Solving PostgreSQL connection scalability: Insights from CERN’s GitLab Service

Maurizio De Giorgi, Ismael Posada Trobo

27th Jun 2024

Maurizio De Giorgi

- ➡ Senior Database Engineer at CERN since Sep 2020
- ➡ DB on Demand: Service Manager and DevOps
- ➡ Long career in many different roles, industry, markets with a strong focus on databases and data stores
- ➡ Always looking at new technology, paradigms and trends

[DB on Demand is hiring a early career technician!]

[Maurizio De Giorgi]

[maurizio.degiorgi@cern.ch]

Ismael Posada Trobo

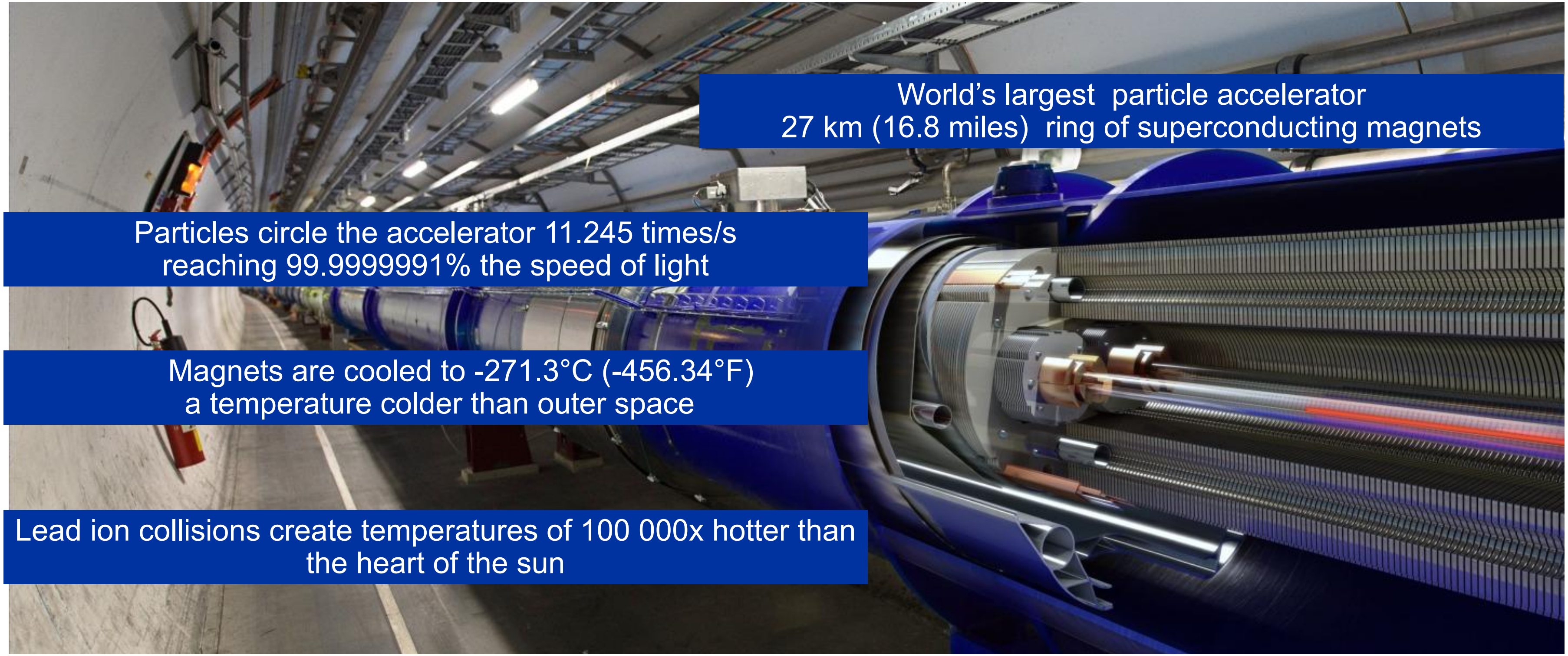
- ➡ Enthusiast Cloud Engineer at CERN since 2014
- ➡ Version Control Systems Tech Lead and Engineering Manager at CERN
- ➡ GitLab Contributor and member of the GitLab Customer Advisory Board
- ➡ Author of several scientific papers
- ➡ Several years of experience in Cloud technologies, fueled by a passion for technologies

in Ismael Posada Trobo

✉ ismael.posada.trobo@cern.ch

- Established in 1954
- 23 Member states
- Our mission:
  - Unveil how the universe works and what it is made of
  - Provide a unique range of particle accelerator facilities to enable research at the forefront of the human knowledge
  - Unite people from all over the world to push the frontiers of science and technology

The Large Hadron Collider



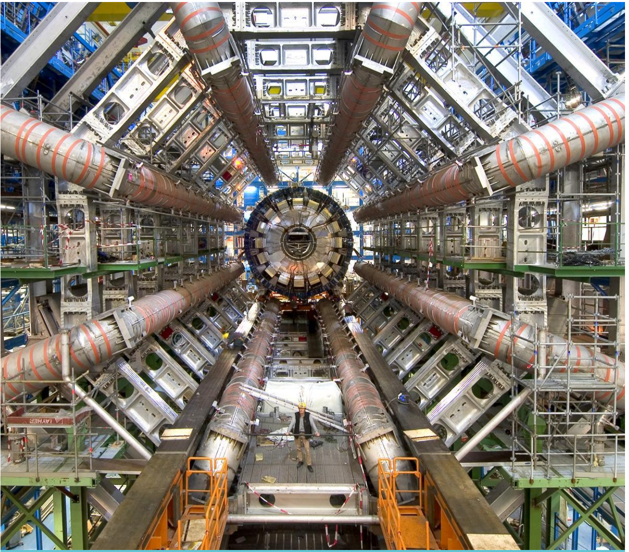
World’s largest particle accelerator
27 km (16.8 miles) ring of superconducting magnets

Particles circle the accelerator 11.245 times/s
reaching 99.9999991% the speed of light

Magnets are cooled to -271.3°C (-456.34°F)
a temperature colder than outer space

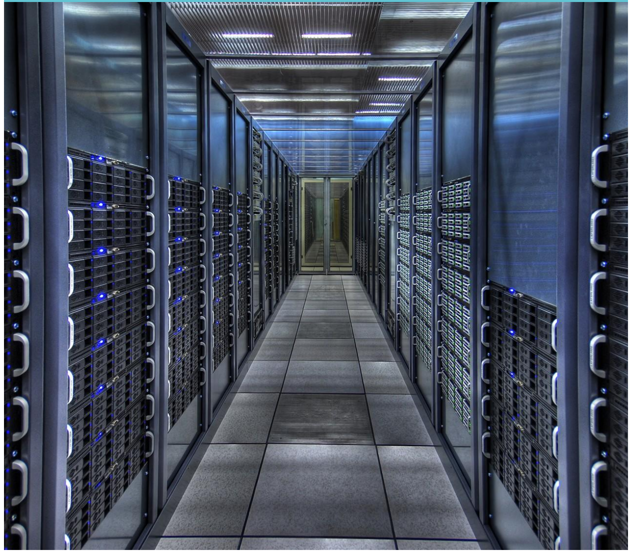
Lead ion collisions create temperatures of 100 000x hotter than
the heart of the sun

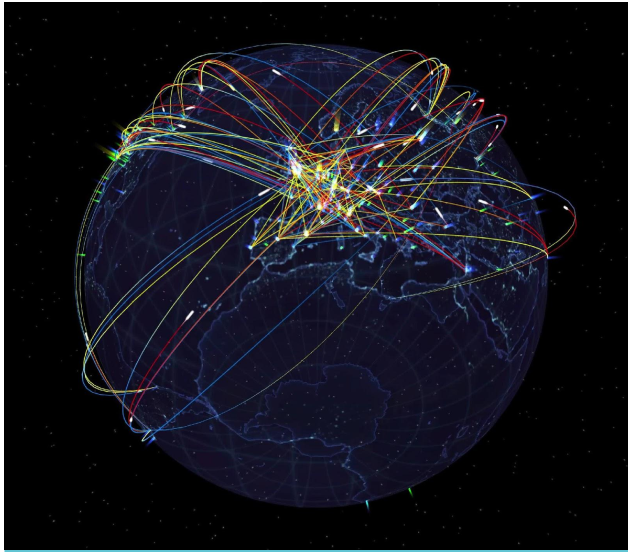
The Worldwide LHC Computing Grid (WLCG)



1 PB of data per second
Only 1% is kept (events with specific characteristics)

Tier0:
Data reconstruction + Tape archival
+ data distribution to other tiers
~ 200 PB of data per year

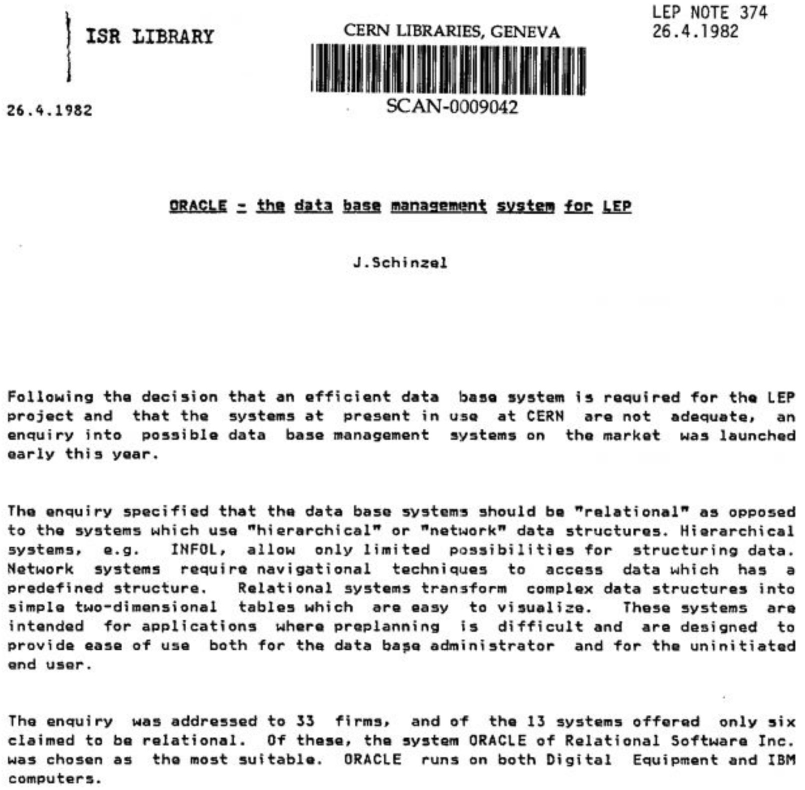




WLCG:
- 170 collaborating centers
- 36 countries
- Data analysis

Databases at CERN: Oracle

- **Oracle databases since 1982**
  - 105 Oracle databases
  - More than 11.800 Oracle accounts
  - RAC, Active Data Guard, OEM, RMAN…
  - Complex environment
  - Used by
    - Administrative Information Services
    - Engineering teams
    - Accelerator and experiments
  - Full DBA support
  - ≈ 5PB of data



LEP NOTE 374
26.4.1982

26.4.1982

ORACLE - the data base management system for LEP

J.Schinzel

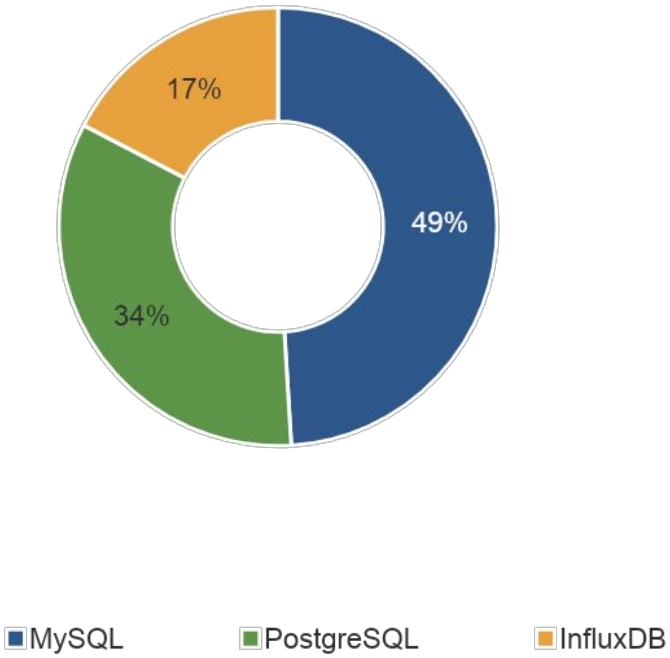
Following the decision that an efficient data base system is required for the LEP project and that the systems at present in use at CERN are not adequate, an enquiry into possible data base management systems on the market was launched early this year.

The enquiry specified that the data base systems should be "relational" as opposed to the systems which use "hierarchical" or "network" data structures. Hierarchical systems, e.g. INFOL, allow only limited possibilities for structuring data. Network systems require navigational techniques to access data which has a predefined structure. Relational systems transform complex data structures into simple two-dimensional tables which are easy to visualize. These systems are intended for applications where preplanning is difficult and are designed to provide ease of use both for the data base administrator and for the uninitiated end user.

The enquiry was addressed to 33 firms, and of the 13 systems offered only six claimed to be relational. Of these, the system ORACLE of Relational Software Inc. was chosen as the most suitable. ORACLE runs on both Digital Equipment and IBM computers.

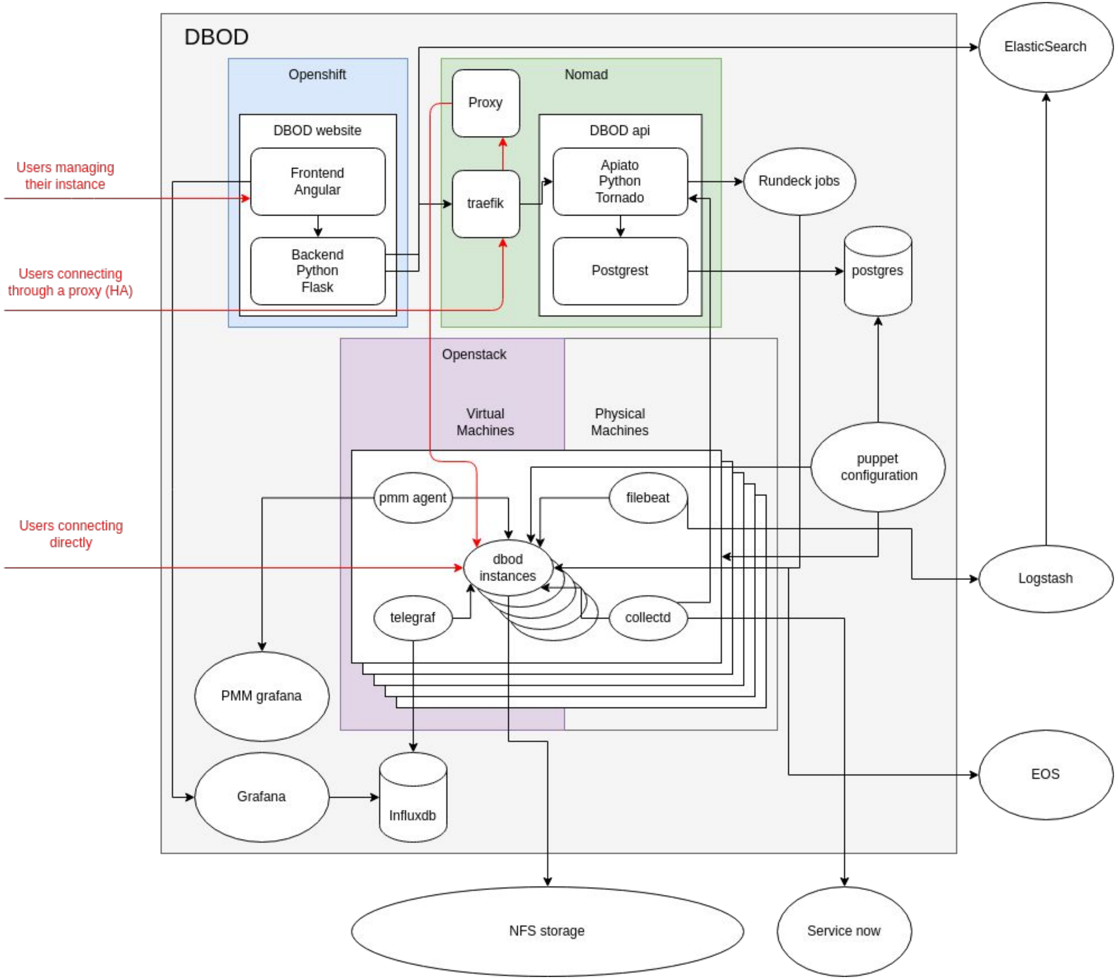
Databases at CERN: DBOD

- **Database On Demand (DBOD)**
  - DBaaS conceived in 2011
  - User-managed MySQL, PostgreSQL, InfluxDB database instances
  - Empowers users to be their own DBA
  - Flexible architecture allowing to easily integrate other DBMS
  - More than 1200 database server instances
    - ≈600 MySQL, ≈400 PostgreSQL, ≈200 InfluxDB
    - ≈150 TB of data
  - A number of key database applications:
    - DBOD own databases
    - Authorization and authentication (SSO)
    - Experiments (ATLAS, LHCb, etc.)
    - WLCG File Transfer Service
    - GitLab, Puppet, Foreman, Teigi (secrets)
    - Openstack (nova, ironic)
    - Security (some SOC apps)
    - Indico, Zenodo, Jira, ServiceNow



DBOD Architecture

- Complex DBaaS environment
- Integrated with CERN infrastructure
- Mostly open source
- Infrastructure as Code
- Deploy on VM/Bare Metal
- Systemd managed services
- NetApp Storage
  - data/wals NFS volumes
  - snapshot based backups
- EOS (EOS Open Storage)
  - snapshots copy archive
  - wals archive



