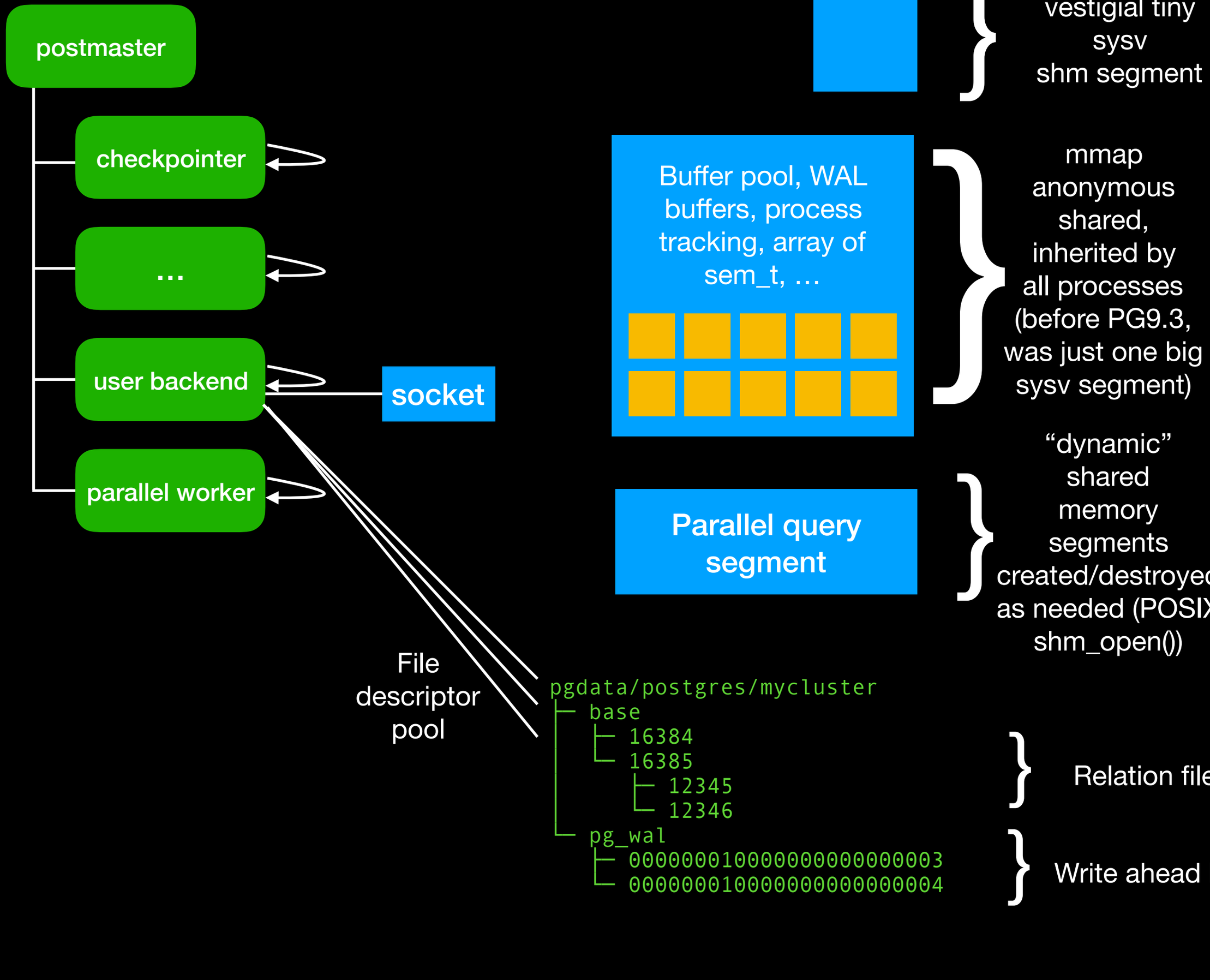
Any questions?

I should have fought back harder. There really is no valid reason for EVER using O_DIRECT. You need a buffer whatever IO you do, and it might as well be the page cache. There are better ways to control the page cache than play games and think that a page cache isn't necessary.

So don't use O_DIRECT. Use things like madvise() and posix_fadvise() instead.

Linus

How PostgreSQL looks to operating systems


postmaster
checkpointer
...
user backend
parallel worker
socket
File descriptor pool
vestigial tiny sysv shm segment
Buffer pool, WAL buffers, process tracking, array of sem_t, …
mmap anonymous shared, inherited by all processes (before PG9.3, was just one big sysv segment)
Parallel query segment
"dynamic" shared memory segments created/destroyed as needed (POSIX shm_open())
pgdata/postgres/mycluster
base
16384
16385
12345
12346
pg_wal
000000010000000000000003
000000010000000000000004
Relation files
Write ahead log

Processes

```
13316  └─ postgres -D /data/clusters/main
13441     ├─ postgres: fred salesdb [local] idle
13437     ├─ postgres: fred salesdb [local] idle
13337     ├─ postgres: fred salesdb [local] SELECT
13323     ├─ postgres: logical replication launcher
13322     ├─ postgres: stats collector
13321     ├─ postgres: autovacuum launcher
13320     ├─ postgres: walwriter
13319     ├─ postgres: background writer
13318     └─ postgres: checkpointer
```

"Currently, POSTGRES runs as one process for each active user. This was done as an expedient to get a system operational as quickly as possible. We plan on converting POSTGRES to use lightweight processes available in the operating systems we are using. These include PRESTO for the Sequent Symmetry and threads in Version 4 of Sun/OS."

