Restore your backups!

Abhijit Menon-Sen
2ndQuadrant

February 14, 2019

Do you have backups?

Once upon a time...

More people have backups now

Today's problems are different

A sensible backup strategy?

Documented procedures?

Not just nice to have

This talk is...

Not about any of those things

No Barman, no pgBackRest

Let's talk about Postgres

pg_dump is awesome

Easy to understand and use

Consistent logical backups

Partial dumps and restores

Small size, no bloat

Cross-version

But pg_dump is slow

Slow to backup (lock waits)

Slow to restore (CPU and IO)

Variable, unpredictable timing

Still usable for small databases

Small databases are an endangered species

How can we do better?

Can't predict Postgres operations

But disks are fairly predictable

A hundred or few hundred MB/s

Perform IO outside Postgres

All or nothing

Just tar up PGDATA

Goes block-by-block

x MB ÷ y MB/s == z s (we hope)

It mostly does work that way

But it's not atomic

Postgres changes stuff alongside

Who needs consistency?

Postgres writes, tar reads

Neither knows about the other

“File changed as we read it”

pg_dump uses a snapshot

No SQL → no snapshot

Aside: an apology

I wanted to make things blow up

I wanted scary error messages!

I managed only boring data loss

Still, boring or not...

Data loss is bad 🤦‍♀️

Things we don't like

SQL snapshots ⇒ slow

No snapshots ⇒ data loss

Shutdown ⇒ downtime

All pretty unpopular

How can we do better?

We need atomic snapshots

Freezing the volume sucks

(What if there are many volumes?)

We don't have fancy hardware

It has to work everywhere

Postgres has to do it

What's a snapshot anyway?