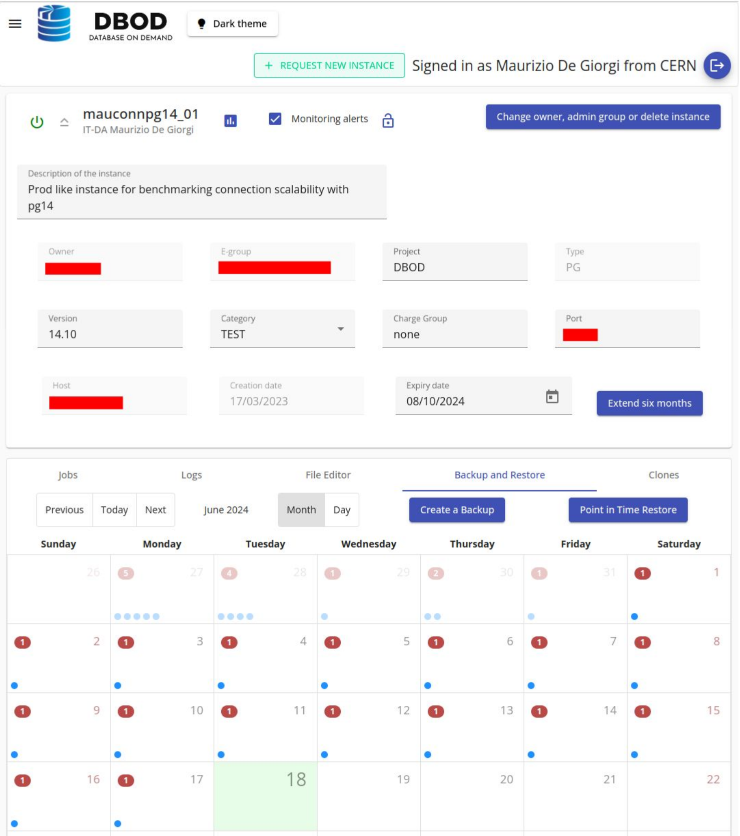
DBOD Automation

Web automation

- Automated backup and recovery services
- Upgrade checker to enable self-service upgrades
  - once errors and warnings in the report are fixed
- Management of configuration files
- Cloning
- Integrated monitoring
- Integrated upgrades
  - Primary-replica upgrade logic

Ops automation

- Continuous validation of backups
- Instance and storage migration
- Automated replica provisioning
- Automated replication switchover
- Detection of idle instances
- Integrated password hash cracker



Signed in as Maurizio De Giorgi from CERN

mauconnpg14_01
IT-DA Maurizio De Giorgi

☑ Monitoring alerts

Description of the instance
Prod like instance for benchmarking connection scalability with pg14

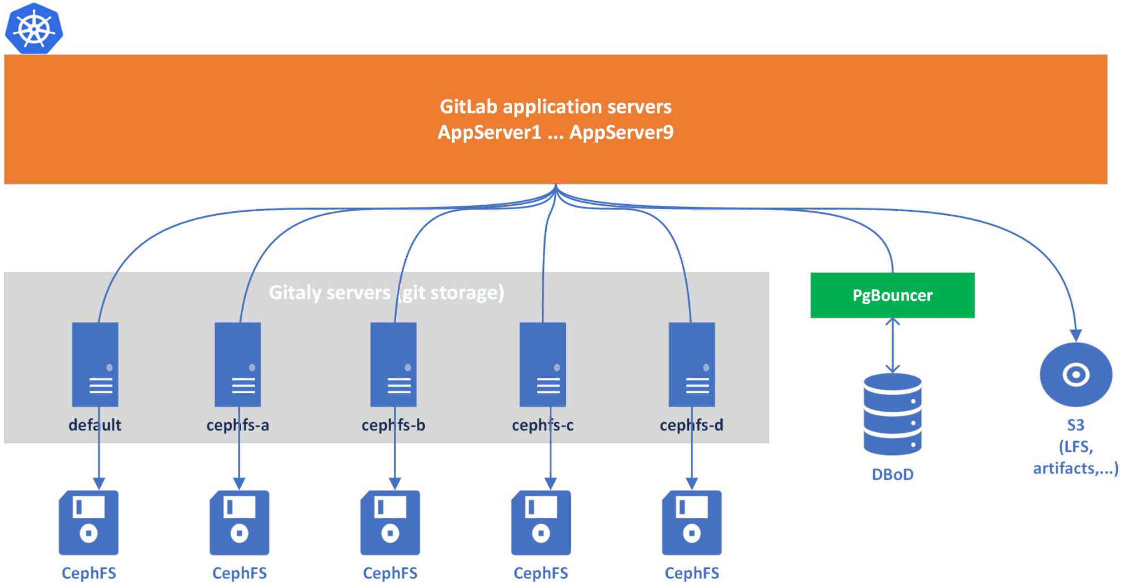
Owner [redacted]	E-group [redacted]	Project DBOD	Type PG
Version 14.10	Category TEST	Charge Group none	Port [redacted]
Host [redacted]	Creation date 17/03/2023	Expiry date 08/10/2024	

June 2024

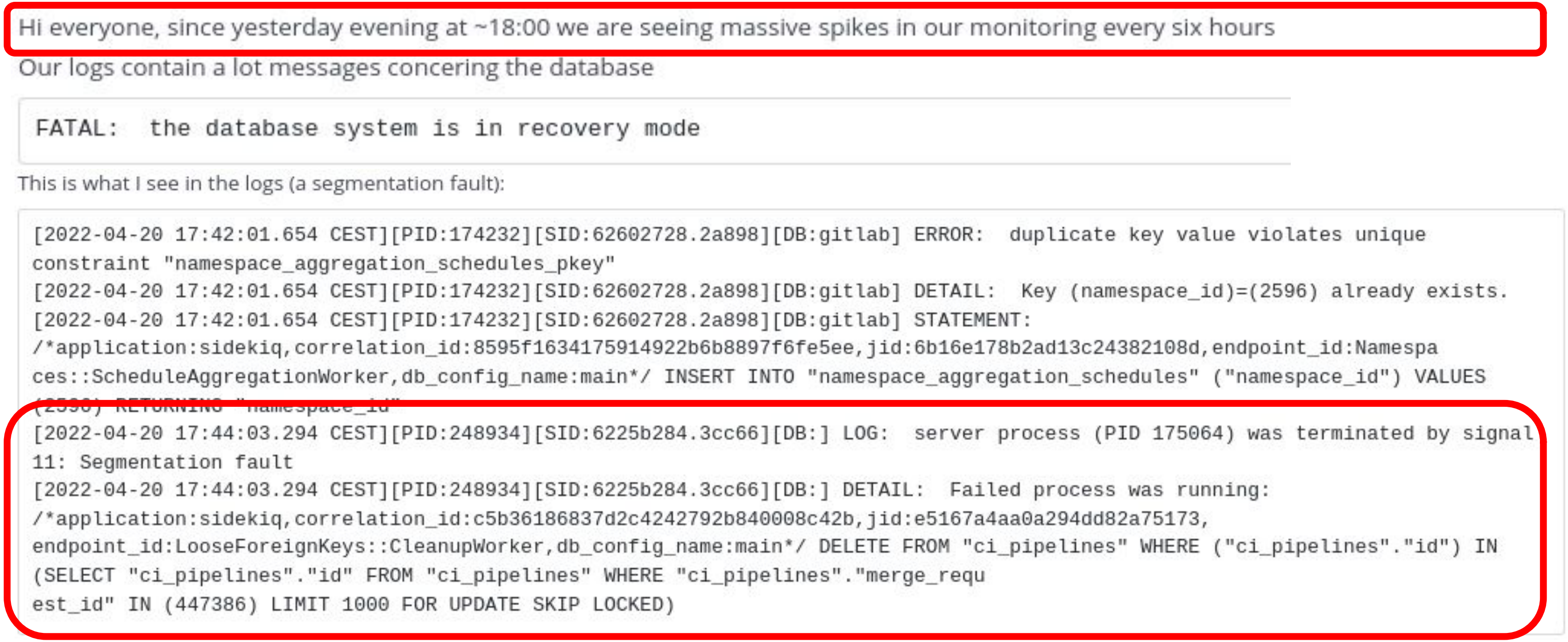
Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
26	5 27	4 28	1 29	2 30	1 31	1 1
1 2	1 3	1 4	1 5	1 6	1 7	1 8
1 9	1 10	1 11	1 12	1 13	1 14	1 15
1 16	1 17	18	19	20	21	22

GitLab at CERN

- GitLab is considered an **important piece of the ecosystem** at CERN
- Cloud Hybrid architecture, using the Helm deployment since 2022 (was Omnibus).
  - DBoD for databases
  - CephFS for storage
  - S3 for buckets
- Composed of:
  - **~150k projects.**
  - **19k users.**
  - **~320k pipelines/month.**
  - **Collaborators from all over the world**
- Almost all the software running our complex infrastructure it is hosted on GitLab



Let’s start from the beginning…



Hi everyone, since yesterday evening at ~18:00 we are seeing massive spikes in our monitoring every six hours

Our logs contain a lot messages concering the database

```
FATAL:  the database system is in recovery mode
```

This is what I see in the logs (a segmentation fault):

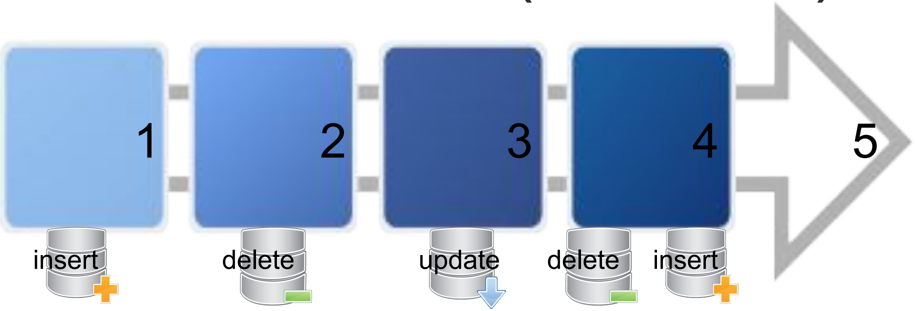
```
[2022-04-20 17:42:01.654 CEST][PID:174232][SID:62602728.2a898][DB:gitlab] ERROR:  duplicate key value violates unique
constraint "namespace_aggregation_schedules_pkey"
[2022-04-20 17:42:01.654 CEST][PID:174232][SID:62602728.2a898][DB:gitlab] DETAIL:  Key (namespace_id)=(2596) already exists.
[2022-04-20 17:42:01.654 CEST][PID:174232][SID:62602728.2a898][DB:gitlab] STATEMENT:
/*application:sidekiq,correlation_id:8595f1634175914922b6b8897f6fe5ee,jid:6b16e178b2ad13c24382108d,endpoint_id:Namespa
ces::ScheduleAggregationWorker,db_config_name:main*/ INSERT INTO "namespace_aggregation_schedules" ("namespace_id") VALUES
(2596) RETURNING "namespace_id"
[2022-04-20 17:44:03.294 CEST][PID:248934][SID:6225b284.3cc66][DB:] LOG:  server process (PID 175064) was terminated by signal
11: Segmentation fault
[2022-04-20 17:44:03.294 CEST][PID:248934][SID:6225b284.3cc66][DB:] DETAIL:  Failed process was running:
/*application:sidekiq,correlation_id:c5b36186837d2c4242792b840008c42b,jid:e5167a4aa0a294dd82a75173,
endpoint_id:LooseForeignKeys::CleanupWorker,db_config_name:main*/ DELETE FROM "ci_pipelines" WHERE ("ci_pipelines"."id") IN
(SELECT "ci_pipelines"."id" FROM "ci_pipelines" WHERE "ci_pipelines"."merge_requ
est_id" IN (447386) LIMIT 1000 FOR UPDATE SKIP LOCKED)
```

Agenda

- **An MVCC primer (boring things everyone knows but it is worth refreshing)**
  - (ACID) Transactions, Isolation, Concurrency, Serializable Snapshot Isolation
- **Connection scalability (showing the problem and its causes)**
  - Benchmarking & bottleneck analysis
- **Troubleshooting GitLab issues (talking about that time when we all had a lot of fun)**
  - The journey to enlightenment
  - The joy of enlightenment
- **The great effects of connection pooling on connection scalability**

An ACID Transaction

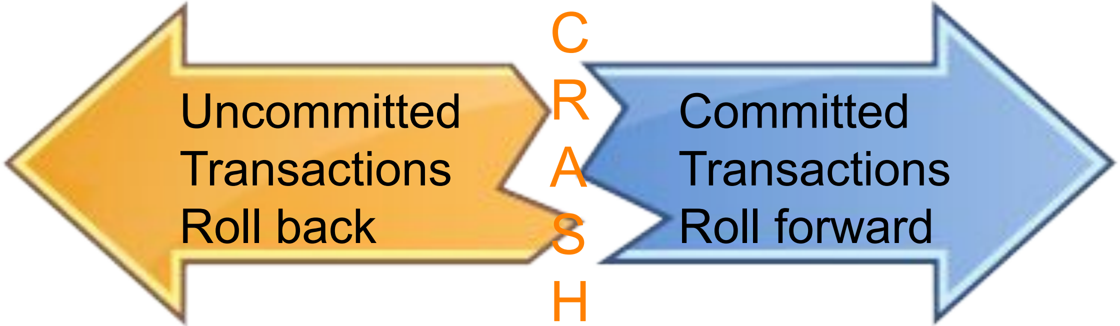
A set of operations that transfers a database from one correct state to another correct state (*Consistency*), provided they are all completed or rolled back (*Atomicity*) without interference from other transactions (*Isolation*)



Credits https://postgrespro.com/blog/pgsql/5967856

An ACID Transaction

Committed transactions must be durable, and withstand a system crash, without being affected by uncommitted transactions, the effects of which, should be rolled back as if they never happened (*Durability*)


Uncommitted Transactions Roll back
CRASH
Committed Transactions Roll forward

Credits https://postgrespro.com/blog/pgsql/5967856

Transactions and Concurrency

What is the fundamental problem?

Providing *concurrent* data access and transaction *isolation* for each database *session*, with reasonable *performance* in a *multi user* environments, while *minimizing lock* contention, so that *reading never blocks writing and writing never blocks reading*

Credits https://www.postgresql.org/docs/current/mvcc-intro.html

Transactions and Concurrency

What is the more commonly used solution for RDBMS?

Multi Version Concurrency Control

“Instead of updating data objects in-place[1],

each update creates a new version of that data object,

such that concurrent readers can still see the old version

while the update transaction proceeds concurrently[2]”

[1] ...and store before images in rollback segments like oracle does

[2] Credits Tobias Mühlbauer https://db.in.tum.de/~muehlbau/papers/mvcc.pdf

Multi Version Concurrency Control

How does it work?

It relies on Serializable Snapshot Isolation[1,2]

Each SQL statement sees a snapshot of data (a database version) as it was some time ago, regardless of the current state of the underlying data, and consisting only of changes committed before it was created

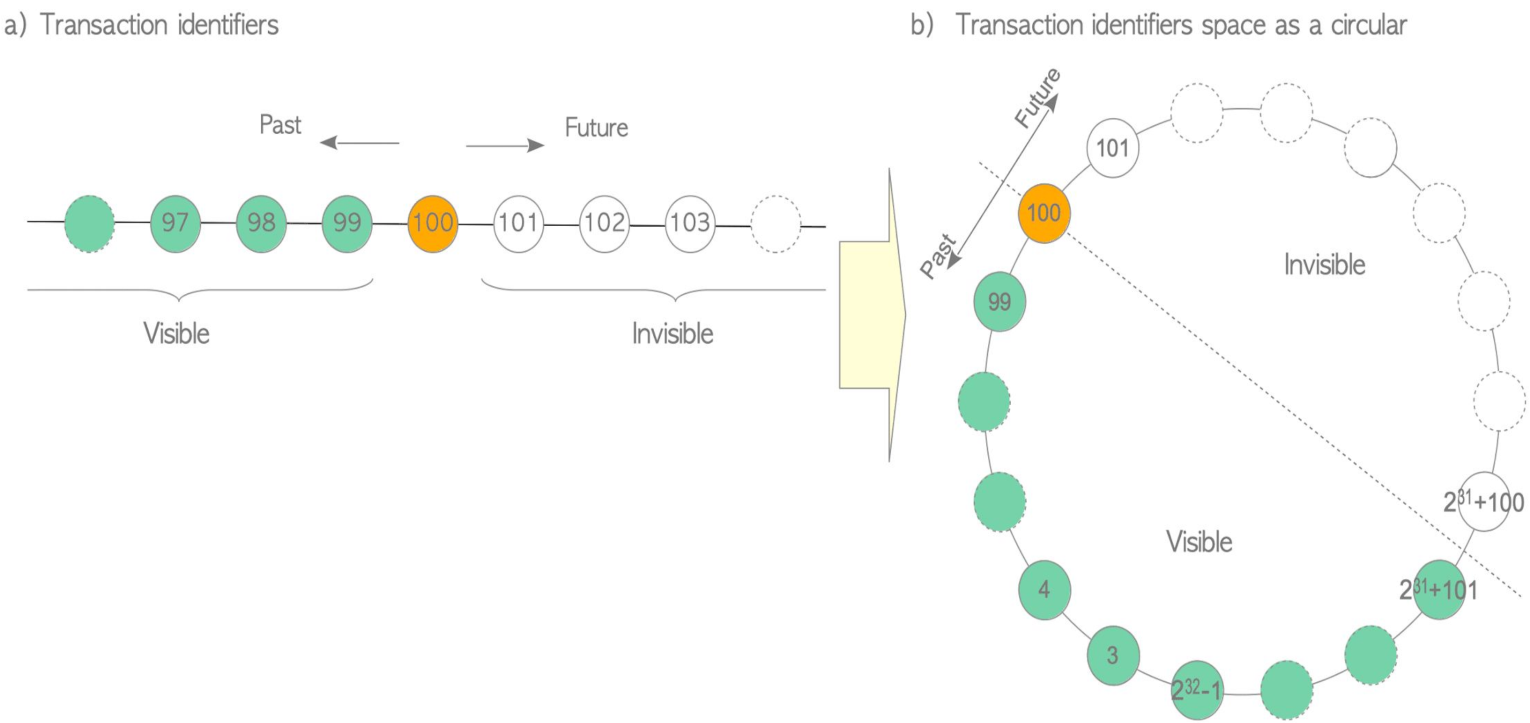
[1] 2009, Cahill, Michael James http://hdl.handle.net/2123/5353
[2] 2012, Dan R. K. Ports, Kevin Grittner https://arxiv.org/pdf/1208.4179.pdf

Multi Version Concurrency Control

“All queries in PostgreSQL are performed with respect to a snapshot, which is represented as the set of transactions whose effects are visible in the snapshot. Each tuple is tagged with the transaction ID of the transaction that created it (xmin), and, if it has been deleted or replaced with a new version, the transaction that did so (xmax)”