Stonebraker, Rowe and Herohama, “The Implementation of POSTGRES”, 1989

System calls

Idle backend process:

```
poll({ 9/POLLIN 10/POLLIN 3/POLLIN },3,-1)       = 1 (0x1)
```

Processing a simple read-only query with and without hot cache:

```
recvfrom(9,"B\0\0\0\^[\0P0_1\0\0\0\0\^A\0\0"...,8192,0,NULL,0x0) = 50 (0x32)
sendto(9,"2\0\0\0\^DT\0\0\0!\0\^Aabalance"...,71,0,NULL,0) = 71 (0x47)

recvfrom(9,"B\0\0\0\^[\0P0_1\0\0\0\0\^A\0\0"...,8192,0,NULL,0x0) = 50 (0x32)
pread(14,"\0\0\0\0000D?\^B\0\0\^D\0\f\^A"...,8192,0x1bc000) = 8192 (0x2000)
sendto(9,"2\0\0\0\^DT\0\0\0!\0\^Aabalance"...,71,0,NULL,0) = 71 (0x47)
```

Writing to the WAL when we COMMIT a transaction:

```
pwrite(30,"\M^X\M-P\^D\0\^A\0\0\0\0`\M-l\n"...,16384,0xec6000) = 16384 (0x4000)
fdatasync(0x1e)                                   = 0 (0x0)
```

The checkpointer process writing back dirty data durably:

```
openat(AT_FDCWD,"base/13002/2674",O_RDWR,00)      = 17 (0x11)
pwrite(17,"\0\0\0\0x\M^?D\f\0\0\0\0\M-P\^C"...,8192,0x2c000) = 8192 (0x2000)
pwrite(17,"\0\0\0\0\bOD\f\0\0\0\0\M-@\^C\0"...,8192,0x4e000) = 8192 (0x2000)
pwrite(17,"\0\0\0\08\^\D\f\0\0\0\0\^P\^D \b"...,8192,0x5a000) = 8192 (0x2000)
...
fsync(0x13)                                       = 0 (0x0)
fsync(0xf)                                        = 0 (0x0)
fsync(0xe)                                        = 0 (0x0)
fsync(0xd)                                        = 0 (0x0)
```

…

PostgreSQL/FreeBSD performance and scalability on a 40-core machine*

Konstantin Belousov <kib@FreeBSD.org>

July 16, 2014

Version 2.0. SVN revision 142

Contents

1 Overview **3**
1.1 Measuring contention 4

2 Page Fault Handler **4**
2.1 Fast Path 6
2.2 Phys backing for shared area 7
2.3 Pre-populating the shared area 7
2.4 DragonFlyBSD shared page tables 8