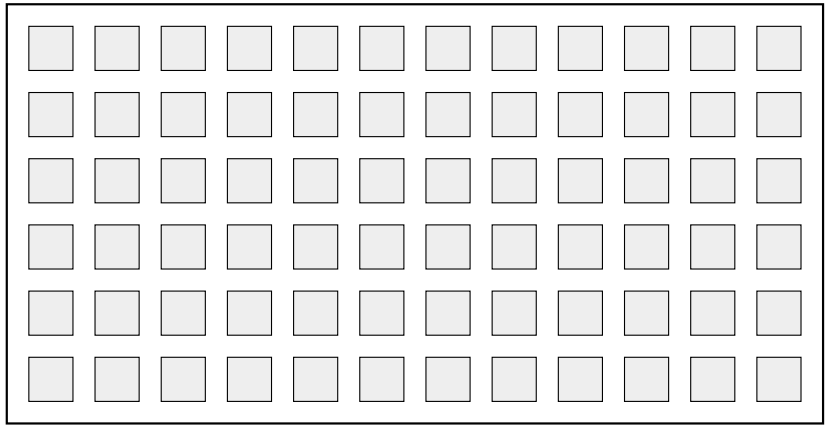
Contents of blocks[0…N] at t_N

Concurrent reads: no problem

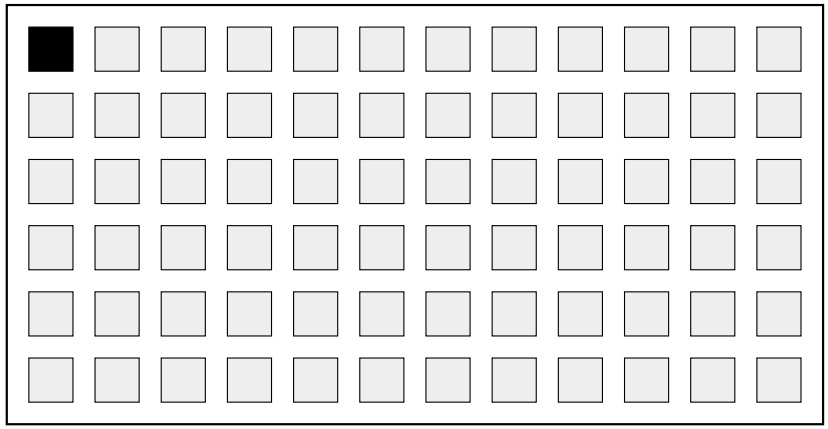
Writes to blocks[x < X]: OK

Writes to blocks[x > X]: Oops

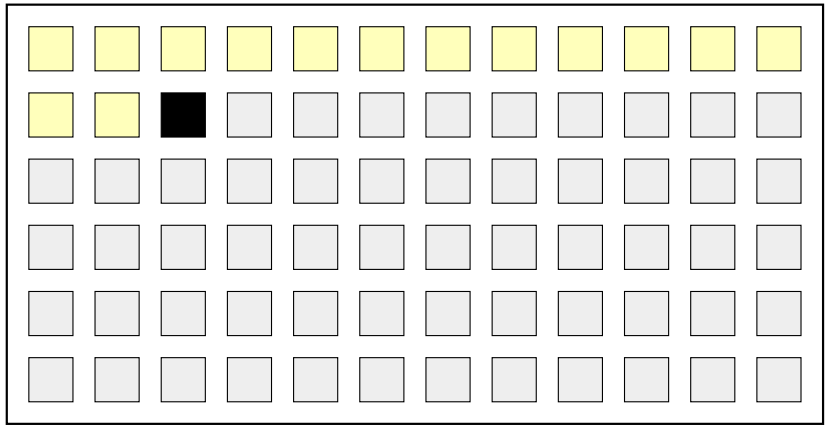
Before snapshot



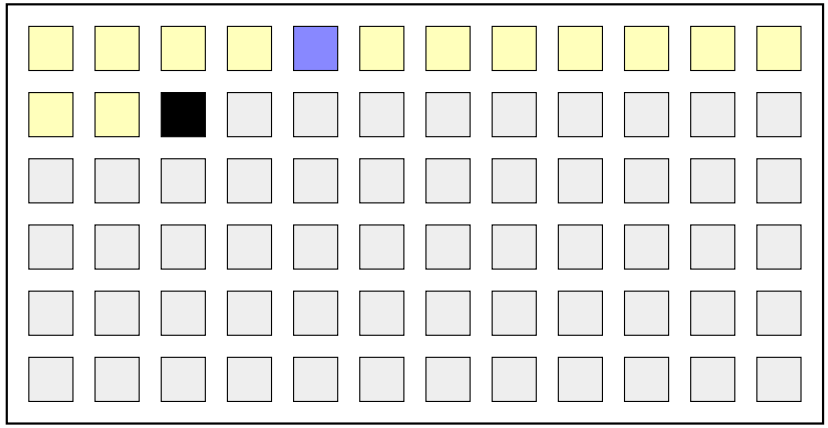
Snapshot starts at t_N



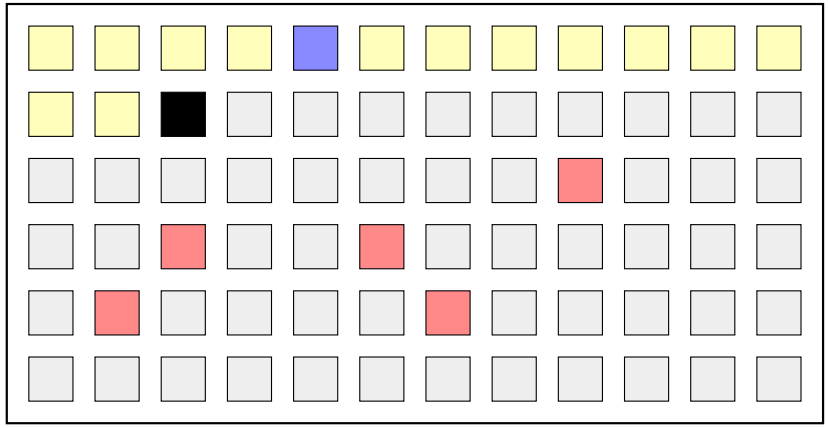
Snapshot progress at t_{N+m}



Writes to read blocks: OK



Writes to unread blocks: Oops



Copy-on-write

Track writes after t_N

Copy any changed blocks

Snapshot uses blocks[x].old

Readers use blocks[x].new

Multiple changes OK

Sounds a lot like MVCC 🤔

Postgres must know

(Because Postgres has to do it)

Tell Postgres: start the backup

```
tar cf backup.tar PGDATA
```

Tell Postgres: stop the backup

Start the backup

```
postgres=# select pg_start_backup('x', exclusive := false);
 pg_start_backup
-----------------
 0/16000028
(1 row)
```

Now copy the files

Copy everything in PGDATA

rsync, tar, whatever

Or an atomic filesystem snapshot

Just get the files somehow

Stop the backup

```
postgres=# select * from pg_stop_backup(false);
NOTICE:  WAL archiving is not enabled; you must ensure that all required
WAL segments are copied through other means to complete the backup
    lsn     |                            labelfile                                | spcmapfile
------------+-----------------------------------------------------------------+------------
 0/160000F8 | START WAL LOCATION: 0/16000028 (file 000000010000000000000016)+|
            | CHECKPOINT LOCATION: 0/16000060                               +|
            | BACKUP METHOD: streamed                                       +|
            | BACKUP FROM: master                                           +|
            | START TIME: 2019-02-14 03:16:37 IST                           +|
            | LABEL: x                                                      +|
            |                                                                |
(1 row)
```

That's all folks!

```
NOTICE:  WAL archiving is not enabled; you must ensure that all required
WAL segments are copied through other means to complete the backup
```

???