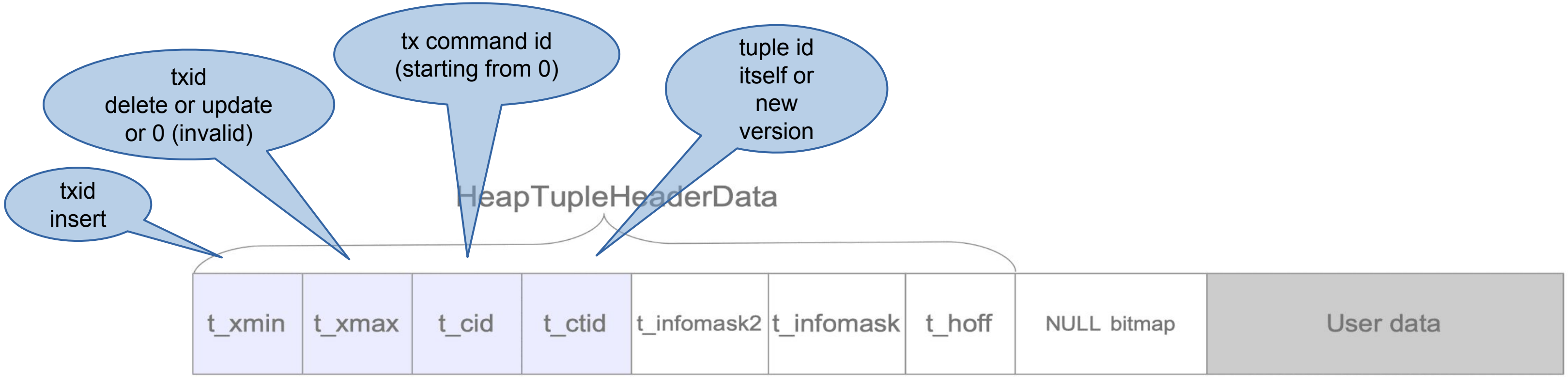
Credits Dan R. K. Ports, Kevin Grittner https://arxiv.org/pdf/1208.4179.pdf

Multi Version Concurrency Control



Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

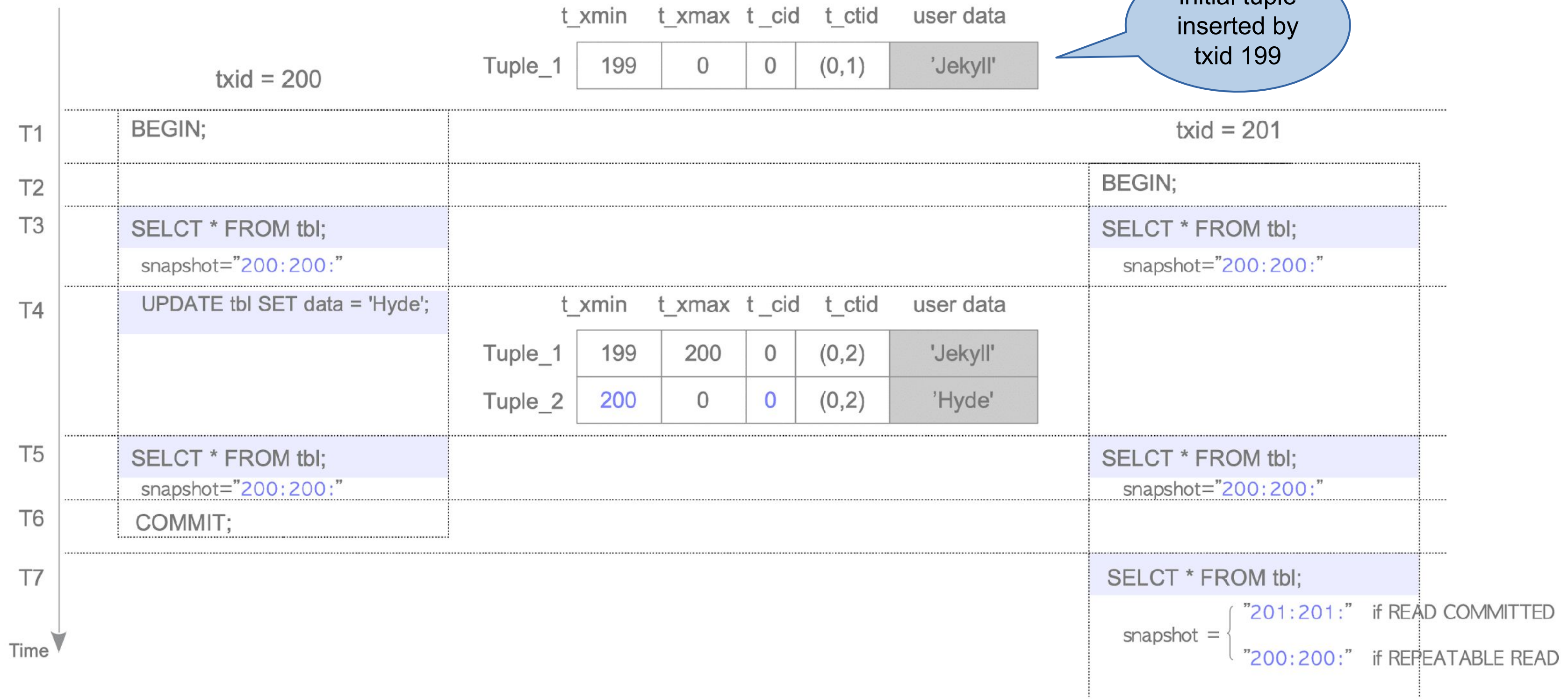
Multi Version Concurrency Control



Peeking relevant fields inside a heap tuple header

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t _cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

initial tuple inserted by txid 199

T1 (txid = 200): BEGIN;

T2 (txid = 201): BEGIN;

T3 (txid = 200): SELCT * FROM tbl;
snapshot="200:200:"

T3 (txid = 201): SELCT * FROM tbl;
snapshot="200:200:"

T4 (txid = 200): UPDATE tbl SET data = 'Hyde';

	t_xmin	t_xmax	t _cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 (txid = 200): SELCT * FROM tbl;
snapshot="200:200:"

T5 (txid = 201): SELCT * FROM tbl;
snapshot="200:200:"

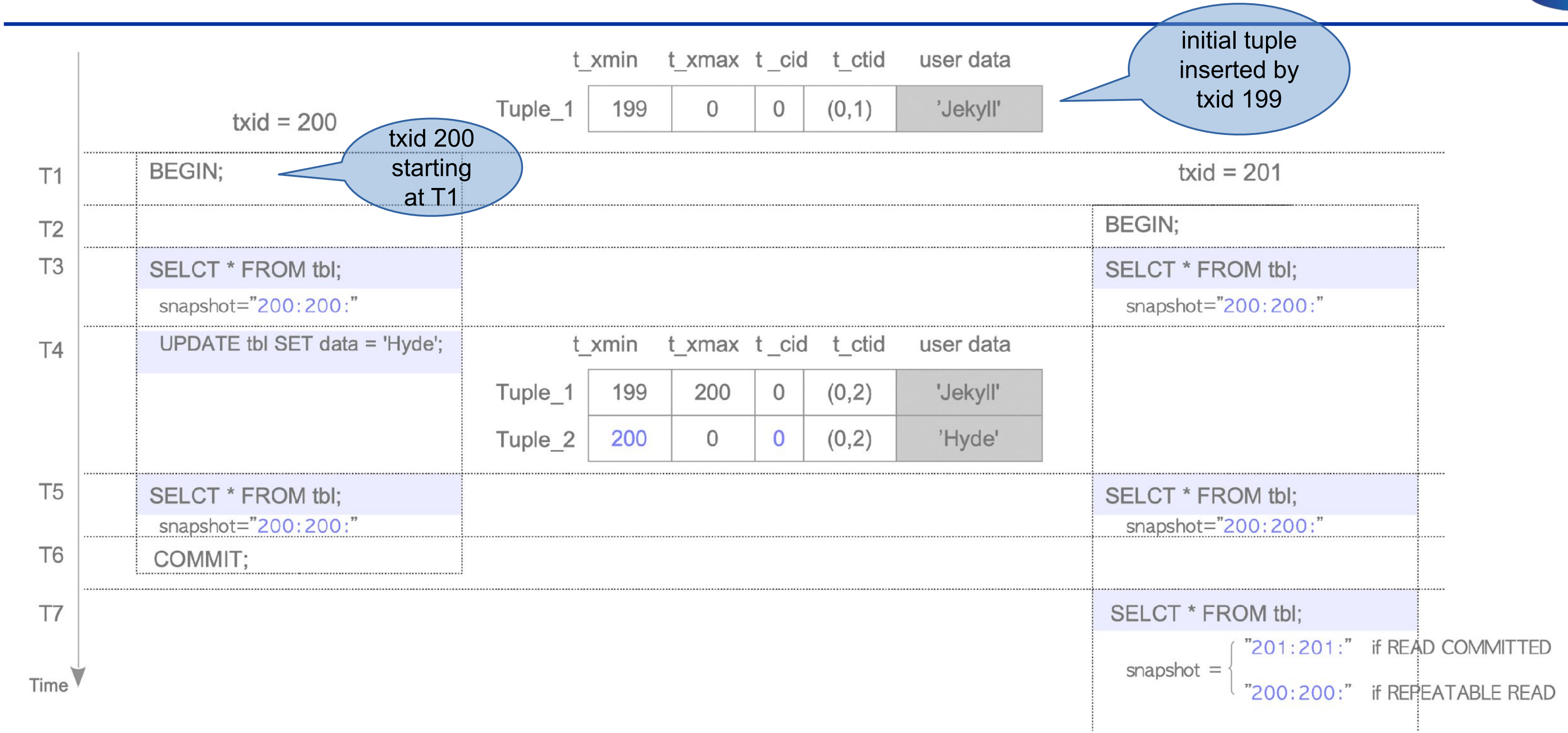
T6 (txid = 200): COMMIT;

T7 (txid = 201): SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED
"200:200:" if REPEATABLE READ

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control

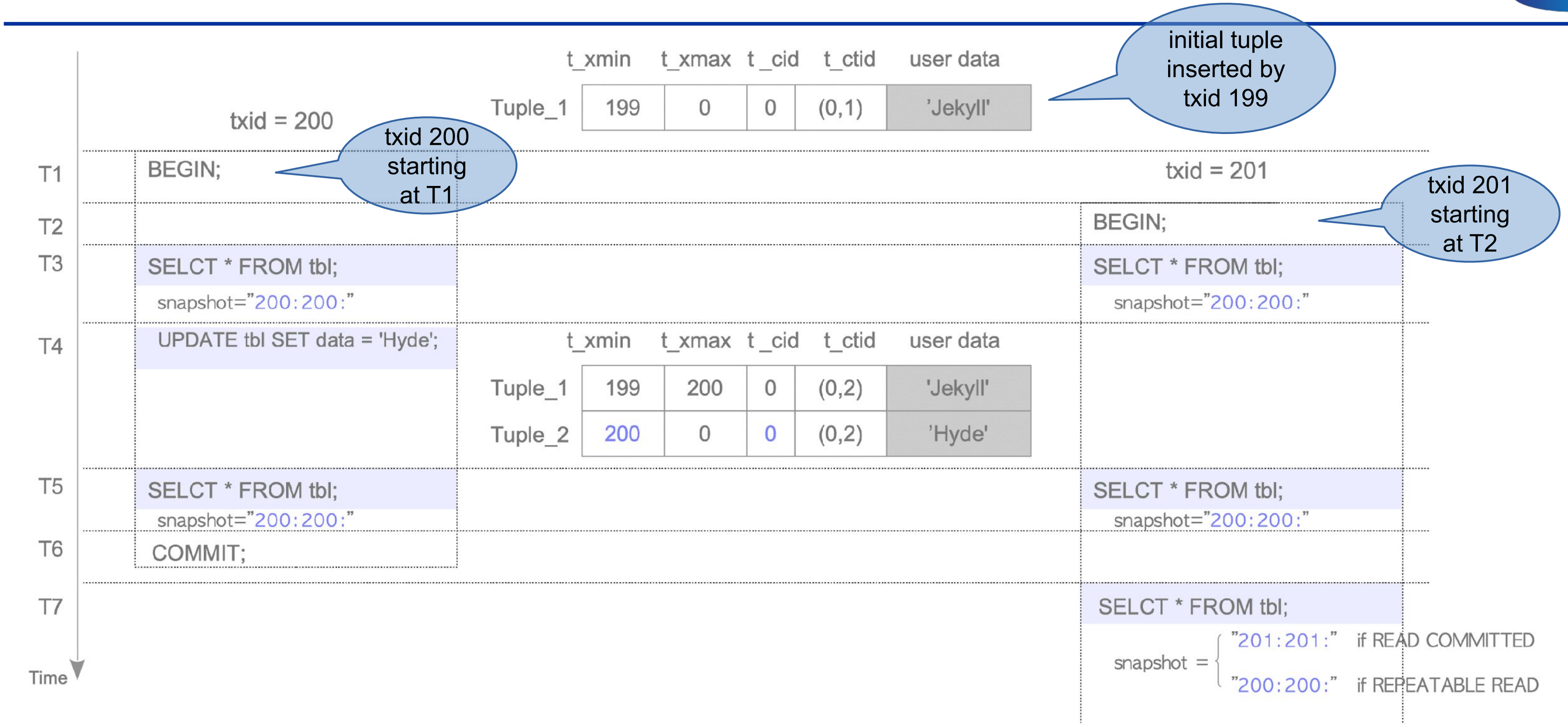


	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	’Jekyll'

initial tuple inserted by txid 199

txid = 200

T1 BEGIN;

txid 200 starting at T1

txid = 201

T2 BEGIN;

txid 201 starting at T2

T3 SELCT * FROM tbl;
snapshot=”200:200:”

SELCT * FROM tbl;
snapshot=”200:200:”

T4 UPDATE tbl SET data = 'Hyde';

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 SELCT * FROM tbl;
snapshot=”200:200:”

SELCT * FROM tbl;
snapshot=”200:200:”

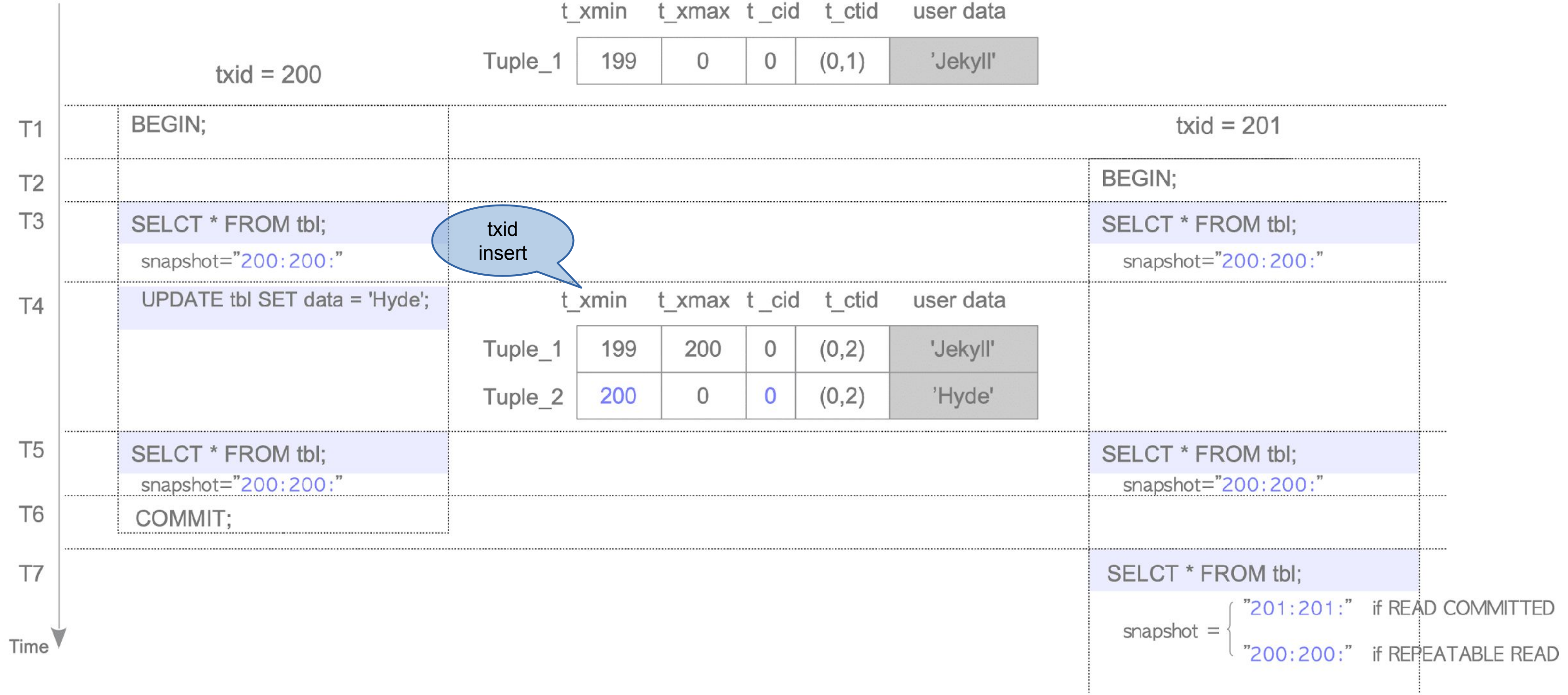
T6 COMMIT;

T7 SELCT * FROM tbl;

snapshot = ”201:201:” if READ COMMITTED
”200:200:” if REPEATABLE READ

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



txid = 200

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	’Jekyll'

T1 BEGIN;

txid = 201

T2 BEGIN;

T3 SELCT * FROM tbl;
snapshot=”200:200:”

SELCT * FROM tbl;
snapshot=”200:200:”

txid
insert

T4 UPDATE tbl SET data = 'Hyde';

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	’Jekyll'
Tuple_2	200	0	0	(0,2)	’Hyde'

T5 SELCT * FROM tbl;
snapshot=”200:200:”

SELCT * FROM tbl;
snapshot=”200:200:”