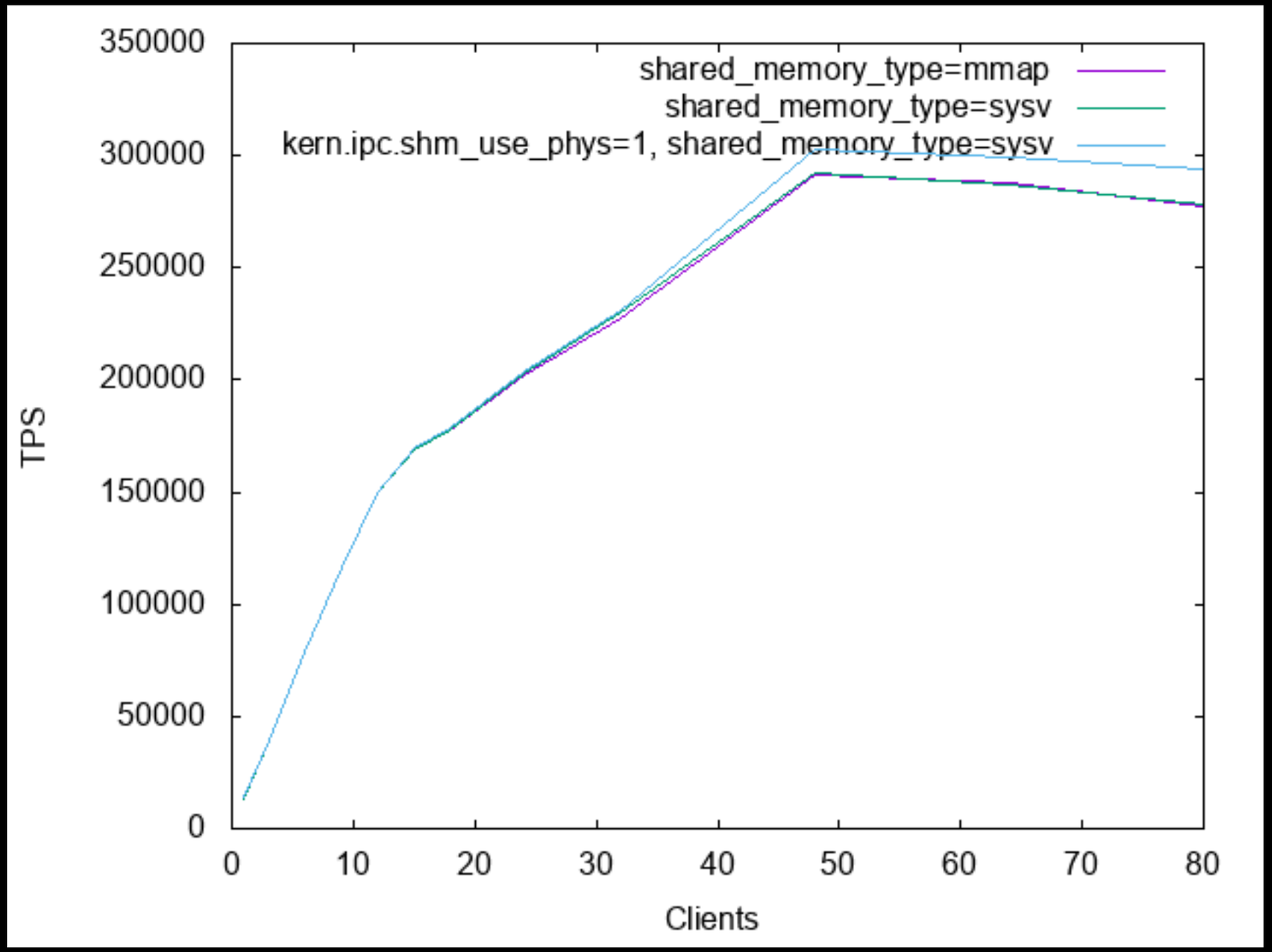
MAP_SHARED | MAP_ANONYMOUS

- In PostgreSQL 9.3 we stopped using a big System V shared memory region and switch to an inherited anonymous shared mmap region

- Performance tanked on large many-cored FreeBSD systems

- The explanation was mostly that kern.ipc.shm_use_phys=1 was being used on large machines (avoiding the creation of pv entries that performed poorly at scale), but we don’t have a similar mode for anonymous memory

- Many improvements were made since then to address the contention problems; is the problem completely fixed?



- We are planning to add a shared_memory_type=sysv option so that we can go back to System V (mainly for AIX), which will allow this option to be used again

- Quick testing seemed to indicate that there is still some speed-up reachable that way on a 40 vCPU m4.x10large system (but I don’t have high confidence in the results, more testing required)

fdatasync()

- Don’t flush file meta-data when flushing data blocks in the WAL, just flush the data. 1 random IO instead of 2?

- I tried to work on this myself… probably a bit too tricky for a starter patch (filesystems are scary), though I had something kinda working…

- I updated my source tree one day and *blam*, the big guns had beaten me to it

Done in FreeBSD 11

setproctitle_fast(3)

```
13316  └─ postgres -D /data/clusters/main
13441     ├─ postgres: fred salesdb [local] idle
13437     ├─ postgres: fred salesdb [local] UPDATE
13337     ├─ postgres: fred salesdb [local] SELECT
13323     ├─ postgres: logical replication launcher
13322     ├─ postgres: stats collector
13321     ├─ postgres: autovacuum launcher
13320     ├─ postgres: walwriter
13319     ├─ postgres: background writer
13318     └─ postgres: checkpointer
```

- PostgreSQL updates the process title 2+ times per query
- Linux and other BSDs: simply write to a buffer in user-space memory

PostgreSQL 9.6 running trivial query:

```
recvfrom(9,"…"...,8192,0,NULL,0x0)
getpid()
__sysctl(0x7fffffffde80,0x4,0x0,0x0,0x801a0f000,0x28)
sendto(8,"\…"...,152,0,NULL,0)
getpid()
__sysctl(0x7fffffffdfd0,0x4,0x0,0x0,0x801a0f000,0x26)
sendto(9,"…"...,63,0,NULL,0)
```

PostgreSQL 12 running trivial query:

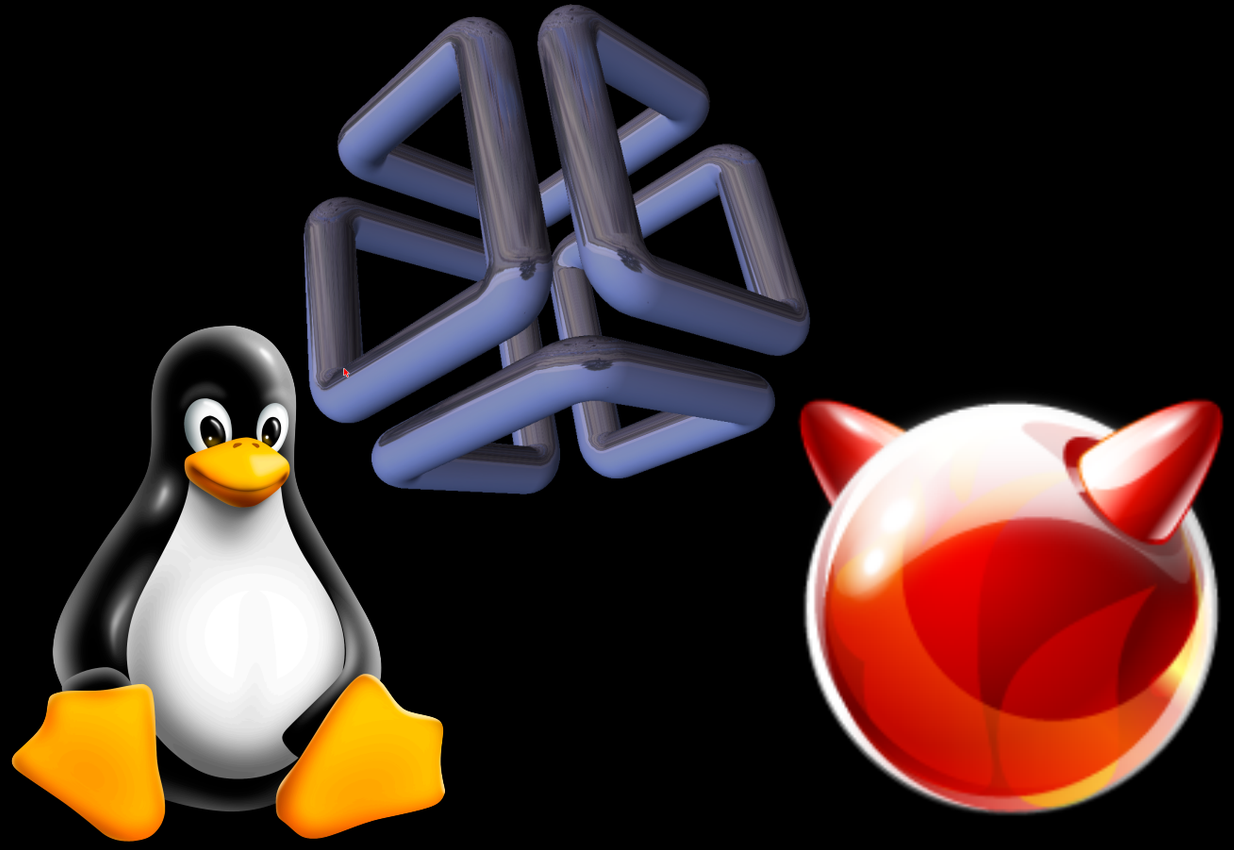
```
recvfrom(9,"…"...,8192,0,NULL,0x0)
sendto(9,"….”…,71,0,NULL,0)
```

- FreeBSD: setproctitle(3) makes two system calls
- New in FreeBSD 12: setproctitle_fast(3): no more syscalls!
- Result: ~10% increase in TPS on 40-core pgbench -S

Done in FreeBSD 12 + PostgreSQL 12

PROC_PDEATHSIG_CTL

```
13316  └─ postgres -D /data/clusters/main
13441     ├─ postgres: fred salesdb [local] idle
13437     ├─ postgres: fred salesdb [local] UPDATE
13337     ├─ postgres: fred salesdb [local] SELECT
13323     ├─ postgres: logical replication launcher
13322     ├─ postgres: stats collector
13321     ├─ postgres: autovacuum launcher
13320     ├─ postgres: walwriter
13319     ├─ postgres: background writer
13318     └─ postgres: checkpointer
```



- We want child processes to exit immediately if the ‘postmaster’ dies. Every process holds a pipe, but testing that is inconvenient and expensive during busy work loops
- Linux has prctl(PR_SET_PDEATHSIG), stolen from IRIX, to request a signal when your parent dies; PostgreSQL 12 now uses that
- New in FreeBSD 11.2: procctrl(PROC_PDEATHSIG_CTL)
- Result: replication/recovery is measurably faster*

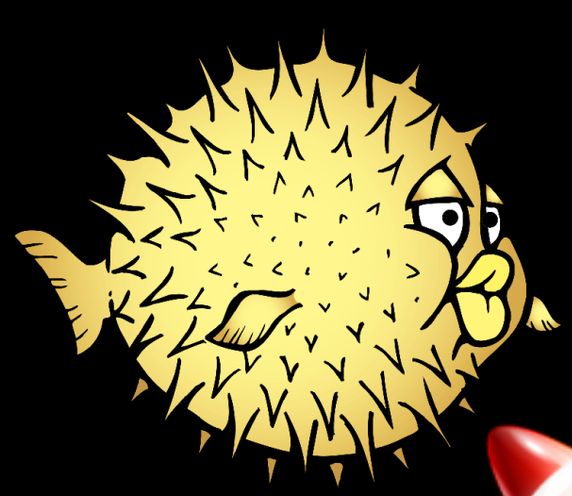
Done in FreeBSD 12

System V shared memory in jails

- Previously, multiple copies of PostgreSQL running in separate jails would interfere with each other, because System V shared memory was not jailed (there was one single key namespace for all jails on the same host); this required using different network ports or UIDs for PostgreSQL instances in different jails! (And probably worse things.)

- This was fixed in FreeBSD 11.

kqueue(2)



- PostgreSQL traditionally used poll(2) or select(2)
- Recently epoll(2) support was added for Linux, reusing an fd set to fix contention problems on large multi-socket machines
- Let’s use kqueue(2)!
- Result: up to ~50% more TPS on some high concurrency pgbench tests, but lower on some others!
- More research needed to understand; apparently related to timing-sensitive wakeup and scheduling logic in the kernel

sync_file_range()

- Bugzilla #203891
- Used by PostgreSQL (and Redis, MongoDB, Hadoop, …) to influence write-back rates, instead of (or in preparation for) the big hammer of fsyncdata()
- Seems easy (?) for UFS; I have no idea for ZFS (does ZFS-on-Linux support it?)

