aiven

Finding and fixing a data-corruption bug with the help of the community

Patrick Stählin

PG16 rollout

- Added as an option to our platform with 16.1
- We started to get some data-corruption cases, that could sometimes be resolved by recycling (promoting the standby)
- Only 0.15% of PG16 installations affected
- Only PG16 affected*
- We started rolling out 16.2 just two weeks before, so a lot of maintenances being applied

* or so we thought

Error

```
Feb 27 13:02:47 postgresql-deadbeef-2 postgres[3244179]: [20-1]
pid=3234179,user=avnadmin,db=defaultdb,app=foo,client=192.168.1.22 ERROR:  could not
read block 3 in file "base/102480/102484": read only 0 of 8192 bytes
```

Internal incident declared

- Following an initial suspicion that this will come to bite us, we declared an internal incident and also removed the possibility to create new PG16 instances
- We started to trace the origin of the error message and (re)-discovered how PG writes to files

Google PG16 “could not read block”

0 usable results

Analyzing the error

```
ERROR:  could not read block 3 in file "base/102480/102484": read only 0 of 8192
bytes
```

Offset in file (* 8 kB)

pg_class.relfilenode

Database OID

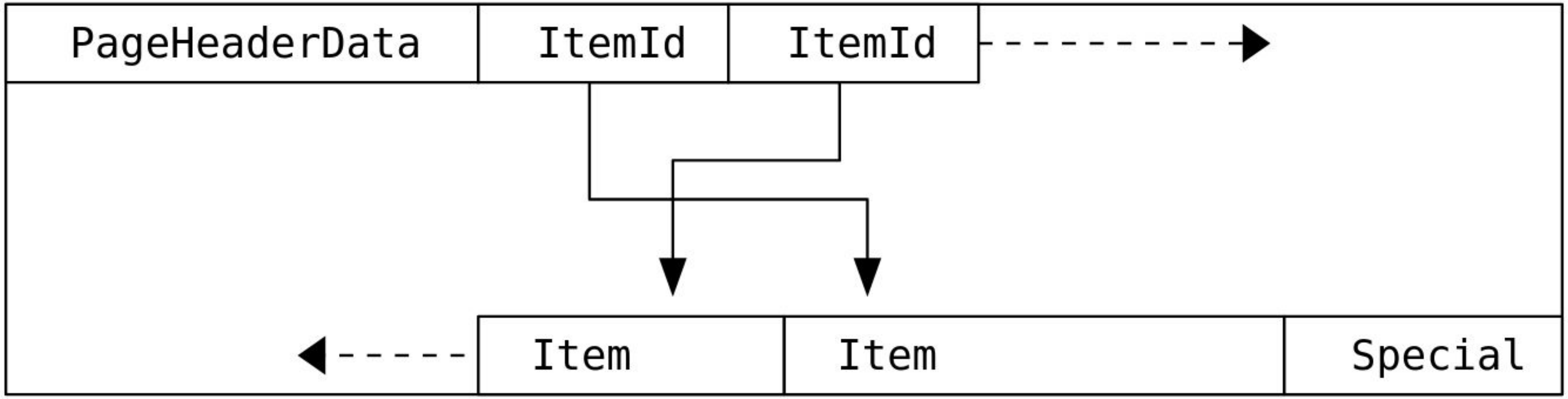
Files on-disk

```
$ tree /var/lib/postgresql/
[...]
base/
  102480/
     102484
     102484_fsm
     102484_vm
     [...]
```