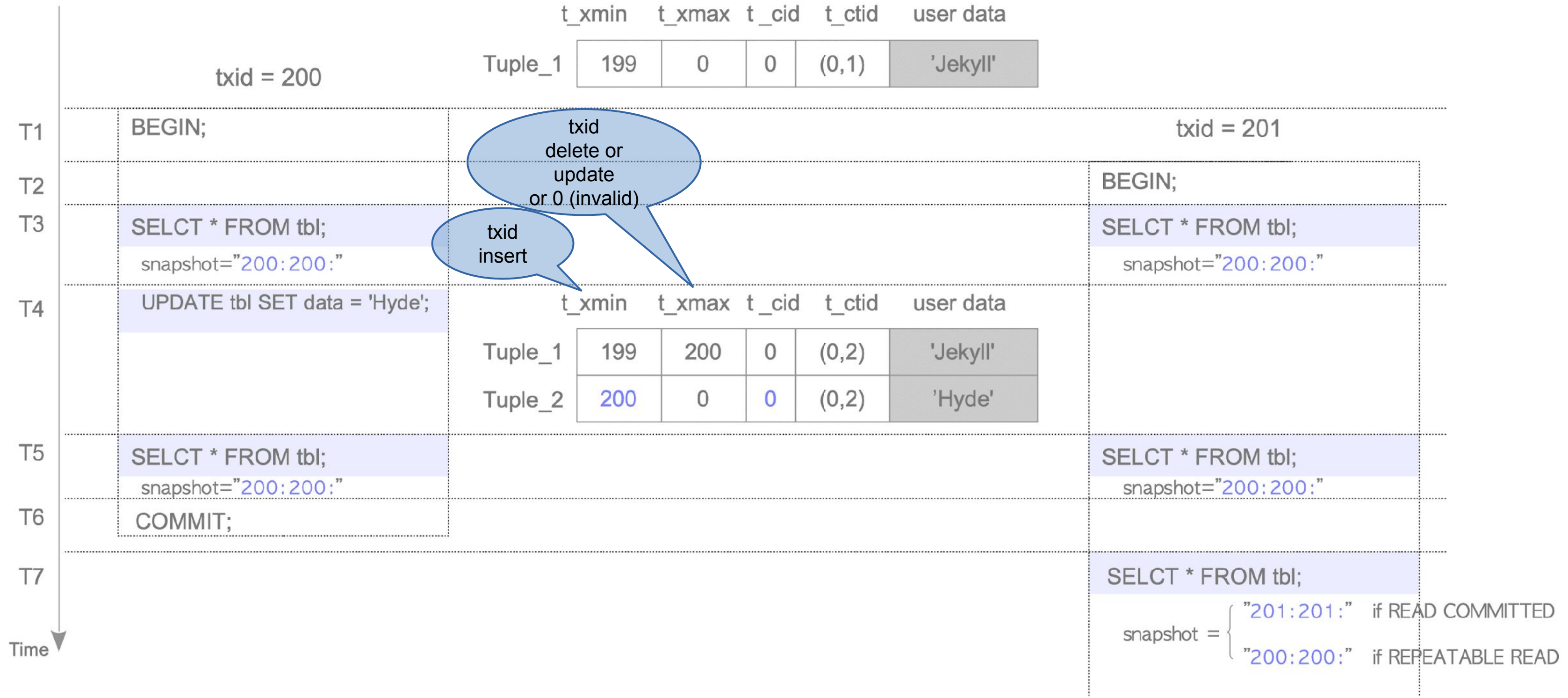
T6 COMMIT;

T7 SELCT * FROM tbl;

snapshot = ”201:201:” if READ COMMITTED
”200:200:” if REPEATABLE READ

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

txid = 200

T1 BEGIN;

T2

T3 SELCT * FROM tbl;

snapshot="200:200:"

T4 UPDATE tbl SET data = 'Hyde';

T5 SELCT * FROM tbl;

snapshot="200:200:"

T6 COMMIT;

T7

Time

txid = 201

BEGIN;

SELCT * FROM tbl;

snapshot="200:200:"

SELCT * FROM tbl;

snapshot="200:200:"

SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED
"200:200:" if REPEATABLE READ

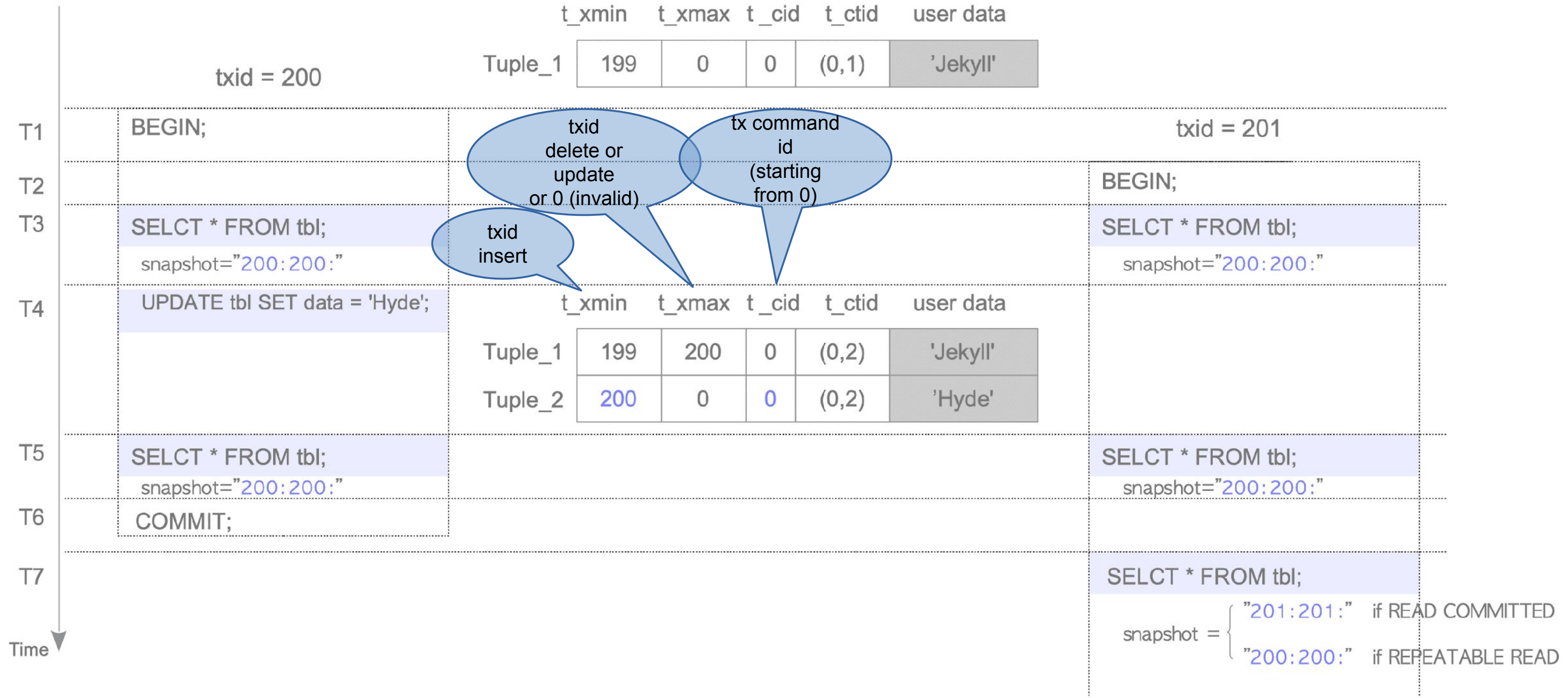
txid delete or update or 0 (invalid)

txid insert

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

txid = 200

txid = 201

T1 BEGIN;

T2 BEGIN;

T3 SELCT * FROM tbl;
snapshot="200:200:"

T3 SELCT * FROM tbl;
snapshot="200:200:"

T4 UPDATE tbl SET data = 'Hyde';

txid delete or update or 0 (invalid)

tx command id (starting from 0)

txid insert

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 SELCT * FROM tbl;
snapshot="200:200:"

T5 SELCT * FROM tbl;
snapshot="200:200:"

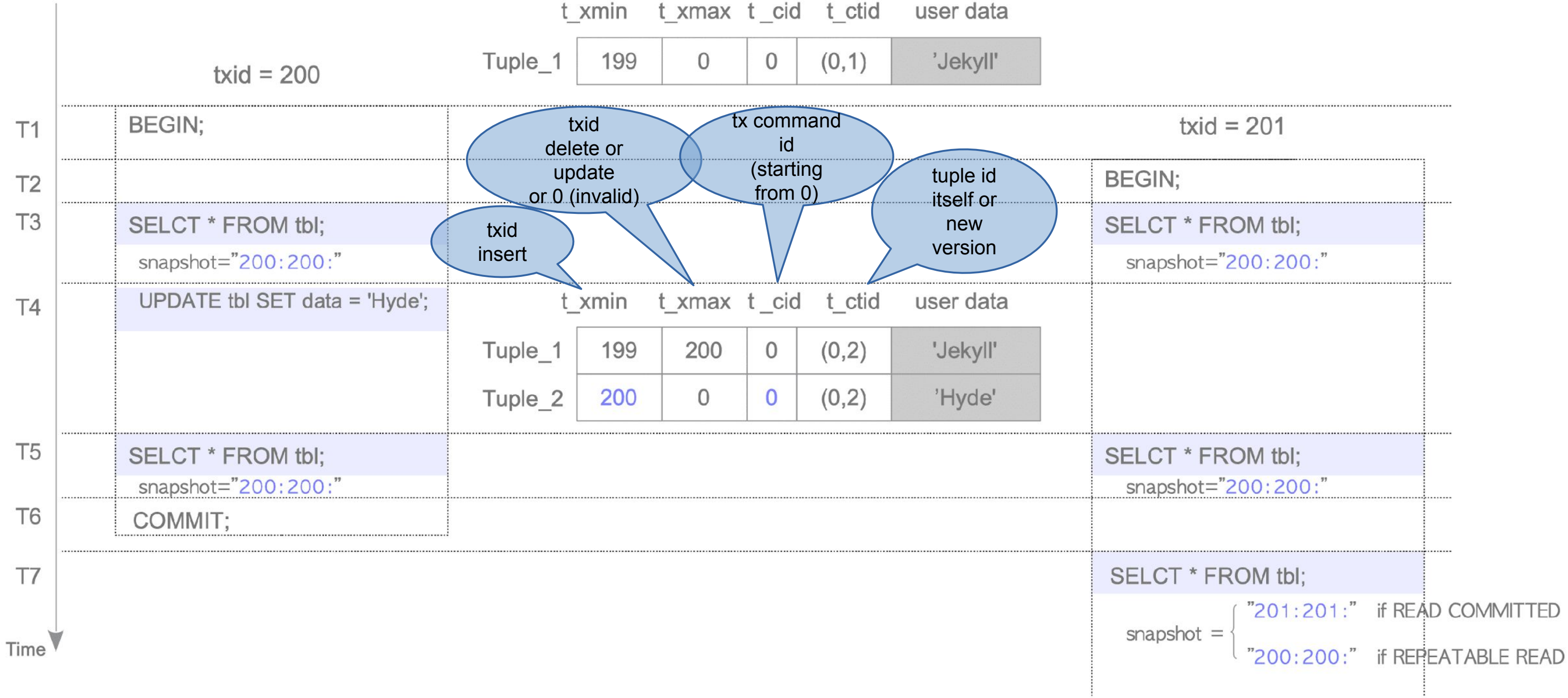
T6 COMMIT;

T7 SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED
"200:200:" if REPEATABLE READ

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



txid = 200

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

txid = 201

T1 BEGIN;

T2 BEGIN;

T3 SELCT * FROM tbl;
snapshot="200:200:"

SELCT * FROM tbl;
snapshot="200:200:"

T4 UPDATE tbl SET data = 'Hyde';

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 SELCT * FROM tbl;
snapshot="200:200:"

SELCT * FROM tbl;
snapshot="200:200:"

T6 COMMIT;

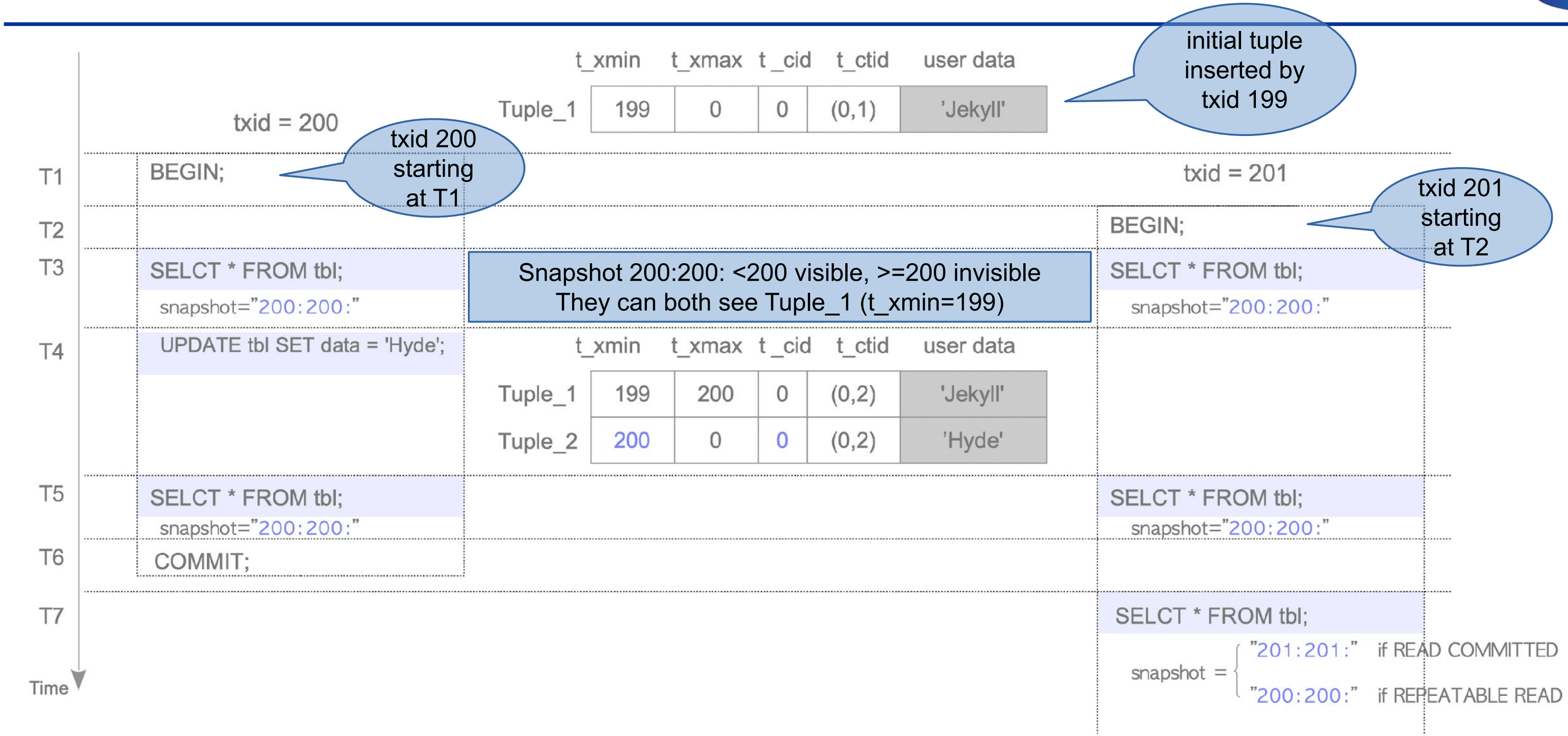
T7 SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED
"200:200:" if REPEATABLE READ

txid delete or update or 0 (invalid)
tx command id (starting from 0)
tuple id itself or new version
txid insert
Time

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

initial tuple inserted by txid 199

txid = 200

txid 200 starting at T1

T1 BEGIN;

txid = 201

T2 BEGIN;

txid 201 starting at T2

T3 SELCT * FROM tbl;
snapshot="200:200:"

Snapshot 200:200: <200 visible, >=200 invisible
They can both see Tuple_1 (t_xmin=199)

SELCT * FROM tbl;
snapshot="200:200:"

T4 UPDATE tbl SET data = 'Hyde';

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 SELCT * FROM tbl;
snapshot="200:200:"

SELCT * FROM tbl;
snapshot="200:200:"

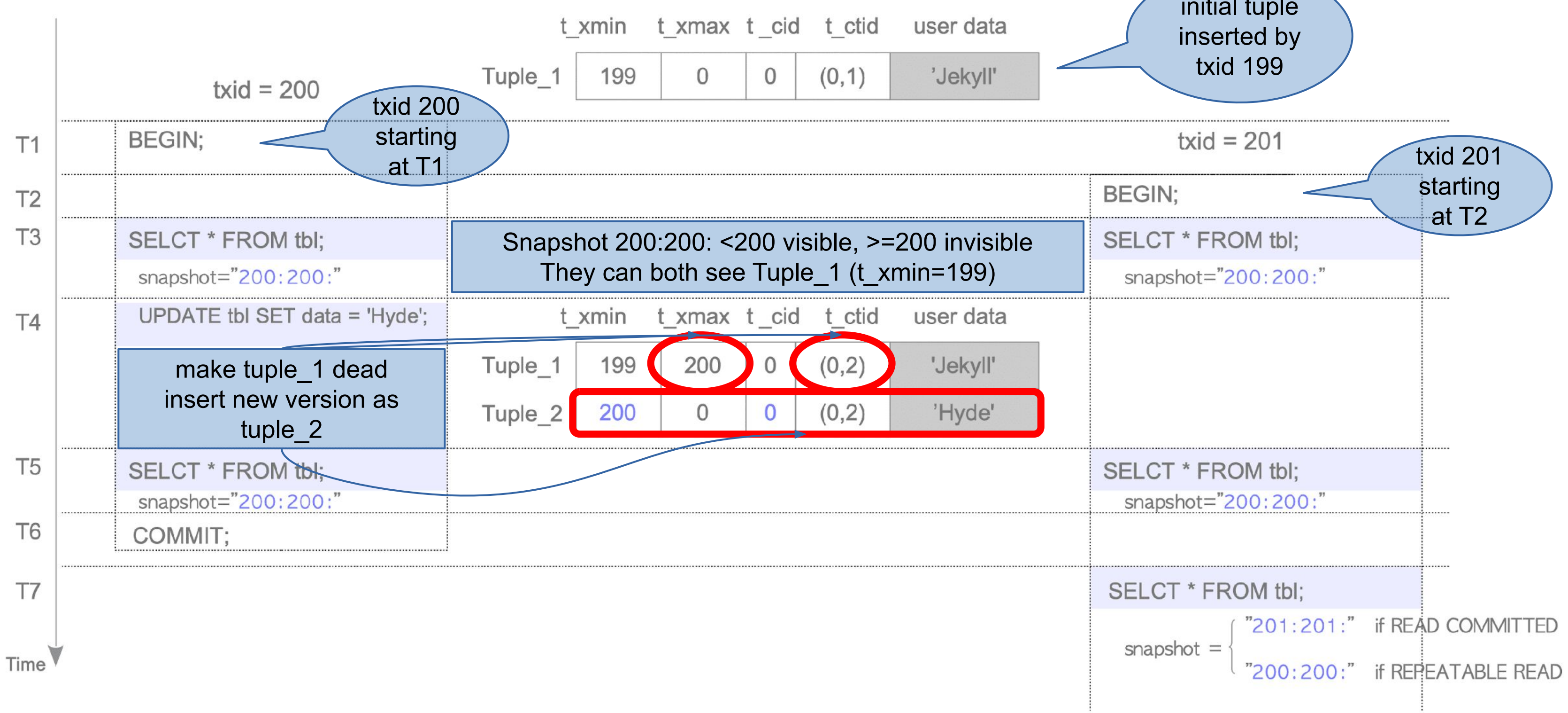
T6 COMMIT;