Idea for FreeBSD 13?

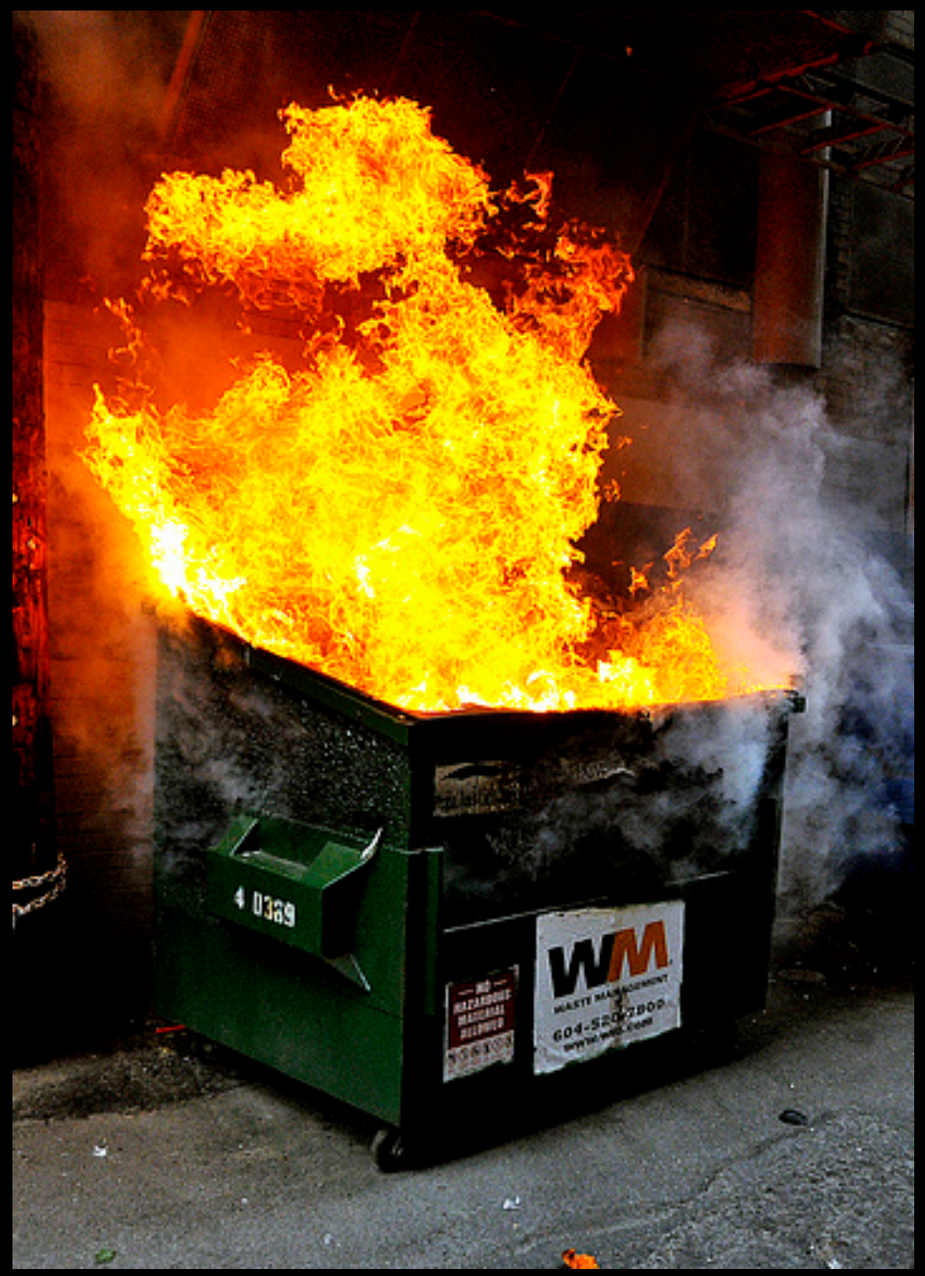
fsyncgate

- It turned out that PostgreSQL didn't understand the semantics of fsync() on a very popular operating system

- If fsync() reports EIO, *the kernel may have dropped your buffered data on the floor* even though it was still dirty; subsequent fsync() calls may therefore report success but your data is gone

- PostgreSQL (and MySQL and MongoDB and probably everyone else who spat out their coffee while reading LWN) now PANIC on any fsync() failure, rather than retrying

- Ancient Unix did the same, but FreeBSD doesn't have this problem since 1999; dirty data is dirty data, you can't drop it unless the device goes away, so future fsync() calls will also fail (or perhaps truly succeed)



Unicode collations

- Previously, FreeBSD couldn’t collate Unicode text with strcoll(). Almost everybody wants to use Unicode. The FreeBSD PostgreSQL port carried a patch to use ICU instead for collations.

- A new implementation was done for FreeBSD 11, sharing code with Illumos and DragonflyBSD.