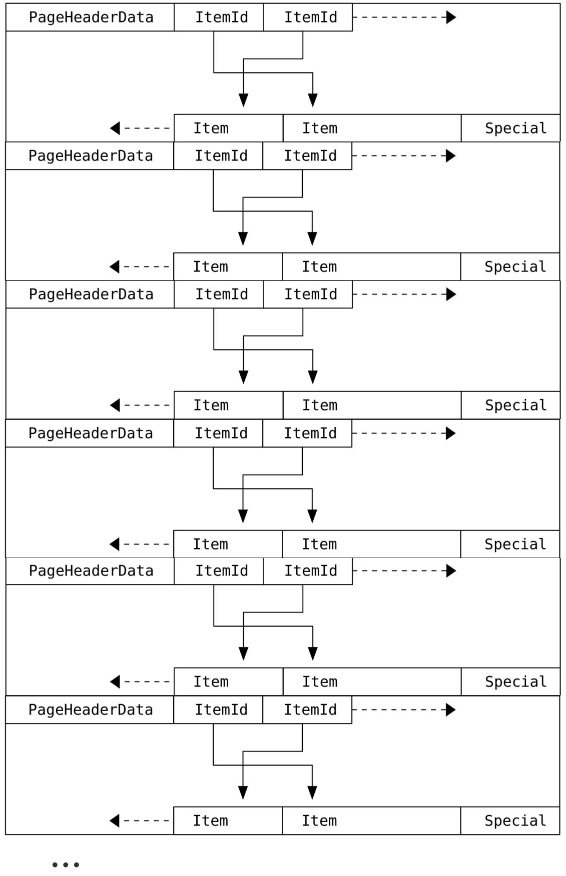
Data storage

- Files of up to 1GB
- Linked by `pg_class.relfilenode`
- Files grow on-demand
- Organized in pages/blocks of 8kB

Page layout



How PG organizes its data



page 0

page 1

page 2

...

To the source!

```
$ ack ‘could not read block’

src/backend/storage/smgr/md.c
782:        errmsg("could not read block %u in file \"%s\": %m",
798:        errmsg("could not read block %u in file \"%s\": read only %d of %d byte
```

src/backend/storage/smgr/md.c

```
nbytes = FileRead(...)

if (nbytes != BLCKSZ)
{
    if (nbytes < 0)
       ereport("could not read block  %u in file \"%s\": %m", … );

    if (!InRecovery && !zero_damaged_pages)
       ereport(“could not read block %u  in file \"%s\": read only %d of %
```

src/backend/storage/smgr/md.c

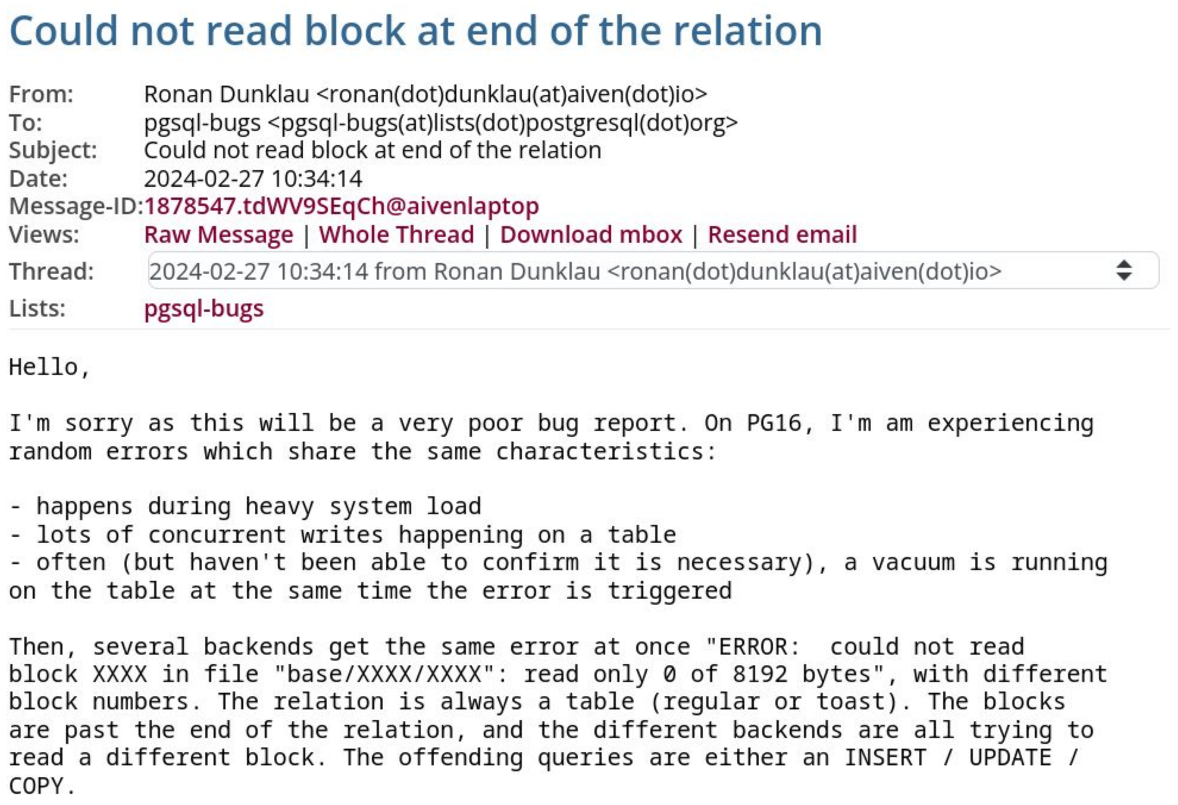
```
nbytes = FileRead(...)

if (nbytes != BLCKSZ)
{
    if (nbytes < 0)
       ereport("could not read block  %u in file \"%s\": %m", …);

    if (!InRecovery && !zero_damaged_pages)
       ereport(“could not read block %u in file \"%s\": read only %d of %d
```