T7 SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED
"200:200:" if REPEATABLE READ

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

initial tuple inserted by txid 199

txid = 200

T1 BEGIN;

txid 200 starting at T1

txid = 201

T2 BEGIN;

txid 201 starting at T2

T3 SELCT * FROM tbl;
snapshot="200:200:"

Snapshot 200:200: <200 visible, >=200 invisible
They can both see Tuple_1 (t_xmin=199)

SELCT * FROM tbl;
snapshot="200:200:"

T4 UPDATE tbl SET data = 'Hyde';

make tuple_1 dead
insert new version as tuple_2

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 SELCT * FROM tbl;
snapshot="200:200:"

SELCT * FROM tbl;
snapshot="200:200:"

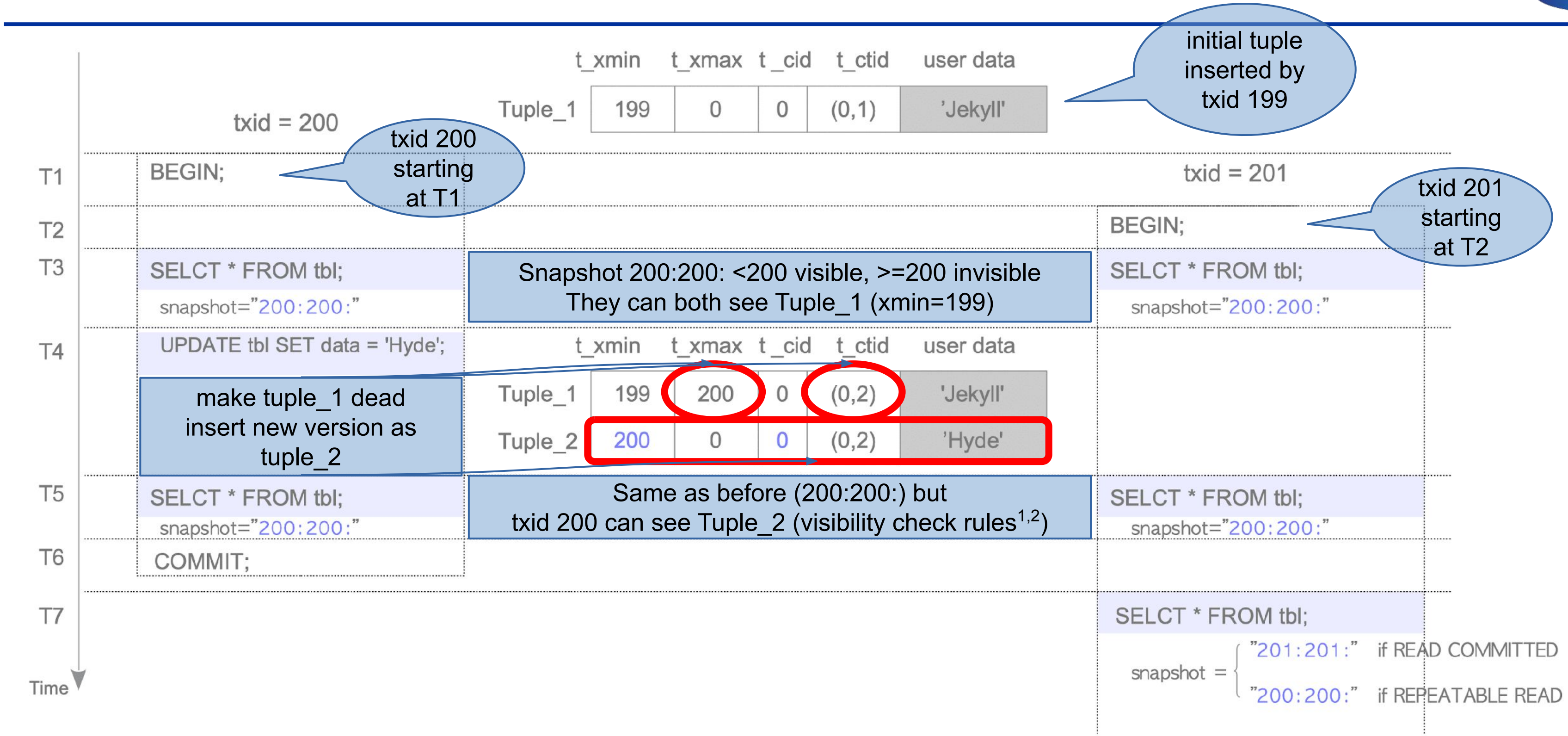
T6 COMMIT;

T7 SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED
"200:200:" if REPEATABLE READ

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

initial tuple inserted by txid 199

txid = 200

txid 200 starting at T1

txid = 201

txid 201 starting at T2

T1 BEGIN;

T2 BEGIN;

T3 SELCT * FROM tbl;
snapshot="200:200:"

Snapshot 200:200: <200 visible, >=200 invisible
They can both see Tuple_1 (xmin=199)

SELCT * FROM tbl;
snapshot="200:200:"

T4 UPDATE tbl SET data = 'Hyde';

make tuple_1 dead
insert new version as tuple_2

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 SELCT * FROM tbl;
snapshot="200:200:"

Same as before (200:200:) but
txid 200 can see Tuple_2 (visibility check rules[1,2])

SELCT * FROM tbl;
snapshot="200:200:"

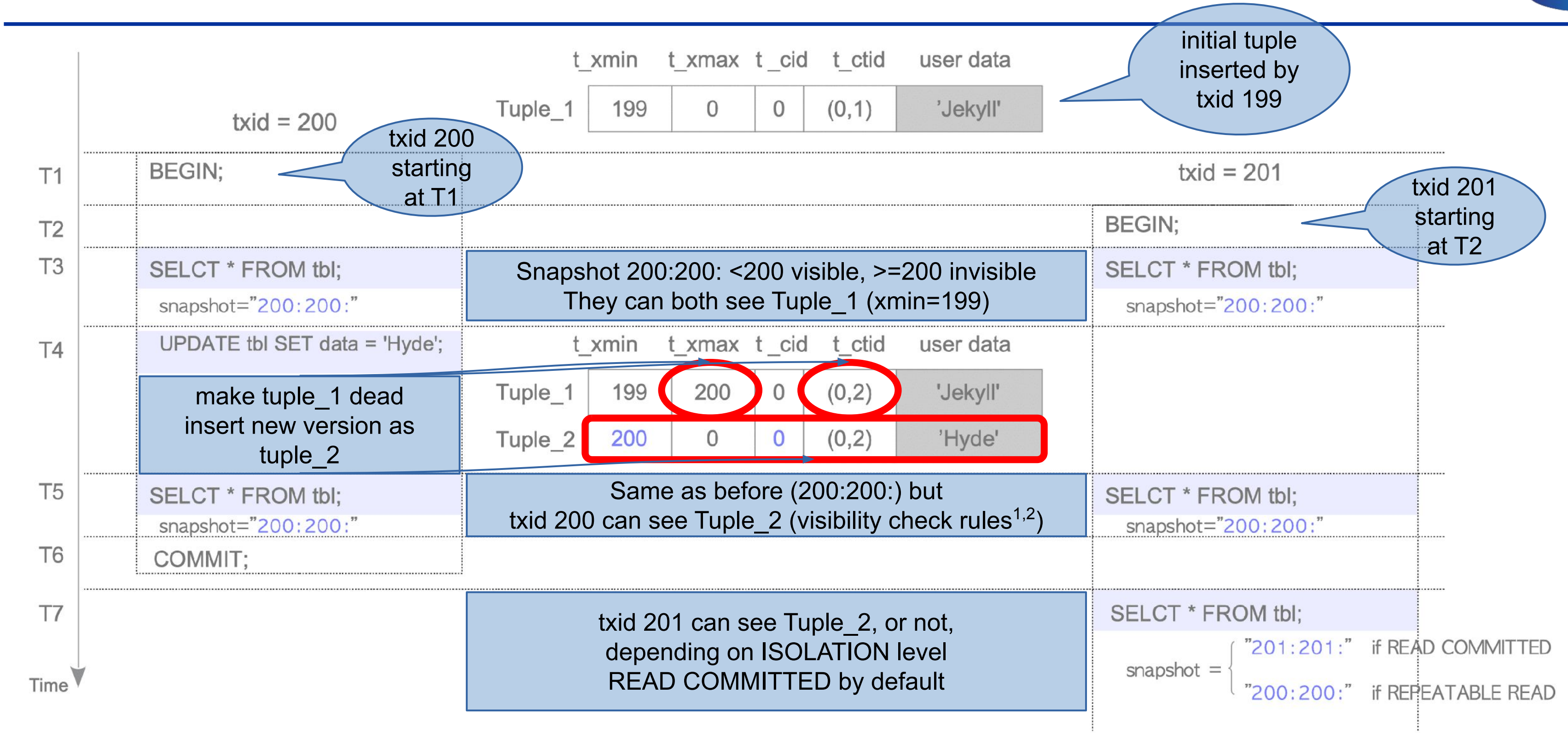
T6 COMMIT;

T7 SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED
"200:200:" if REPEATABLE READ

[1] https://www.interdb.jp/pg/pgsql05/06.html
[2] https://www.interdb.jp/pg/pgsql05/07.html

Multi Version Concurrency Control



	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	0	0	(0,1)	'Jekyll'

initial tuple inserted by txid 199

txid = 200

txid 200 starting at T1

T1 BEGIN;

txid = 201

txid 201 starting at T2

T2 BEGIN;

T3 SELCT * FROM tbl;

snapshot="200:200:"

Snapshot 200:200: <200 visible, >=200 invisible They can both see Tuple_1 (xmin=199)

SELCT * FROM tbl;

snapshot="200:200:"

T4 UPDATE tbl SET data = 'Hyde';

make tuple_1 dead insert new version as tuple_2

	t_xmin	t_xmax	t_cid	t_ctid	user data
Tuple_1	199	200	0	(0,2)	'Jekyll'
Tuple_2	200	0	0	(0,2)	'Hyde'

T5 SELCT * FROM tbl;

snapshot="200:200:"

Same as before (200:200:) but txid 200 can see Tuple_2 (visibility check rules[1,2])

SELCT * FROM tbl;

snapshot="200:200:"

T6 COMMIT;

T7

txid 201 can see Tuple_2, or not, depending on ISOLATION level READ COMMITTED by default

SELCT * FROM tbl;

snapshot = "201:201:" if READ COMMITTED

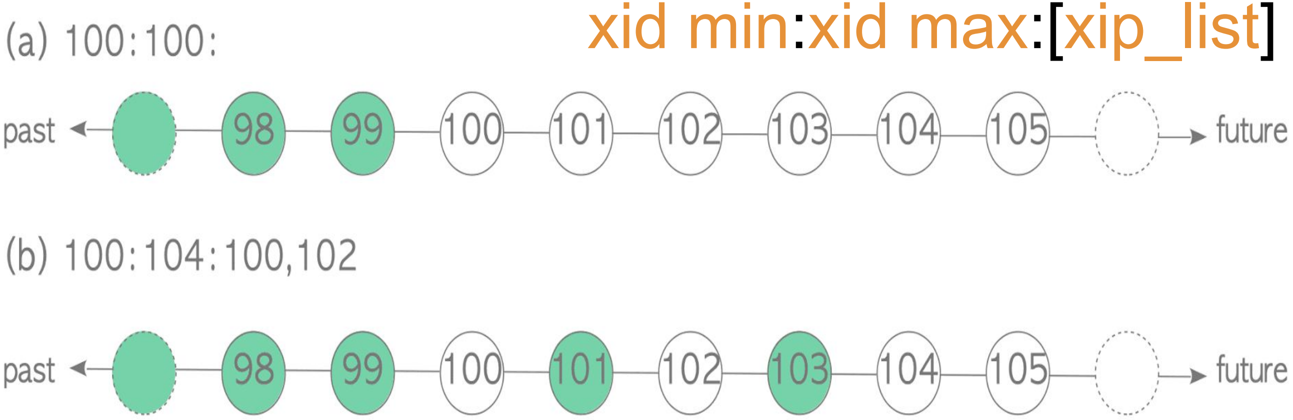
"200:200:" if REPEATABLE READ

[1] https://www.interdb.jp/pg/pgsql05/06.html

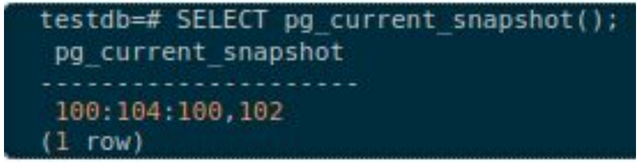
[2] https://www.interdb.jp/pg/pgsql05/07.html

Multi Version Concurrency Control

xid min:xid max:[xip_list]



ıen



```
testdb=# SELECT pg_current_snapshot();
 pg_current_snapshot
---------------------
 100:104:100,102
(1 row)
```

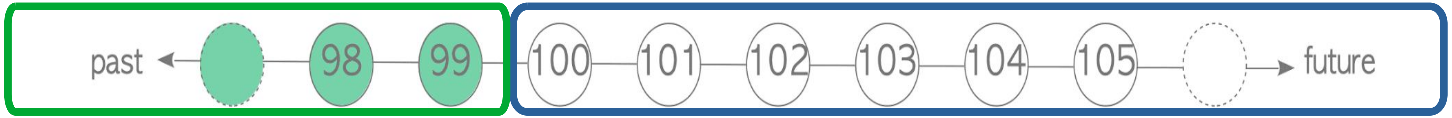
○ Active txid:it is in progress or is not yet started, and is invisible.

🟩 Inactive txid: it is committed or aborted, and is visible if committed.

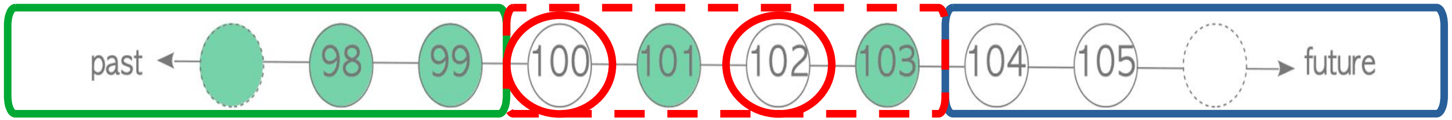
Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

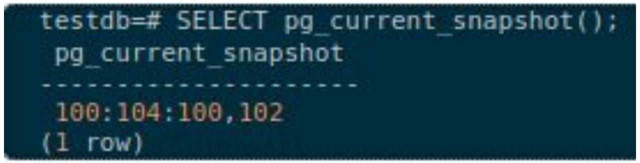
Multi Version Concurrency Control

xid min:xid max:[xip_list]



not active<xmin, active if in xmin<=xip_list<xmax, not yet started >=xmax





```
testdb=# SELECT pg_current_snapshot();
 pg_current_snapshot
---------------------
 100:104:100,102
(1 row)
```

○ Active txid:it is in progress or is not yet started, and is invisible.

● Inactive txid: it is committed or aborted, and is visible if committed.

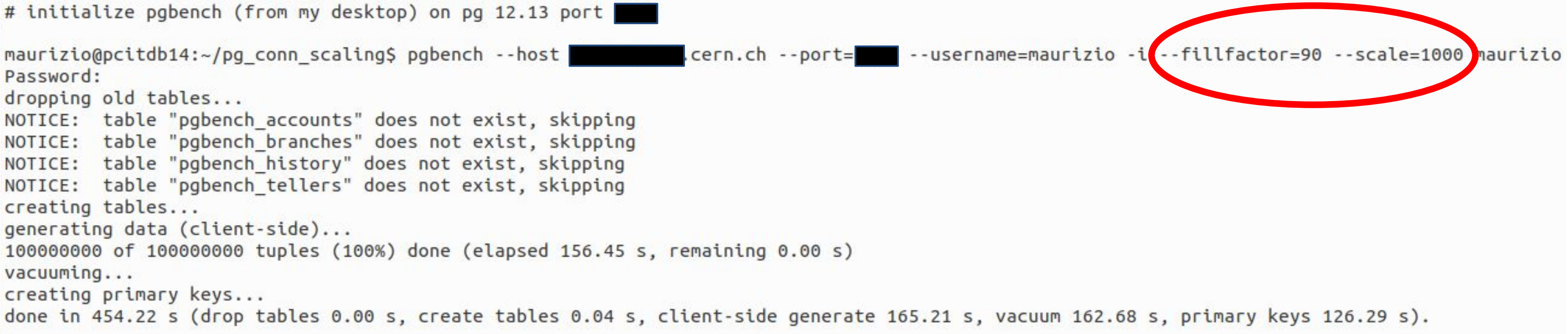
Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html

Agenda

- **An MVCC primer (boring things everyone knows but it is worth refreshing)**
  - (ACID) Transactions, Isolation, Concurrency, Serializable Snapshot Isolation
- **Connection scalability (showing the problem and its causes)**
  - Benchmarking & bottleneck analysis
- **Troubleshooting GitLab issues (talking about that time when we all had a lot of fun)**
  - The journey to enlightenment
  - The joy of enlightenment
- **The great effects of connection pooling on connection scalability**