- Recent PostgreSQL also supports ICU as a runtime option (you can use libc and ICU collations in the same database).

Fixed in FreeBSD 11

LC_VERSION_MASK

- Collations define the sort order of text with strcoll_l(3), and come from upstream sources like the Unicode CLDR project

- Order controls the structure of btree indexes

- Whenever collation definitions change silently, the indexes are corrupted … this really happens!

- Proposal forFreeBSD 13: a way to ask the OS for the version of the collation definition with querylocale(3) so we know when we need to rebuild indexes

```
cote
côte
coté
côté
```

Idea for FreeBSD 13?

Collation performance

- strcoll_l() currently expands string to wide characters every time, malloc->expand->free. Could we… not do that?

- PostgreSQL also copies string every time, to add NUL terminator! Non-standard strncoll_l()? (note “n”, was rejected from C99, semantics unclear)

- PostgreSQL can also use strxfrm_l() to sort text faster, but we turned it off because popular implementations were busted (didn’t always match strcoll_l() order, causing corruption); can we make a 100% reliable strxfrm_l()? strnxfrm_l() (note “n”)?

Some more ideas

SIGDANGER

- Warn processes before the OOM killer strikes (like AIX)
- Chance to free up some memory or exit voluntarily
- Is this useful?
- Similar ideas exist on iOS, and Linux user-space OOM tools

Ports ideas?

- In most Linux distributions, multiple PostgreSQL versions can be installed simultaneously (using paths like /usr/local/libexec/postgresql/12/bin/postgres)

- IMHO the Debian maintainers' postgresql-common package is the nicest way to manage starting/stopping/listing multiple PostgreSQL server instances on the same machine, possibly of different major versions

```
pg_lsclusters
pg_createcluster <version> <name>
…
```

- If someone were to port postgresql-common to FreeBSD, it would be to add special ZFS based commands (snapshot, clone, send/receive database clusters) and jails support

Shared memory

- The POSIX shm_open() facility has no way to list the segments that exist on the system! There is no ipcs or similar to see them. Solving this possibly involves changing the way shm_open() jailing works first.

- Once mapped into a process, procstat -v doesn’t show the name of the mapped segment. It should!

- shm_open() is managed with a fixed size smallish hash table with a big lock around it; perhaps that could be improved.

Development hurdles

- valgrind port doesn’t work for PostgreSQL (ENOSYS, aborts); it also contains a bunch of patches that are no upstreamed

- ~~dtrace ustack() broken in CURRENT~~

CloudABI?

- Software that is trapped entirely inside a directory, and starts with a file descriptor for its root directory
- Alternative to jails/containers/VMs etc
- Replace open(…) with openat(root_fd, …)
- Requires removing “global” stuff like SysV shm (need a new interlocking trick probably based on a file under pgdata, flock() not available) and POSIX shm (would need to file-backed mmap files for parallel query?)

Atomicity

- After each checkpoint (periodic restart point), PostgreSQL logs a “full page image” the first time you dirty each data page, to defend against torn pages on power loss

- This is not necessary if the filesystem guarantees 8KB atomic writes (= on power failure and restart, you either have the old 8KB page or the new 8KB page contents, but not some kind of frankenpage)

- ZFS can make such atomic guarantees due to its data journaling system, so you can set full_page_writes = off to avoid a ton of extra WAL IO on some workloads (a form of “write amplication” that people complain about on PostgreSQL)

- If UFS could somehow offer such guarantees too (for example because the underlying device can make a gurantee about power failure atomic write size) then it’s be nice if it could report that to us somehow

ZFS: Recycled file performance

- Joyent reported that PostgreSQL's recycling of WAL files, designed to be faster (on non-COW filesystems), was slower than simply creating new files every time on ZFS, for them