👍 No corruption in the file, it’s “just short”

Who you gonna call?

pgsql-bugs!

Reaching out to the community

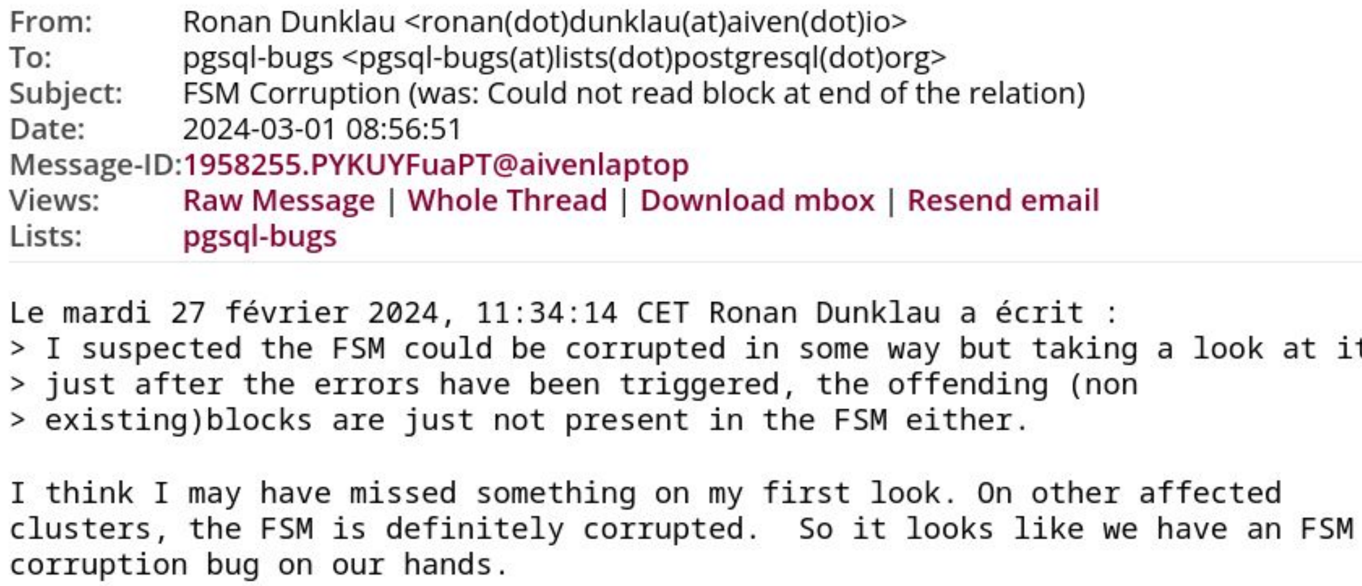


Could not read block at end of the relation

From:	Ronan Dunklau <ronan(dot)dunklau(at)aiven(dot)io>
To:	pgsql-bugs <pgsql-bugs(at)lists(dot)postgresql(dot)org>
Subject:	Could not read block at end of the relation
Date:	2024-02-27 10:34:14
Message-ID:	1878547.tdWV9SEqCh@aivenlaptop
Views:	Raw Message | Whole Thread | Download mbox | Resend email
Thread:	2024-02-27 10:34:14 from Ronan Dunklau <ronan(dot)dunklau(at)aiven(dot)io>
Lists:	pgsql-bugs

```
Hello,

I'm sorry as this will be a very poor bug report. On PG16, I'm am experiencing
random errors which share the same characteristics:

- happens during heavy system load
- lots of concurrent writes happening on a table
- often (but haven't been able to confirm it is necessary), a vacuum is running
on the table at the same time the error is triggered

Then, several backends get the same error at once "ERROR:  could not read
block XXXX in file "base/XXXX/XXXX": read only 0 of 8192 bytes", with different
block numbers. The relation is always a table (regular or toast). The blocks
are past the end of the relation, and the different backends are all trying to
read a different block. The offending queries are either an INSERT / UPDATE /
COPY.
```

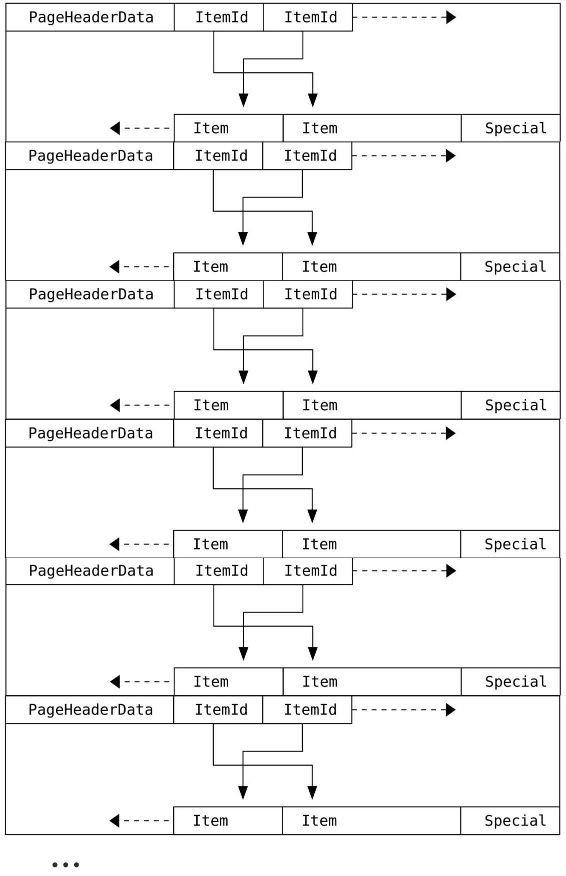
Reaching out to the community, again



From: Ronan Dunklau <ronan(dot)dunklau(at)aiven(dot)io>
To: pgsql-bugs <pgsql-bugs(at)lists(dot)postgresql(dot)org>
Subject: FSM Corruption (was: Could not read block at end of the relation)
Date: 2024-03-01 08:56:51
Message-ID: 1958255.PYKUYFuaPT@aivenlaptop
Views: Raw Message | Whole Thread | Download mbox | Resend email
Lists: pgsql-bugs

```
Le mardi 27 février 2024, 11:34:14 CET Ronan Dunklau a écrit :
> I suspected the FSM could be corrupted in some way but taking a look at i
> just after the errors have been triggered, the offending (non
> existing)blocks are just not present in the FSM either.

I think I may have missed something on my first look. On other affected
clusters, the FSM is definitely corrupted.  So it looks like we have an FSM
corruption bug on our hands.
```