Isn't the backup complete?

What happened to copy-on-write anyway?

Everything is backwards

We *don't* make a consistent snapshot

tar just reads whatever's there

It could be wildly inconsistent

We can fix it afterwards

Just replay WAL

Why? Why?!?!

That's how WAL already works

It's how crash recovery works

Filesystem contents may be inconsistent

We have to be able to cope

Just replay WAL

So we need the WAL

```
NOTICE:  WAL archiving is not enabled; you must ensure that all required
WAL segments are copied through other means to complete the backup
```

From the start of the backup

Until the end of the backup

Stream, archive, or “other means”

What does restoring mean?

Create PGDATA from copied files

Start Postgres

Let it replay WAL

It reaches consistency

Congratulations all around!

pg_basebackup

pg_start_backup

Copy (most of) PGDATA

Stream WAL concurrently

pg_stop_backup

Write backup label file

Taking a backup

pg_basebackup is simple

Run command, get output

```
$ pg_basebackup -h0 -p15432 -R -v -P -c fast -D x
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 1/33020100 on timeline 1
pg_basebackup: starting background WAL receiver
3104691/3104691 kB (100%), 1/1 tablespace
pg_basebackup: write-ahead log end point: 1/472F8B58
pg_basebackup: waiting for background process to finish streaming ...
pg_basebackup: base backup completed
```

How long will the backup take?

Time for initial checkpoint

Time to copy all the data

Time to archive WAL

Checkpoint: spread vs. fast

WAL: archive vs. stream

How long will the restore take?

Time to copy all the data

Time to replay all the WAL

???

WAL replay is unpredictable

Depends on WAL volume

…but not linearly

Rate varies by operation

You can measure average rate

At least it will keep you occupied

while waiting for it to finish

Controlling recovery

Recovery controlled by recovery.conf

Written by hand or `pg_basebackup -R`

WAL from pg_wal or via restore_command

Default: recover to end of WAL

(Remember: infinite WAL archive)

End recovery ASAP

`recovery_target='immediate'`

Stop when consistency reached

For us, at the end of backup

Won't replay rest of WAL

Point in time recovery

Specify any one of:

recovery_target_time

recovery_target_xid/lsn

recovery_target_name
(pg_create_restore_point('name'))

Also recovery_target_inclusive

Stop before/after the point

Recover everything

The default if you don't specify anything

recovery_target_timeline='latest' follows timeline changes

(Most useful for replicas)

Recovery action

What to do when recovery point reached

`recovery_target_action`

pause, promote, shutdown

Set recovery point, test, change point, continue

`recovery_end_command`

Extra: exclusive backups

Remember `exclusive:=false`?

pg_start_backup would write the backup label to PGDATA

Postgres wouldn't start after a crash

Didn't work on replicas anyway

Deprecated

Extra: full_page_writes

pg_start_backup enables full_page_writes

Includes complete copy of changed block in WAL

Usually done on first write after checkpoint

Always done during backup

Extra: backup label

```
START WAL LOCATION: 1/33020100 (file 000000010000000100000033)
CHECKPOINT LOCATION: 1/333577F8
BACKUP METHOD: streamed
BACKUP FROM: master
START TIME: 2019-02-14 05:27:08 IST
LABEL: pg_basebackup base backup
```

Extra: pg_controldata (checkpoint info)

```
$ pg_controldata -D x|grep checkpoint
Latest checkpoint location:           1/333577F8
Prior checkpoint location:            1/2D000098
Latest checkpoint's REDO location:    1/33020100
Latest checkpoint's REDO WAL file:    000000010000000100000033
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:148809
Latest checkpoint's NextOID:          32768
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        548
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  148795
Latest checkpoint's oldestMultiXid:   1
Latest checkpoint's oldestMulti's DB: 1
Latest checkpoint's oldestCommitTsXid:0
Latest checkpoint's newestCommitTsXid:0
Time of latest checkpoint:            Thu 14 Feb 2019 05:27:06 AM IST
```

Questions?

Thank you