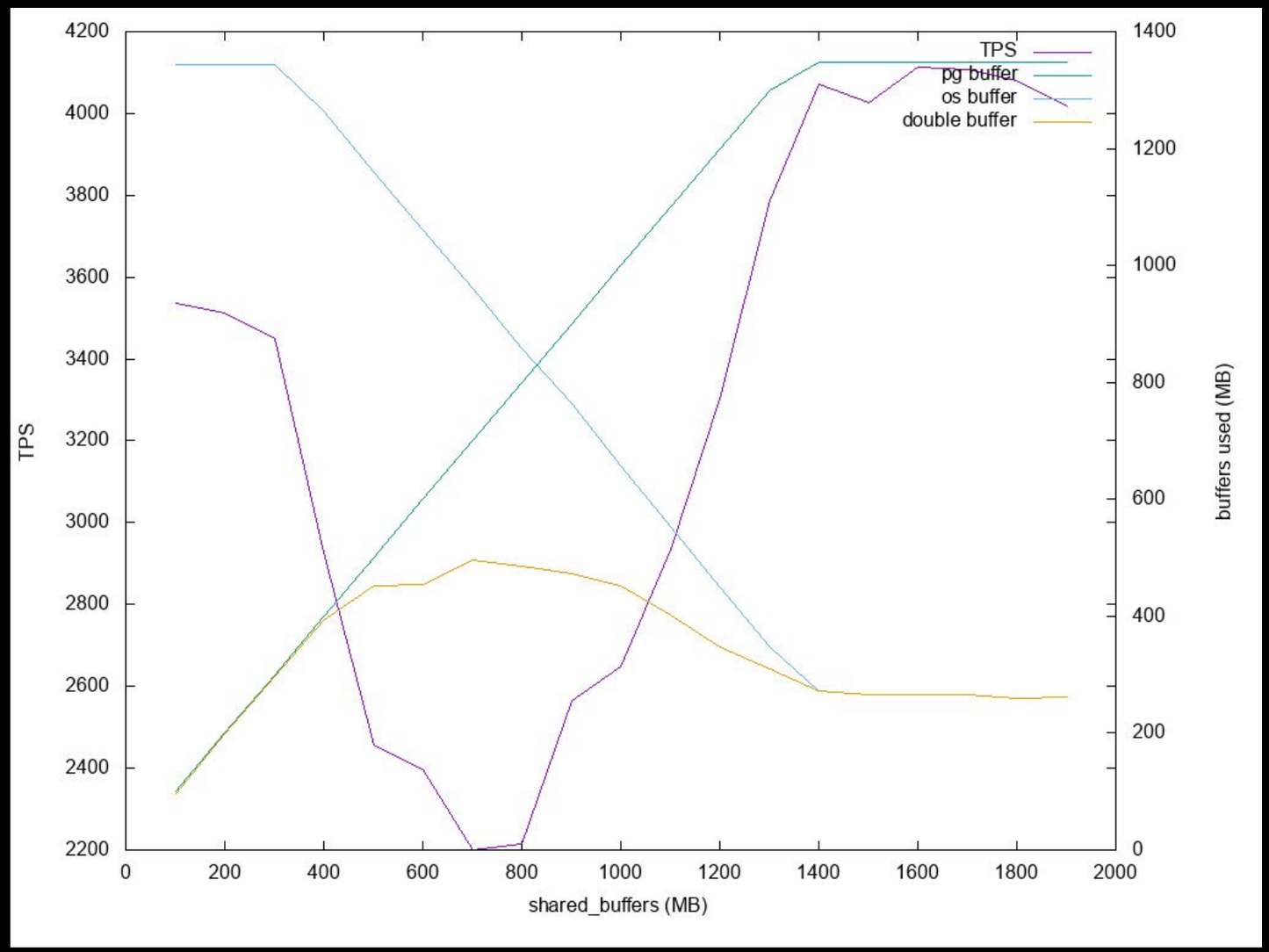
- I was able to reproduce this phenomenon on a low-end box; many others were unable to

- Hypothesis: it might be caused by block pointers and/or dnodes falling out of cache, so that when you come to write over the old file again you have to fault in some random access disk pages (even though the actual data pages don't need to be faulted in, because we entirely write over them and the record size matches PostgreSQL's block size)

- PostgreSQL should probably just take Joyent's patch to disable recycling; the whole concept doesn't really make sense for COW…

Double buffering

- If your data fits in PostgreSQL's buffers, performance is best
- If your data doesn't fit in PostgreSQL's buffers, but does fit in OS page cache, performance is nearly as good
- There is a valley of doom where it doesn't quite fit in either, even though you have enough physical RAM



Direct IO solves the double buffering problem but…

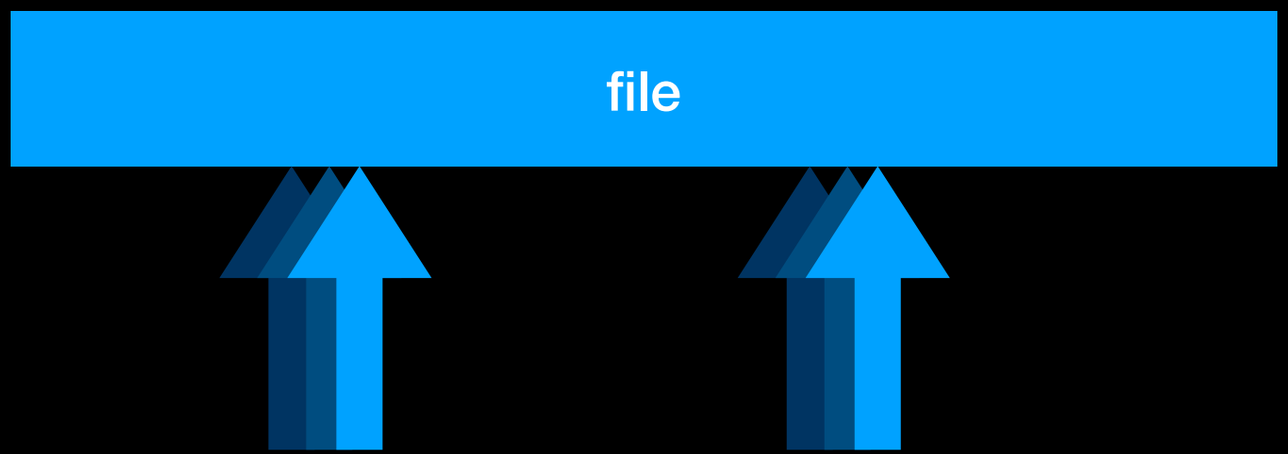
- Most relational databases added support for direct IO as it became available and reliable on operating systems