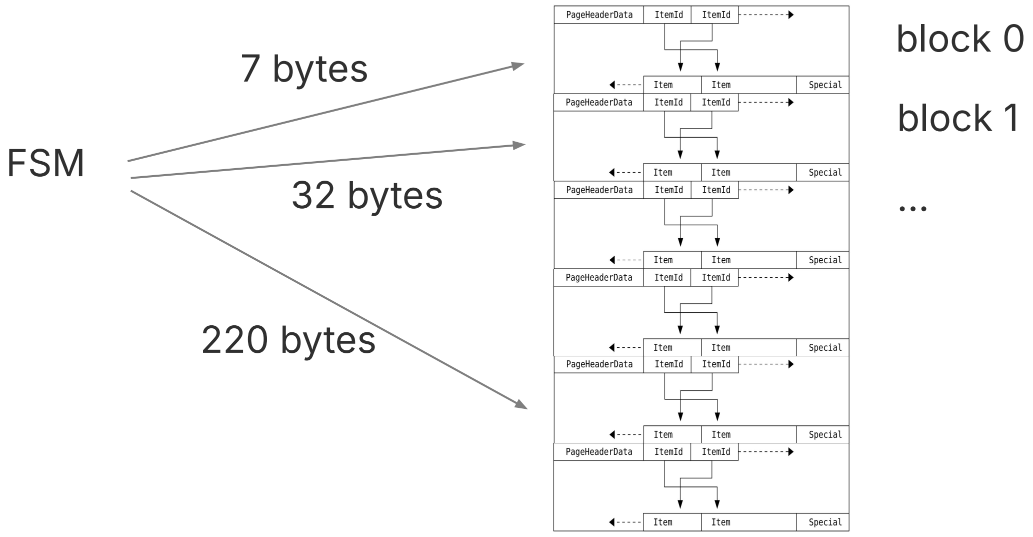
FSM, or where to write?



Free Space Map (FSM)

- Can return the next page/block with at least N bytes of space
- Rebuilt on `VACUUM FULL`
- Rebuilt on `ANALYZE FULL` if missing
- Stored in `base/<db_oid>/<pg_class.relfileid>_fsm`

Where to write?



...

Detecting FSM errors

```
postgres defaultdb= # SELECT oid AS reloid,
       pg_relation_filepath(oid) || '_fsm' AS fsm
FROM pg_catalog.pg_class,
     CAST(pg_catalog.current_setting('block_size') AS bigint) AS bs
WHERE relkind IN ('r', 'i', 't', 'm') AND EXISTS
  (SELECT 1 FROM
   generate_series(pg_catalog.pg_relation_size(oid) / bs,
                   (pg_catalog.pg_relation_size(oid, 'fsm') - 2*bs) / 2) AS blk
   WHERE freespacemap.pg_freespace(oid, blk) > 0);
 reloid |           fsm
--------+------------------------
  18265 | base/16421/112775_fsm
  18255 | base/16421/112677_fsm
  18079 | base/16421/112654_fsm
  18274 | base/16421/112780_fsm
(4 rows)
```

From https://wiki.postgresql.org/wiki/Free_Space_Map_Problems

Fixing it (temporarily)

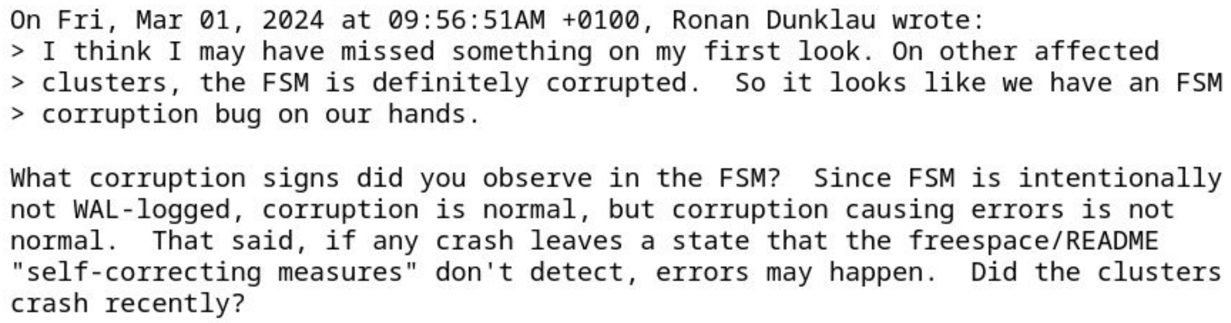
- Rebuild it by VACUUM FULL
  - - locks the relation
  - + fixes it without restart

- Remove FSM and rebuild it (+ no locking, - restart)
  - CHECKPOINT; CHECKPOINT;
  - systemctl postgresql-16 stop
  - rm base/<db_oid>/<pg_class.relfileid>_fsm
  - systemctl postgresql-16 start
  - ANALYZE FULL foo;