Connections scalability

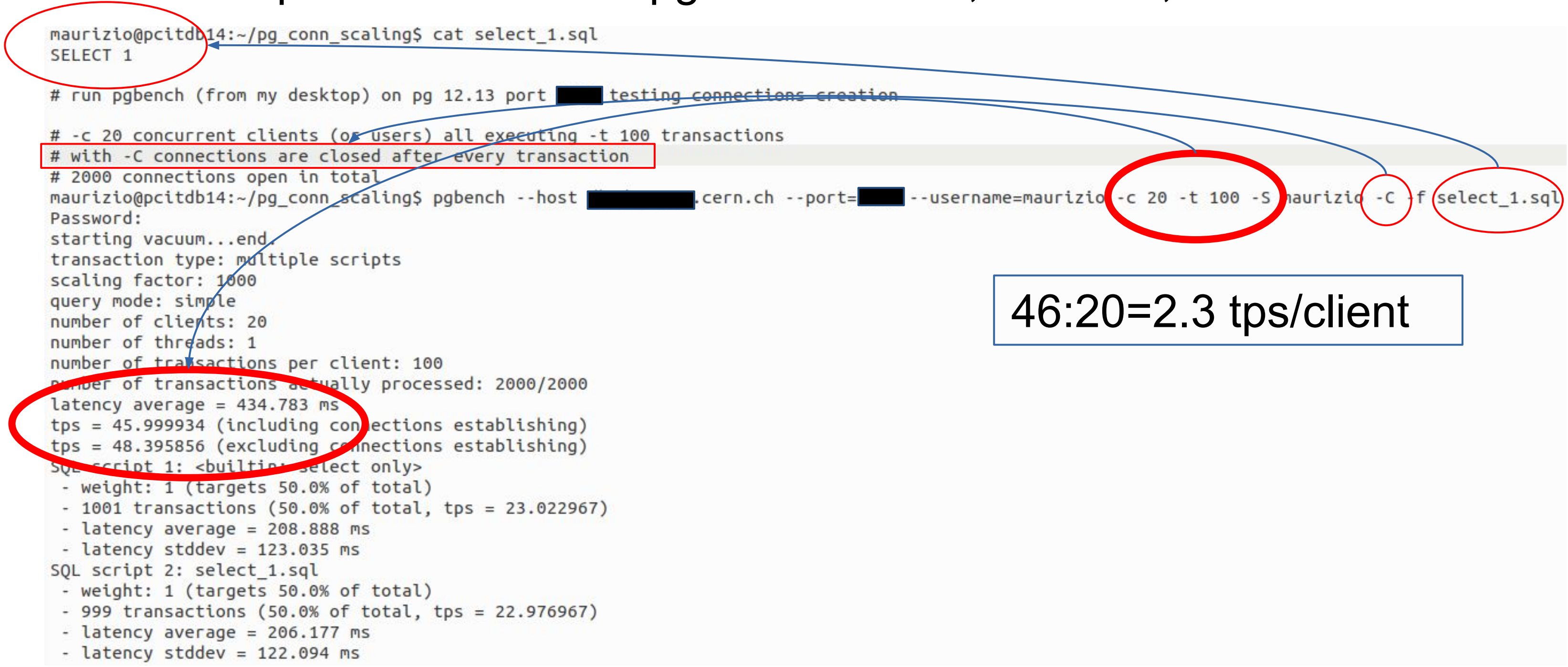
Initialize pgbench data set



```
# initialize pgbench (from my desktop) on pg 12.13 port [redacted]

maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host [redacted].cern.ch --port=[redacted] --username=maurizio -i --fillfactor=90 --scale=1000 maurizio
Password:
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000000 of 100000000 tuples (100%) done (elapsed 156.45 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 454.22 s (drop tables 0.00 s, create tables 0.04 s, client-side generate 165.21 s, vacuum 162.68 s, primary keys 126.29 s).
```

Connections scalability

A simple 1st run with 20 pgbench clients, 1 thread, 100 trx/client

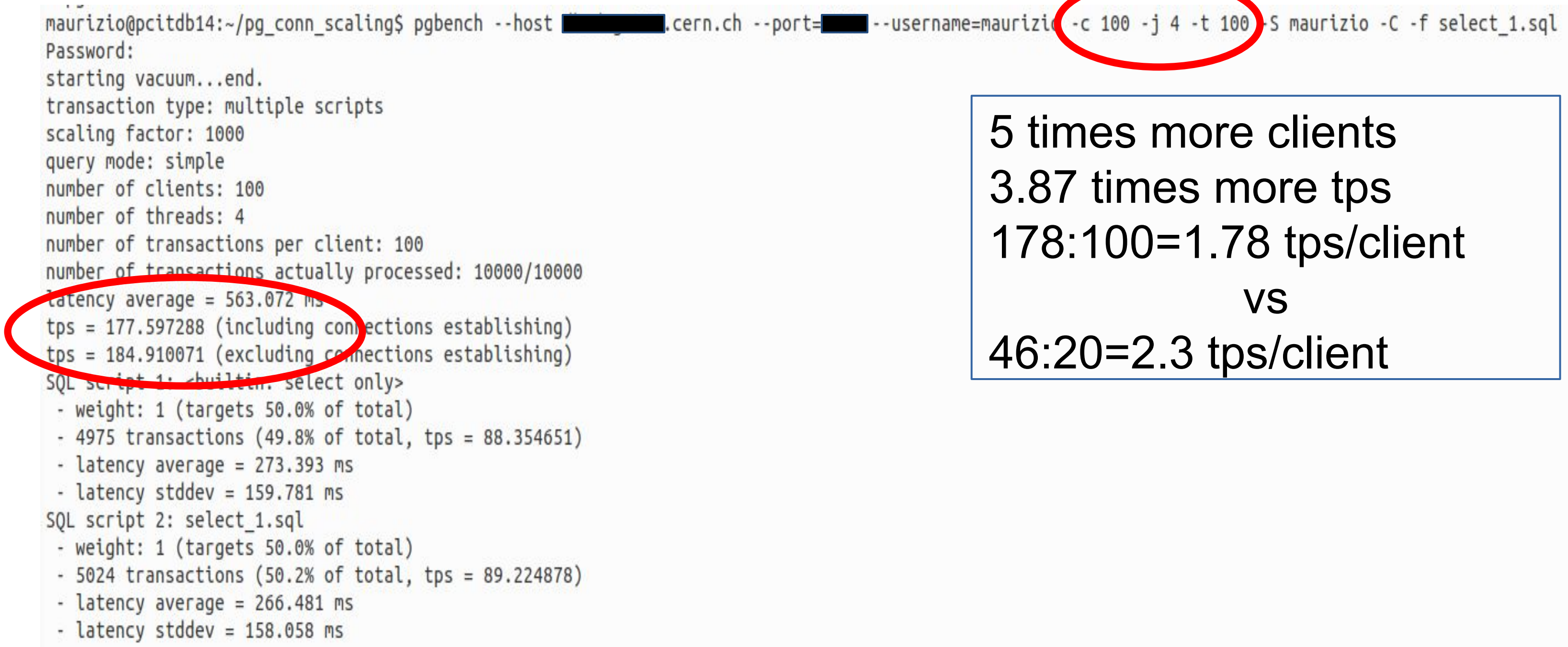


```
maurizio@pcitdb14:~/pg_conn_scaling$ cat select_1.sql
SELECT 1

# run pgbench (from my desktop) on pg 12.13 port [redacted] testing connections creation

# -c 20 concurrent clients (or users) all executing -t 100 transactions
# with -C connections are closed after every transaction
# 2000 connections open in total
maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host [redacted].cern.ch --port=[redacted] --username=maurizio -c 20 -t 100 -S maurizio -C -f select_1.sql
Password:
starting vacuum...end.
transaction type: multiple scripts
scaling factor: 1000
query mode: simple
number of clients: 20
number of threads: 1
number of transactions per client: 100
number of transactions actually processed: 2000/2000
latency average = 434.783 ms
tps = 45.999934 (including connections establishing)
tps = 48.395856 (excluding connections establishing)
SQL script 1: <builtin: select only>
 - weight: 1 (targets 50.0% of total)
 - 1001 transactions (50.0% of total, tps = 23.022967)
 - latency average = 208.888 ms
 - latency stddev = 123.035 ms
SQL script 2: select_1.sql
 - weight: 1 (targets 50.0% of total)
 - 999 transactions (50.0% of total, tps = 22.976967)
 - latency average = 206.177 ms
 - latency stddev = 122.094 ms
```

46:20=2.3 tps/client

Connections scalability

A simple 2nd run with 100 pgbench clients, 4 threads, 100 trx/client

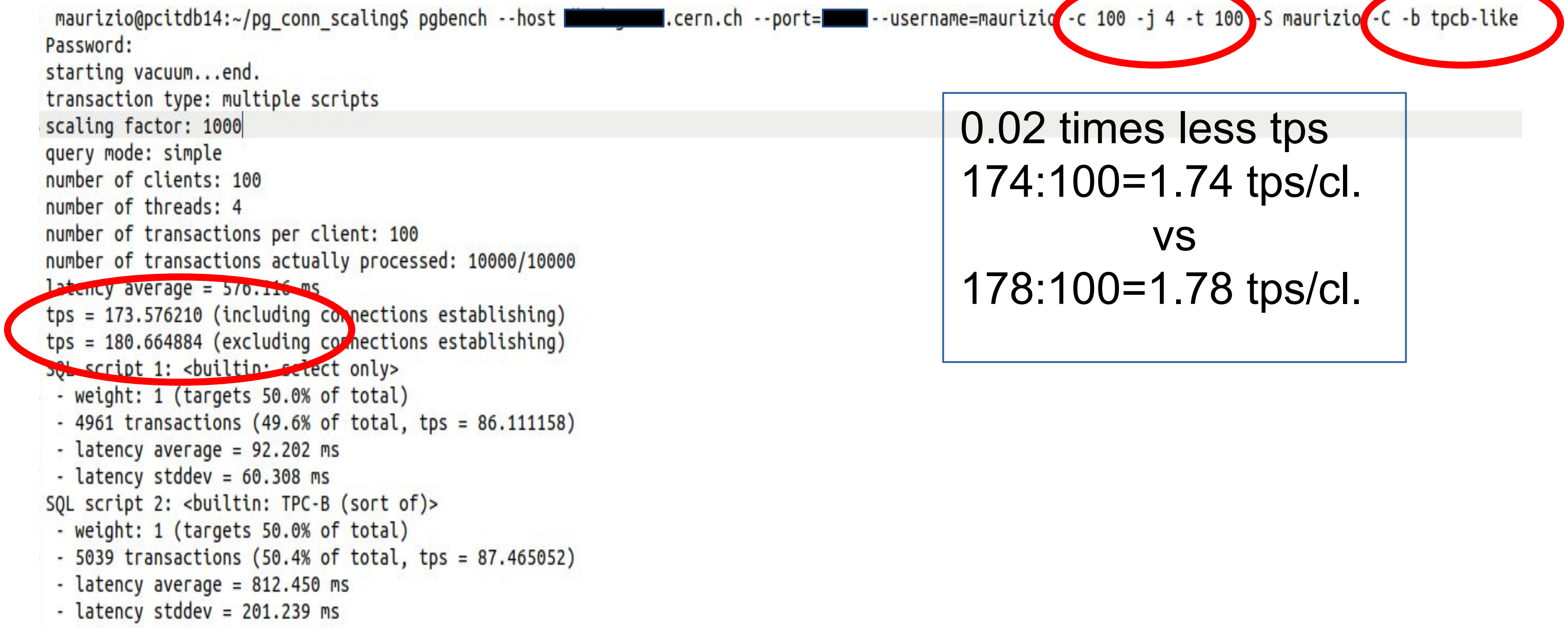


```
maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host [redacted].cern.ch --port=[redacted] --username=maurizio -c 100 -j 4 -t 100 -S maurizio -C -f select_1.sql
Password:
starting vacuum...end.
transaction type: multiple scripts
scaling factor: 1000
query mode: simple
number of clients: 100
number of threads: 4
number of transactions per client: 100
number of transactions actually processed: 10000/10000
latency average = 563.072 ms
tps = 177.597288 (including connections establishing)
tps = 184.910071 (excluding connections establishing)
SQL script 1: <builtin: select only>
 - weight: 1 (targets 50.0% of total)
 - 4975 transactions (49.8% of total, tps = 88.354651)
 - latency average = 273.393 ms
 - latency stddev = 159.781 ms
SQL script 2: select_1.sql
 - weight: 1 (targets 50.0% of total)
 - 5024 transactions (50.2% of total, tps = 89.224878)
 - latency average = 266.481 ms
 - latency stddev = 158.058 ms
```

5 times more clients
3.87 times more tps
178:100=1.78 tps/client
vs
46:20=2.3 tps/client

Connections scalability

A **tpcb-like** run with 100 pgbench clients, 4 threads, 100 trx/client

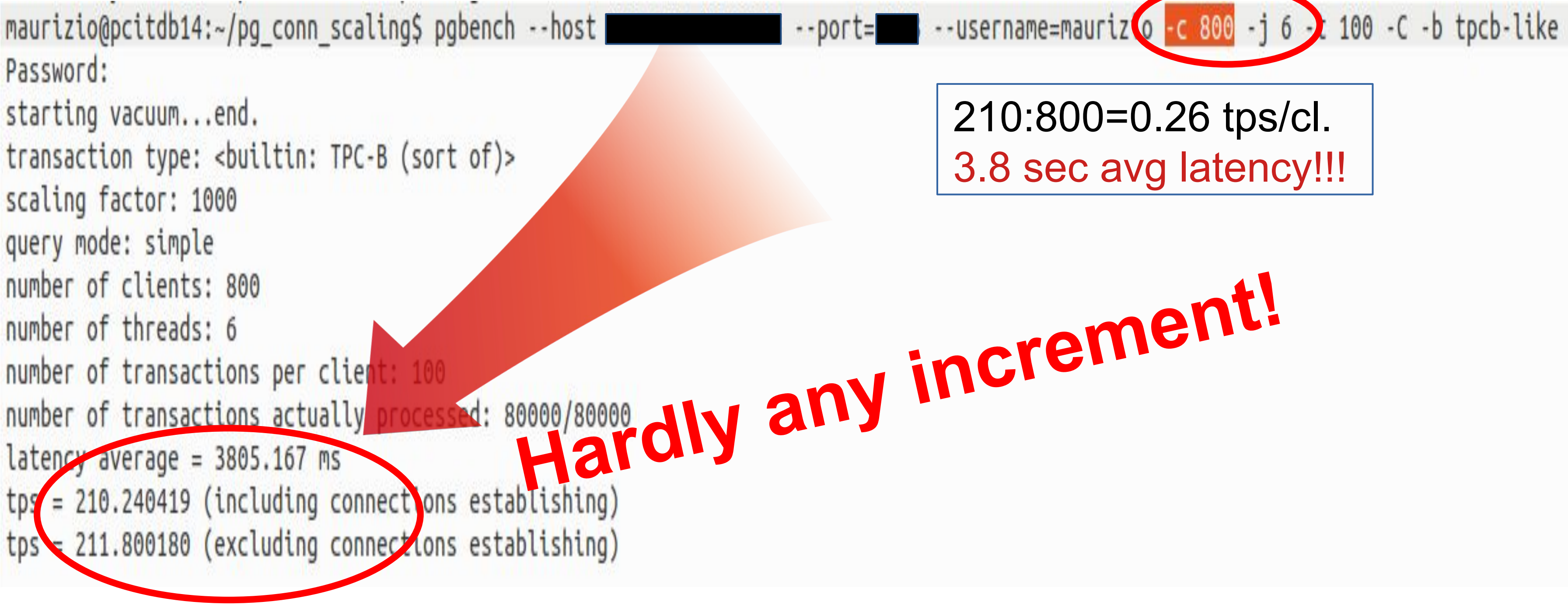


```
 maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host [redacted].cern.ch --port=[redacted] --username=maurizio -c 100 -j 4 -t 100 -S maurizio -C -b tpcb-like
Password:
starting vacuum...end.
transaction type: multiple scripts
scaling factor: 1000
query mode: simple
number of clients: 100
number of threads: 4
number of transactions per client: 100
number of transactions actually processed: 10000/10000
latency average = 576.116 ms
tps = 173.576210 (including connections establishing)
tps = 180.664884 (excluding connections establishing)
SQL script 1: <builtin: select only>
 - weight: 1 (targets 50.0% of total)
 - 4961 transactions (49.6% of total, tps = 86.111158)
 - latency average = 92.202 ms
 - latency stddev = 60.308 ms
SQL script 2: <builtin: TPC-B (sort of)>
 - weight: 1 (targets 50.0% of total)
 - 5039 transactions (50.4% of total, tps = 87.465052)
 - latency average = 812.450 ms
 - latency stddev = 201.239 ms
```

0.02 times less tps
174:100=1.74 tps/cl.
vs
178:100=1.78 tps/cl.

Connections scalability

A **tpcb-like** run with 800 pgbench clients, 6 threads, 100 trx/client



```
maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host [redacted] --port=[redacted] --username=maurizio -c 800 -j 6 -t 100 -C -b tpcb-like
Password:
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 800
number of threads: 6
number of transactions per client: 100
number of transactions actually processed: 80000/80000
latency average = 3805.167 ms
tps = 210.240419 (including connections establishing)
tps = 211.800180 (excluding connections establishing)
```

210:800=0.26 tps/cl.
3.8 sec avg latency!!!

Hardly any increment!

Connections scalability

What is the bottleneck?

Connections scalability

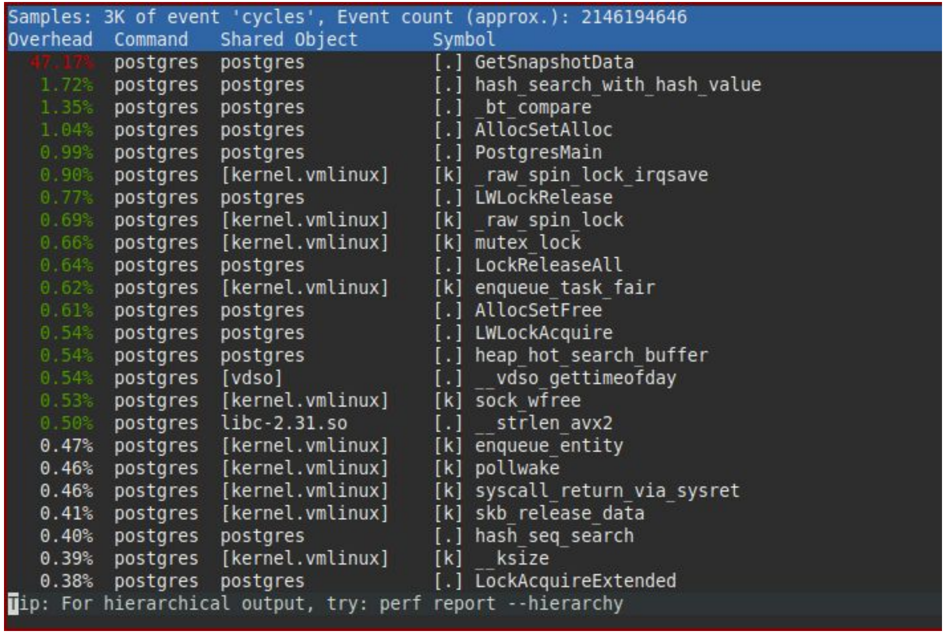
What is the bottleneck?