- You still need to support buffered IO for filesystems like ZFS

- Developing all the user-space IO scheduling machinery may be a bit tricky, but probably worth the pain

- In theory you could have a system call that allows “clean” data to be returned to the kernel page cache (or ARC), so you could have exclusive buffering and fix the “valley of doom”

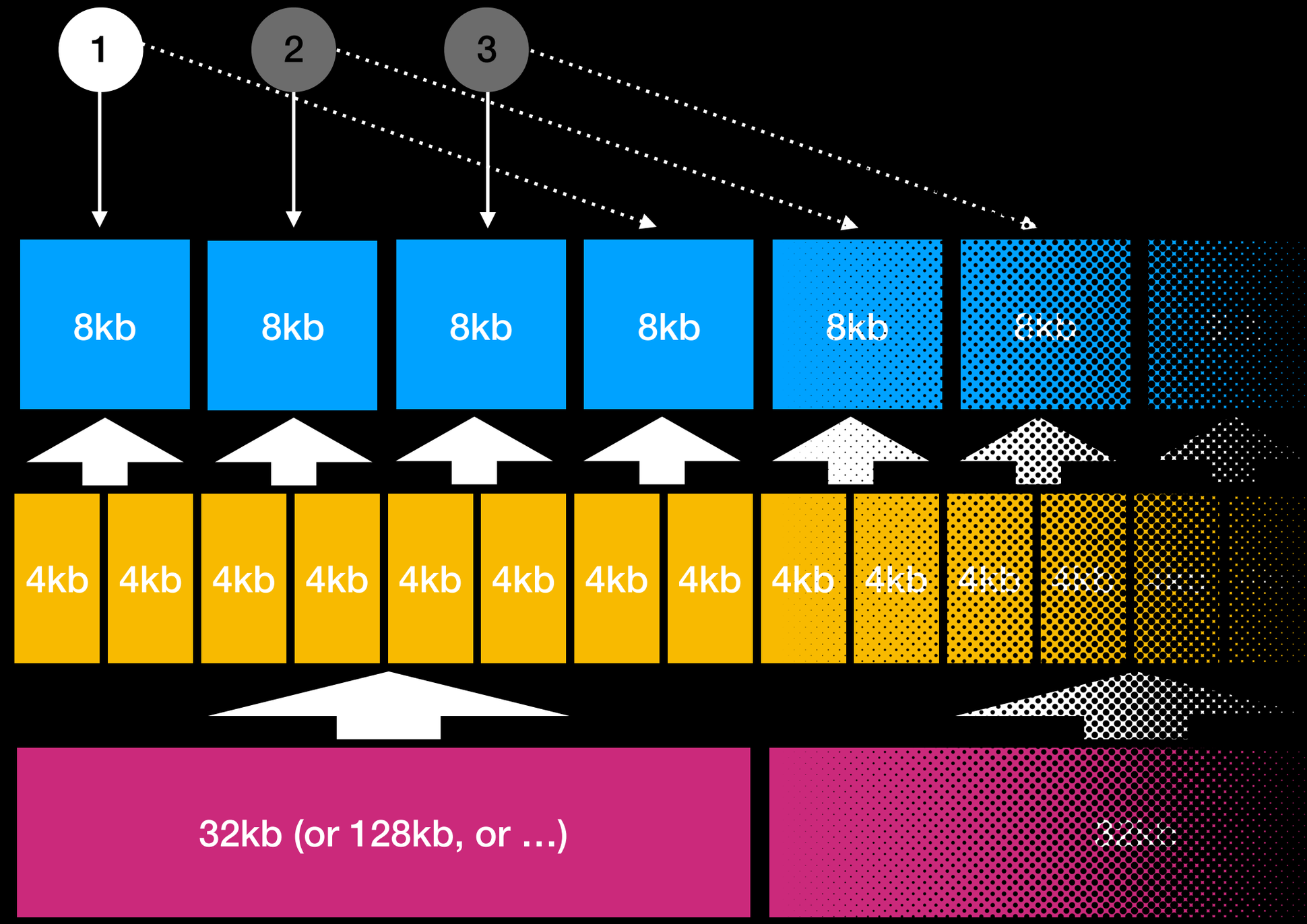
UFS: Read-ahead vs write-behind



- Sometimes PostgreSQL generates alternating reads and writes that are really sequential, but UFS doesn’t detect that
- vfs_nops.c’s fp->f_nextoff is a single int to track sequential access
- Should we track separate read and write nextoff variables? (But then why stop there, you could have more streams like ZFS does….)

UFS: Read-ahead vs parallel query execution

- Processes read 8kb pages into the PostgreSQL buffer pool
- The OS's **read-ahead** heuristics (hopefully) detects this pattern and ideally begins issuing larger reads to the disk to pre-load OS page cache pages
- We could do better with direct IO and our own IO scheduler



NFS

- PostgreSQL generally doesn’t work well on NFS, because it expects eager reservation (close() and fsync() are not great times to get ENOSPC, after pwrite() succeeded).
- NFSv4 protocol allows for ALLOCATE, but it’s not entirely clear how a humble userspace program can get the right guarantees (on various operating systems)
- Topic for further research
- Curiously some other famous relational database has its own NFS client implementation

Thank you!

Questions, more ideas?