Fixing it (by cheating)

- We have a custom extension where we can add functionality
  - We added a `pg_truncate_freespace` function to remove the FSM with just a short exclusive lock
- The patch has also submitted to be added to the `pg_freespace` extension but it was deemed a bit too dangerous

Getting some actionable feedback



```
On Fri, Mar 01, 2024 at 09:56:51AM +0100, Ronan Dunklau wrote:
> I think I may have missed something on my first look. On other affected
> clusters, the FSM is definitely corrupted.  So it looks like we have an FSM
> corruption bug on our hands.

What corruption signs did you observe in the FSM?  Since FSM is intentionally
not WAL-logged, corruption is normal, but corruption causing errors is not
normal.  That said, if any crash leaves a state that the freespace/README
"self-correcting measures" don't detect, errors may happen.  Did the clusters
crash recently?
```

Looking at the root cause again

- Focus on FSM corruption cases
- Look at and dump WAL files for small-ish relations/FSMs, as the error seems to propagate

Write path for a single tuple

- We consult the FSM to get a page with enough space
- If, after locking the page, we don’t see enough free space, move to the next one
- If all pages are full, grow the file by one page

Growing a file

- Changed slightly in PG16
- We now count all the processes holding a lock and allocate “enough for everybody”™
- We knew that the bug exposes itself in that code-path
- But staring at it didn’t make it easier to find

Growing a file

- Extend the FSM
- Extend the file
- Great success!

Extending the FSM

- Is WAL logged in certain conditions
- Extends the file by zeroed pages

Extending relation

- Is **NOT** WAL logged until something writes to it
- Extends the file by zeroed pages