“Postgres uses a process forking model to handle concurrency instead of threading. When it accepts a new connection, the Postmaster forks a new backend (in postmaster.c). Backends are represented by the PGPROC structure (in proc.h), and the entire set of active processes is tracked in shared memory”

Credits Brandur, https://brandur.org/postgres-atomicity#shared-memory

Connections scalability

What is the bottleneck?



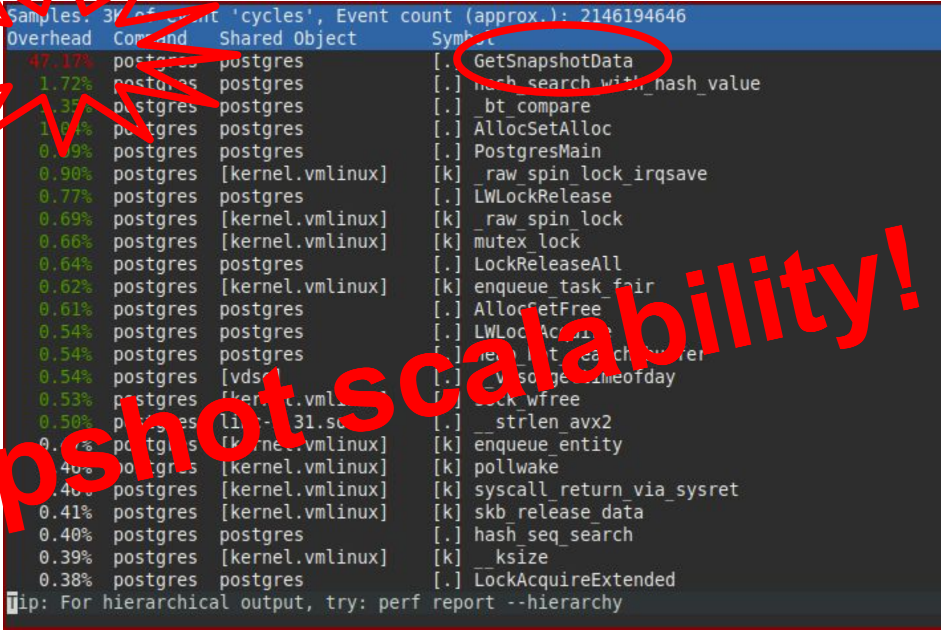
```
Samples: 3K of event 'cycles', Event count (approx.): 2146194646
Overhead  Command   Shared Object        Symbol
  47.17%  postgres  postgres             [.] GetSnapshotData
   1.72%  postgres  postgres             [.] hash_search_with_hash_value
   1.35%  postgres  postgres             [.] _bt_compare
   1.04%  postgres  postgres             [.] AllocSetAlloc
   0.99%  postgres  postgres             [.] PostgresMain
   0.90%  postgres  [kernel.vmlinux]     [k] _raw_spin_lock_irqsave
   0.77%  postgres  postgres             [.] LWLockRelease
   0.69%  postgres  [kernel.vmlinux]     [k] _raw_spin_lock
   0.66%  postgres  [kernel.vmlinux]     [k] mutex_lock
   0.64%  postgres  postgres             [.] LockReleaseAll
   0.62%  postgres  [kernel.vmlinux]     [k] enqueue_task_fair
   0.61%  postgres  postgres             [.] AllocSetFree
   0.54%  postgres  postgres             [.] LWLockAcquire
   0.54%  postgres  postgres             [.] heap_hot_search_buffer
   0.54%  postgres  [vdso]               [.] __vdso_gettimeofday
   0.53%  postgres  [kernel.vmlinux]     [k] sock_wfree
   0.50%  postgres  libc-2.31.so         [.] __strlen_avx2
   0.47%  postgres  [kernel.vmlinux]     [k] enqueue_entity
   0.46%  postgres  [kernel.vmlinux]     [k] pollwake
   0.46%  postgres  [kernel.vmlinux]     [k] syscall_return_via_sysret
   0.41%  postgres  [kernel.vmlinux]     [k] skb_release_data
   0.40%  postgres  postgres             [.] hash_seq_search
   0.39%  postgres  [kernel.vmlinux]     [k] __ksize
   0.38%  postgres  postgres             [.] LockAcquireExtended
Tip: For hierarchical output, try: perf report --hierarchy
```

Profile of one active connection running read-only pgbench concurrently with 5000 idle connections

Credits Andres Freund@MS http://cern.ch/go/9WRh

Connections scalability

What is the bottleneck?



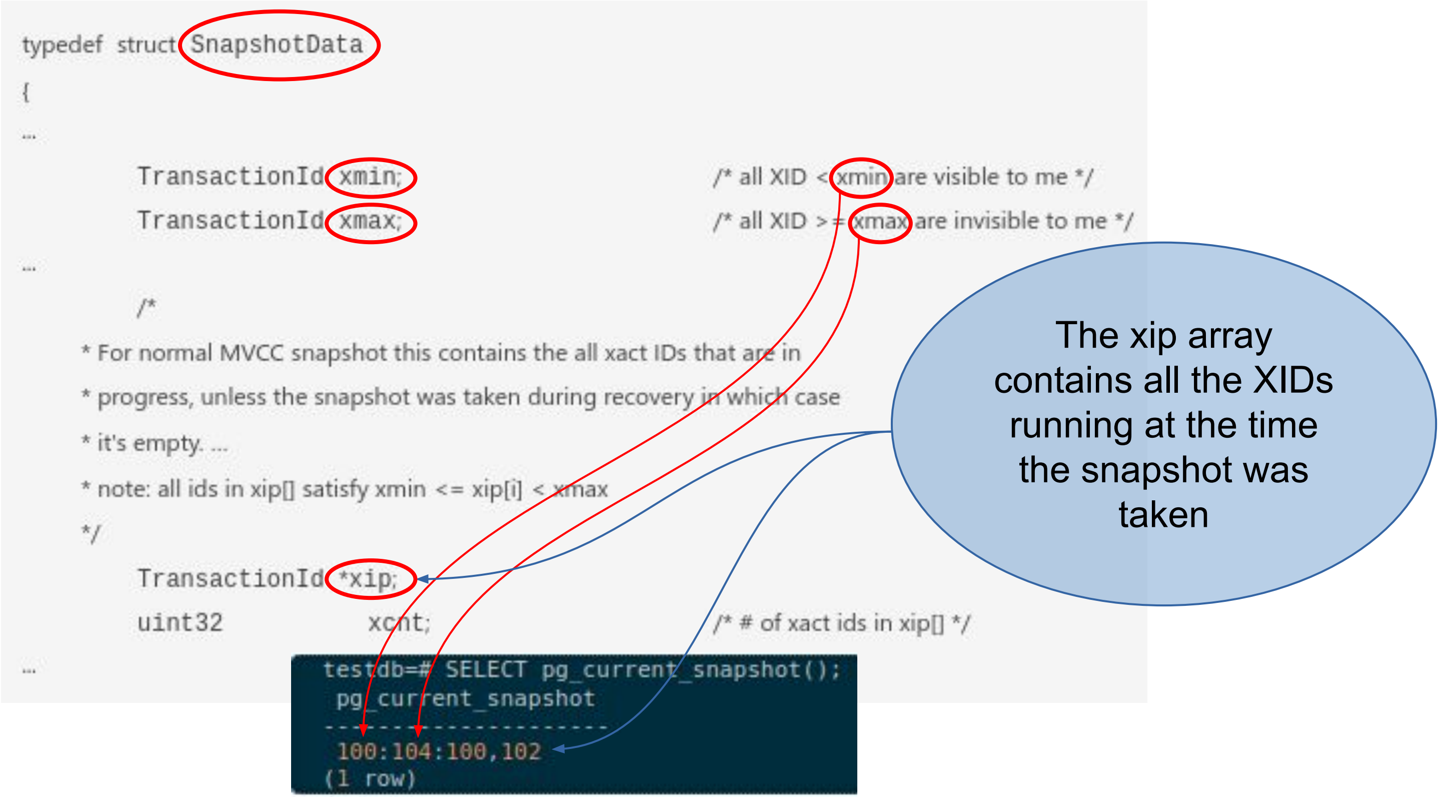
```
Samples: 3K of event 'cycles', Event count (approx.): 2146194646
Overhead  Command   Shared Object        Symbol
  47.17%  postgres  postgres             [.] GetSnapshotData
   1.72%  postgres  postgres             [.] hash_search_with_hash_value
   1.35%  postgres  postgres             [.] _bt_compare
   1.04%  postgres  postgres             [.] AllocSetAlloc
   0.99%  postgres  postgres             [.] PostgresMain
   0.90%  postgres  [kernel.vmlinux]     [k] _raw_spin_lock_irqsave
   0.77%  postgres  postgres             [.] LWLockRelease
   0.69%  postgres  [kernel.vmlinux]     [k] _raw_spin_lock
   0.66%  postgres  [kernel.vmlinux]     [k] mutex_lock
   0.64%  postgres  postgres             [.] LockReleaseAll
   0.62%  postgres  [kernel.vmlinux]     [k] enqueue_task_fair
   0.61%  postgres  postgres             [.] AllocSetFree
   0.54%  postgres  postgres             [.] LWLockAcquire
   0.54%  postgres  postgres             [.] heap_hot_search_buffer
   0.54%  postgres  [vdso]               [.] __vdso_gettimeofday
   0.53%  postgres  [kernel.vmlinux]     [k] sock_wfree
   0.50%  postgres  libc-2.31.so         [.] __strlen_avx2
   0.4[illegible]%  postgres  [kernel.vmlinux]     [k] enqueue_entity
   0.40%  postgres  [kernel.vmlinux]     [k] pollwake
   0.40%  postgres  [kernel.vmlinux]     [k] syscall_return_via_sysret
   0.41%  postgres  [kernel.vmlinux]     [k] skb_release_data
   0.40%  postgres  postgres             [.] hash_seq_search
   0.39%  postgres  [kernel.vmlinux]     [k] __ksize
   0.38%  postgres  postgres             [.] LockAcquireExtended
Tip: For hierarchical output, try: perf report --hierarchy
```

Snapshot scalability!

Profile of one active connection running read-only pgbench concurrently with 5000 idle connections

Credits Andres Freund@MS http://cern.ch/go/9WRh

Connections scalability

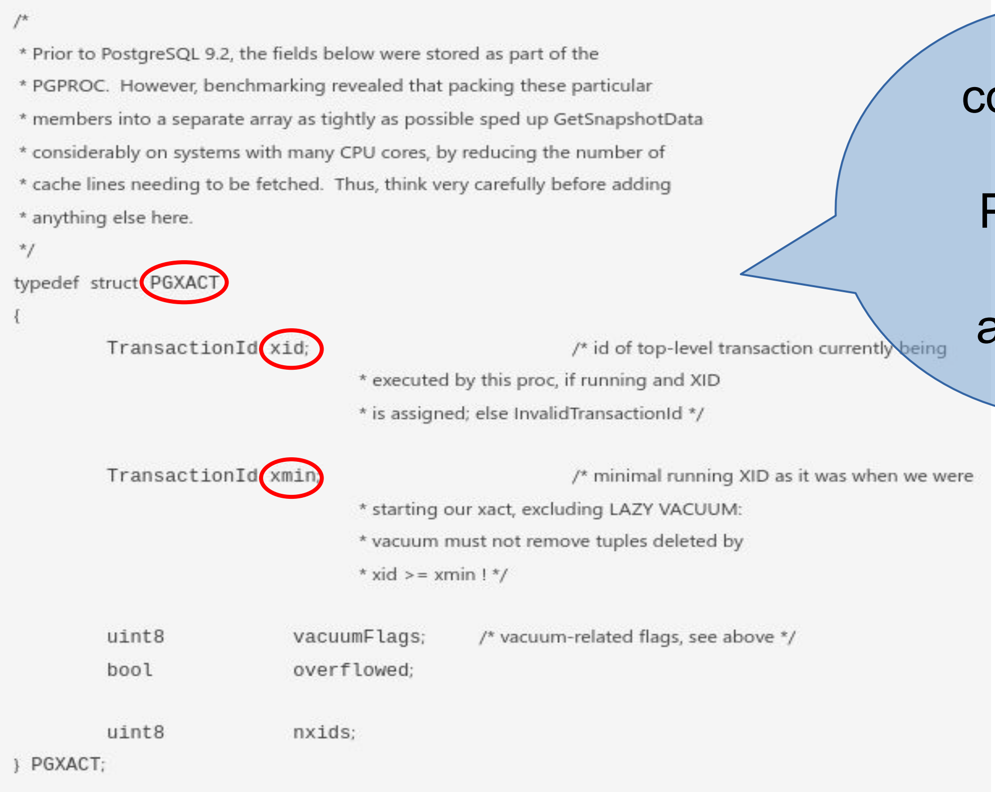


```
typedef  struct SnapshotData
{
...
        TransactionId xmin;                     /* all XID < xmin are visible to me */
        TransactionId xmax;                     /* all XID >= xmax are invisible to me */
...
        /*
     * For normal MVCC snapshot this contains the all xact IDs that are in
     * progress, unless the snapshot was taken during recovery in which case
     * it's empty. ...
     * note: all ids in xip[] satisfy xmin <= xip[i] < xmax
     */
        TransactionId *xip;
        uint32          xcnt;                   /* # of xact ids in xip[] */
...
```

```
testdb=# SELECT pg_current_snapshot();
 pg_current_snapshot
---------------------
 100:104:100,102
(1 row)
```

The xip array contains all the XIDs running at the time the snapshot was taken

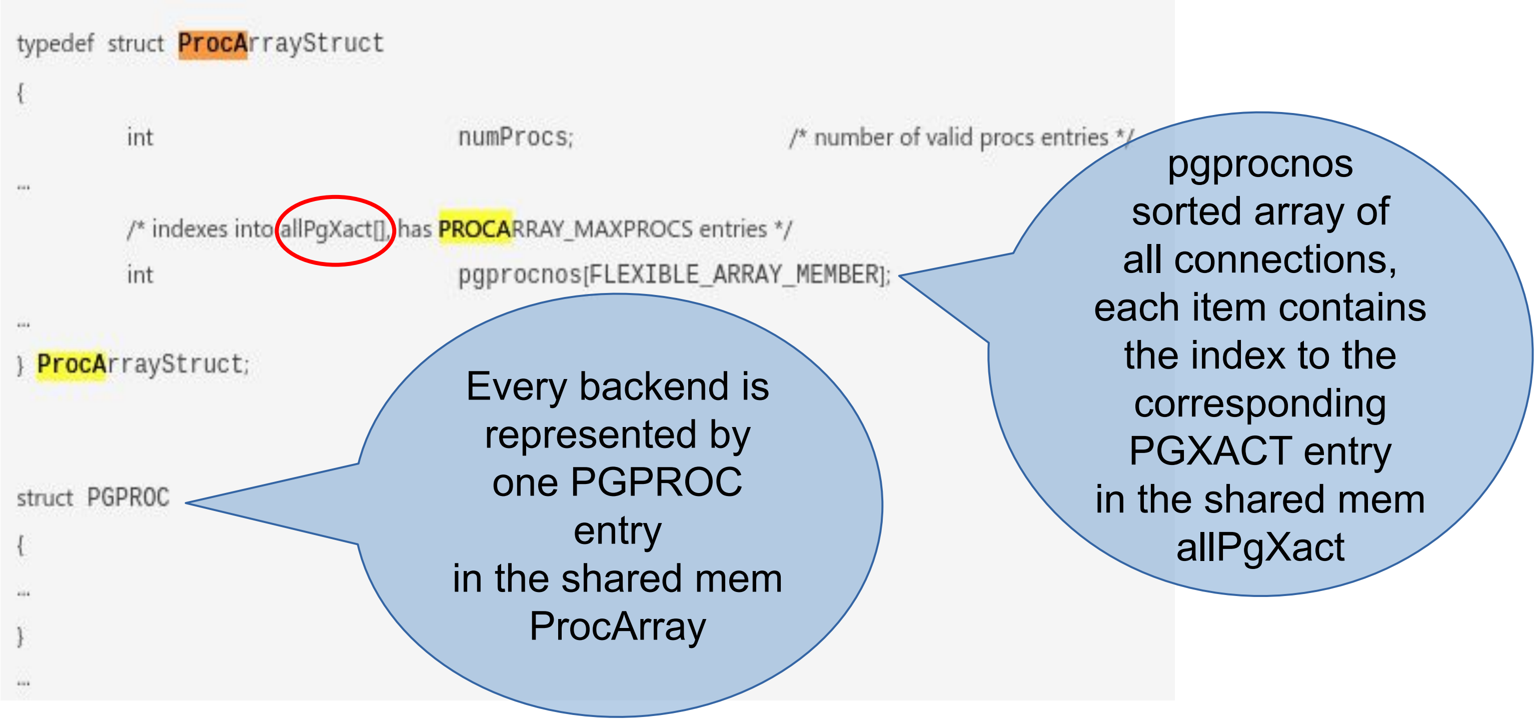
Connections scalability



```
/*
 * Prior to PostgreSQL 9.2, the fields below were stored as part of the
 * PGPROC.  However, benchmarking revealed that packing these particular
 * members into a separate array as tightly as possible sped up GetSnapshotData
 * considerably on systems with many CPU cores, by reducing the number of
 * cache lines needing to be fetched.  Thus, think very carefully before adding
 * anything else here.
 */
typedef  struct PGXACT
{
        TransactionId xid;                      /* id of top-level transaction currently being
                                                 * executed by this proc, if running and XID
                                                 * is assigned; else InvalidTransactionId */

        TransactionId xmin;                     /* minimal running XID as it was when we were
                                                 * starting our xact, excluding LAZY VACUUM:
                                                 * vacuum must not remove tuples deleted by
                                                 * xid >= xmin ! */

        uint8           vacuumFlags;    /* vacuum-related flags, see above */
        bool            overflowed;

        uint8           nxids;
} PGXACT;
```