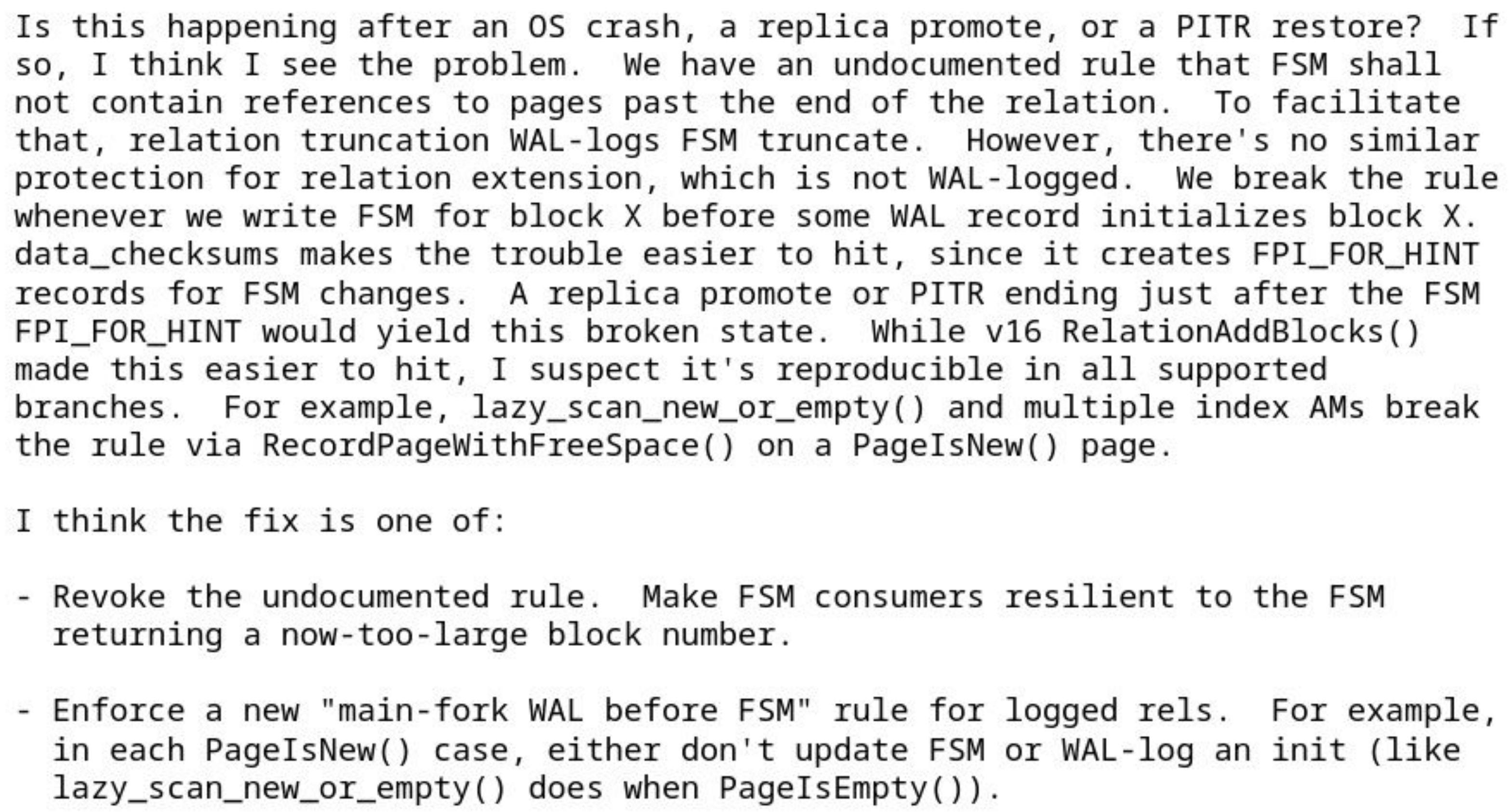
Getting into an error state

- As long as everything stays on one machine there is no issue, as the files on-disk are OK
- If you fail-over, do a restore or a PITR you can end up with the WAL-record already applied to the FSM but no data written to the relation.
- This got amplified as we started to allocate more space for all waiting backends, making it easier to hit.

Consequences

- If the FSM points to a block that is beyond the file boundary we would just fail a write
- This would fail the transaction and issue a rollback
- It can be self-corrected by needing more space than is available in a single block, then the underlying relation would be extended again.
- This made it very difficult to reproduce as the right amount of data is needed.

How to fix it (from Noah Misch)?



Is this happening after an OS crash, a replica promote, or a PITR restore? If so, I think I see the problem. We have an undocumented rule that FSM shall not contain references to pages past the end of the relation. To facilitate that, relation truncation WAL-logs FSM truncate. However, there's no similar protection for relation extension, which is not WAL-logged. We break the rule whenever we write FSM for block X before some WAL record initializes block X. data_checksums makes the trouble easier to hit, since it creates FPI_FOR_HINT records for FSM changes. A replica promote or PITR ending just after the FSM FPI_FOR_HINT would yield this broken state. While v16 RelationAddBlocks() made this easier to hit, I suspect it's reproducible in all supported branches. For example, lazy_scan_new_or_empty() and multiple index AMs break the rule via RecordPageWithFreeSpace() on a PageIsNew() page.

I think the fix is one of:

- Revoke the undocumented rule. Make FSM consumers resilient to the FSM returning a now-too-large block number.

- Enforce a new "main-fork WAL before FSM" rule for logged rels. For example, in each PageIsNew() case, either don't update FSM or WAL-log an init (like lazy_scan_new_or_empty() does when PageIsEmpty()).

Naive approach

- If the FSM points to a block that is beyond the file check it in the caller and report it as full
- The problem is that this is at least one system call per tuple insert
- Noticeable slow-down in benchmarks we ran (+1.25%)

Refined approach by Noah

- When finding a FSM entry
- Check the size of the relation
- This is still a system-call, but it gets cached on a process-level
- Slow down by ~0.1%
- Patched as of 16.3 and backpatched

Take aways

- Talk to the community, even if your initial investigation turns out to find nothing. Your findings will help another person.
- Test and release new versions, new PG versions are very stable
- Monitor your systems/logs!
- If you have the possibility to ship your own extension (or PG version), you can react very quickly to reduce operational load.

Thank you!

Feedback →

Patrick Stählin

Member of the PostgreSQL team at Aiven

Senior Software Developer, Aiven

https://linkedin.com/in/patrickstaehlin

patrick.staehlin@aiven.io