Every connection has one PGXACT entry in allPgXact array

Connections scalability

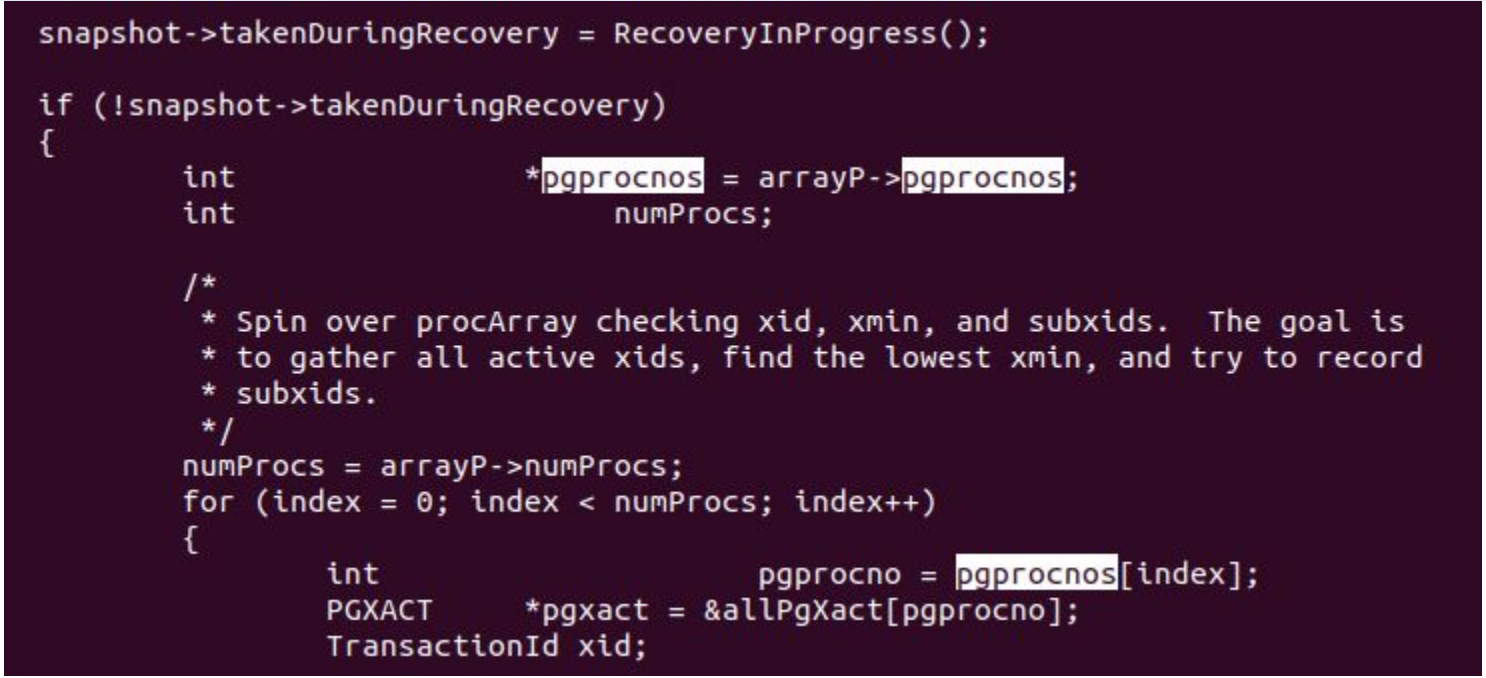


```
typedef  struct  ProcArrayStruct
{
            int                          numProcs;                    /* number of valid procs entries */
...
            /* indexes into allPgXact[], has PROCARRAY_MAXPROCS entries */
            int                          pgprocnos[FLEXIBLE_ARRAY_MEMBER];
...
} ProcArrayStruct;


struct  PGPROC
{
...
}
...
```

Every backend is represented by one PGPROC entry
in the shared mem ProcArray

pgprocnos
sorted array of all connections, each item contains the index to the corresponding PGXACT entry
in the shared mem allPgXact

Connections scalability

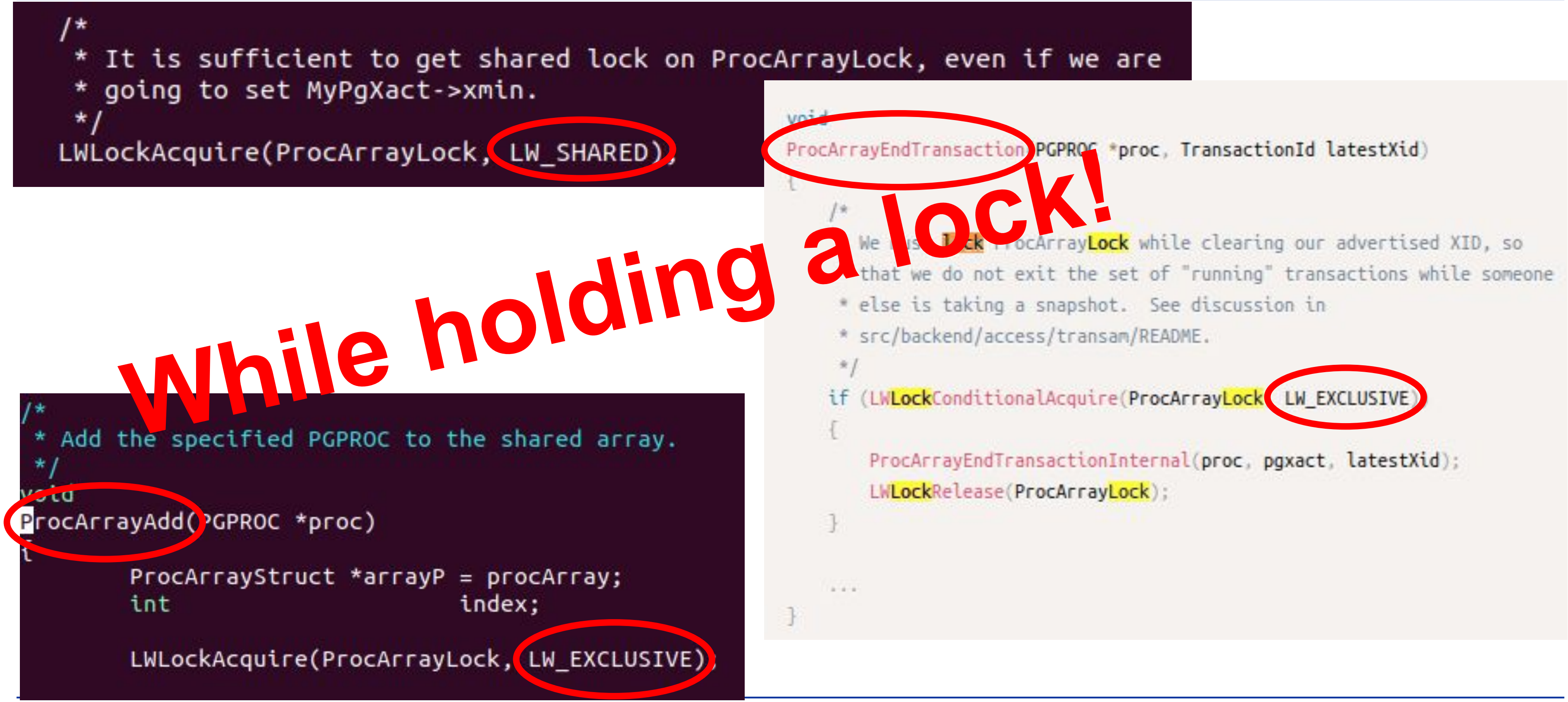


```
snapshot->takenDuringRecovery = RecoveryInProgress();

if (!snapshot->takenDuringRecovery)
{
        int                *pgprocnos = arrayP->pgprocnos;
        int                     numProcs;

        /*
         * Spin over procArray checking xid, xmin, and subxids.  The goal is
         * to gather all active xids, find the lowest xmin, and try to record
         * subxids.
         */
        numProcs = arrayP->numProcs;
        for (index = 0; index < numProcs; index++)
        {
                int                     pgprocno = pgprocnos[index];
                PGXACT     *pgxact = &allPgXact[pgprocno];
                TransactionId xid;
```

GetSnapshotData() iterates over **all** entries in pgprocnos (ProcArray), collecting PGXACT->xid for all connections with an assigned transaction ID

Credits Andres Freund@MS http://cern.ch/go/9WRh

Connections scalability



```
/*
 * It is sufficient to get shared lock on ProcArrayLock, even if we are
 * going to set MyPgXact->xmin.
 */
LWLockAcquire(ProcArrayLock, LW_SHARED);
```

```
void
ProcArrayEndTransaction(PGPROC *proc, TransactionId latestXid)
{
    /*
     * We must lock ProcArrayLock while clearing our advertised XID, so
     * that we do not exit the set of "running" transactions while someone
     * else is taking a snapshot.  See discussion in
     * src/backend/access/transam/README.
     */
    if (LWLockConditionalAcquire(ProcArrayLock, LW_EXCLUSIVE))
    {
        ProcArrayEndTransactionInternal(proc, pgxact, latestXid);
        LWLockRelease(ProcArrayLock);
    }

    ...
}
```

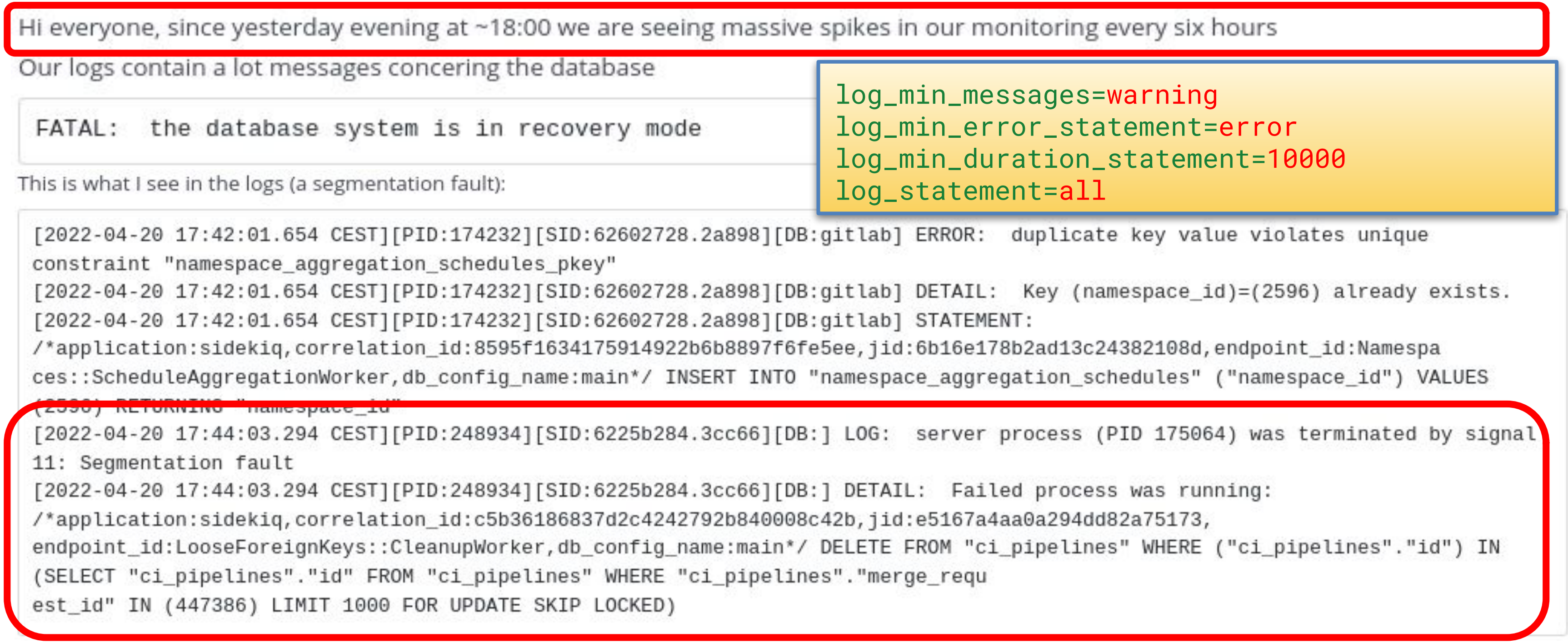
While holding a lock!

```
/*
 * Add the specified PGPROC to the shared array.
 */
void
ProcArrayAdd(PGPROC *proc)
{
        ProcArrayStruct *arrayP = procArray;
        int                     index;

        LWLockAcquire(ProcArrayLock, LW_EXCLUSIVE);
```

Agenda

- **An MVCC primer (boring things everyone knows but it is worth refreshing)**
    - (ACID) Transactions, Isolation, Concurrency, Serializable Snapshot Isolation
- **Connection scalability (showing the problem and its causes)**
  - Benchmarking & bottleneck analysis
- **Troubleshooting GitLab issues (talking about that time when we all had a lot of fun)**
  - The journey to enlightenment
  - The joy of enlightenment
- **The great effects of connection pooling on connection scalability**

Troubleshooting GitLab issues



Hi everyone, since yesterday evening at ~18:00 we are seeing massive spikes in our monitoring every six hours

Our logs contain a lot messages concering the database

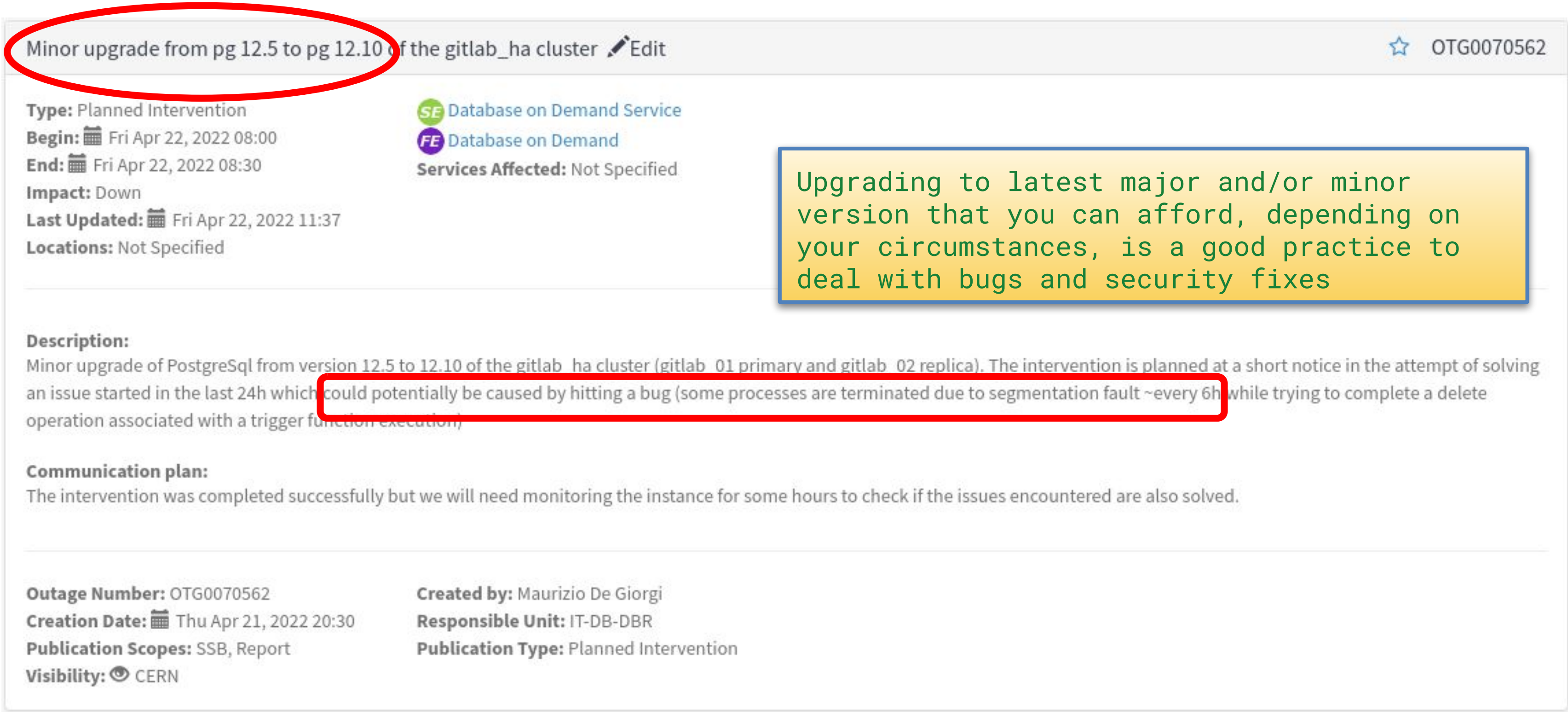
```
FATAL:  the database system is in recovery mode
```

This is what I see in the logs (a segmentation fault):

```
[2022-04-20 17:42:01.654 CEST][PID:174232][SID:62602728.2a898][DB:gitlab] ERROR:  duplicate key value violates unique
constraint "namespace_aggregation_schedules_pkey"
[2022-04-20 17:42:01.654 CEST][PID:174232][SID:62602728.2a898][DB:gitlab] DETAIL:  Key (namespace_id)=(2596) already exists.
[2022-04-20 17:42:01.654 CEST][PID:174232][SID:62602728.2a898][DB:gitlab] STATEMENT:
/*application:sidekiq,correlation_id:8595f1634175914922b6b8897f6fe5ee,jid:6b16e178b2ad13c24382108d,endpoint_id:Namespa
ces::ScheduleAggregationWorker,db_config_name:main*/ INSERT INTO "namespace_aggregation_schedules" ("namespace_id") VALUES
(2596) RETURNING "namespace_id"
[2022-04-20 17:44:03.294 CEST][PID:248934][SID:6225b284.3cc66][DB:] LOG:  server process (PID 175064) was terminated by signal
11: Segmentation fault
[2022-04-20 17:44:03.294 CEST][PID:248934][SID:6225b284.3cc66][DB:] DETAIL:  Failed process was running:
/*application:sidekiq,correlation_id:c5b36186837d2c4242792b840008c42b,jid:e5167a4aa0a294dd82a75173,
endpoint_id:LooseForeignKeys::CleanupWorker,db_config_name:main*/ DELETE FROM "ci_pipelines" WHERE ("ci_pipelines"."id") IN
(SELECT "ci_pipelines"."id" FROM "ci_pipelines" WHERE "ci_pipelines"."merge_requ
est_id" IN (447386) LIMIT 1000 FOR UPDATE SKIP LOCKED)
```

```
log_min_messages=warning
log_min_error_statement=error
log_min_duration_statement=10000
log_statement=all
```

Troubleshooting GitLab issues



Minor upgrade from pg 12.5 to pg 12.10 of the gitlab_ha cluster ✎Edit ☆ OTG0070562

Type: Planned Intervention
Begin: Fri Apr 22, 2022 08:00
End: Fri Apr 22, 2022 08:30
Impact: Down
Last Updated: Fri Apr 22, 2022 11:37
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Minor upgrade of PostgreSql from version 12.5 to 12.10 of the gitlab_ha cluster (gitlab_01 primary and gitlab_02 replica). The intervention is planned at a short notice in the attempt of solving an issue started in the last 24h which could potentially be caused by hitting a bug (some processes are terminated due to segmentation fault ~every 6h while trying to complete a delete operation associated with a trigger function execution)

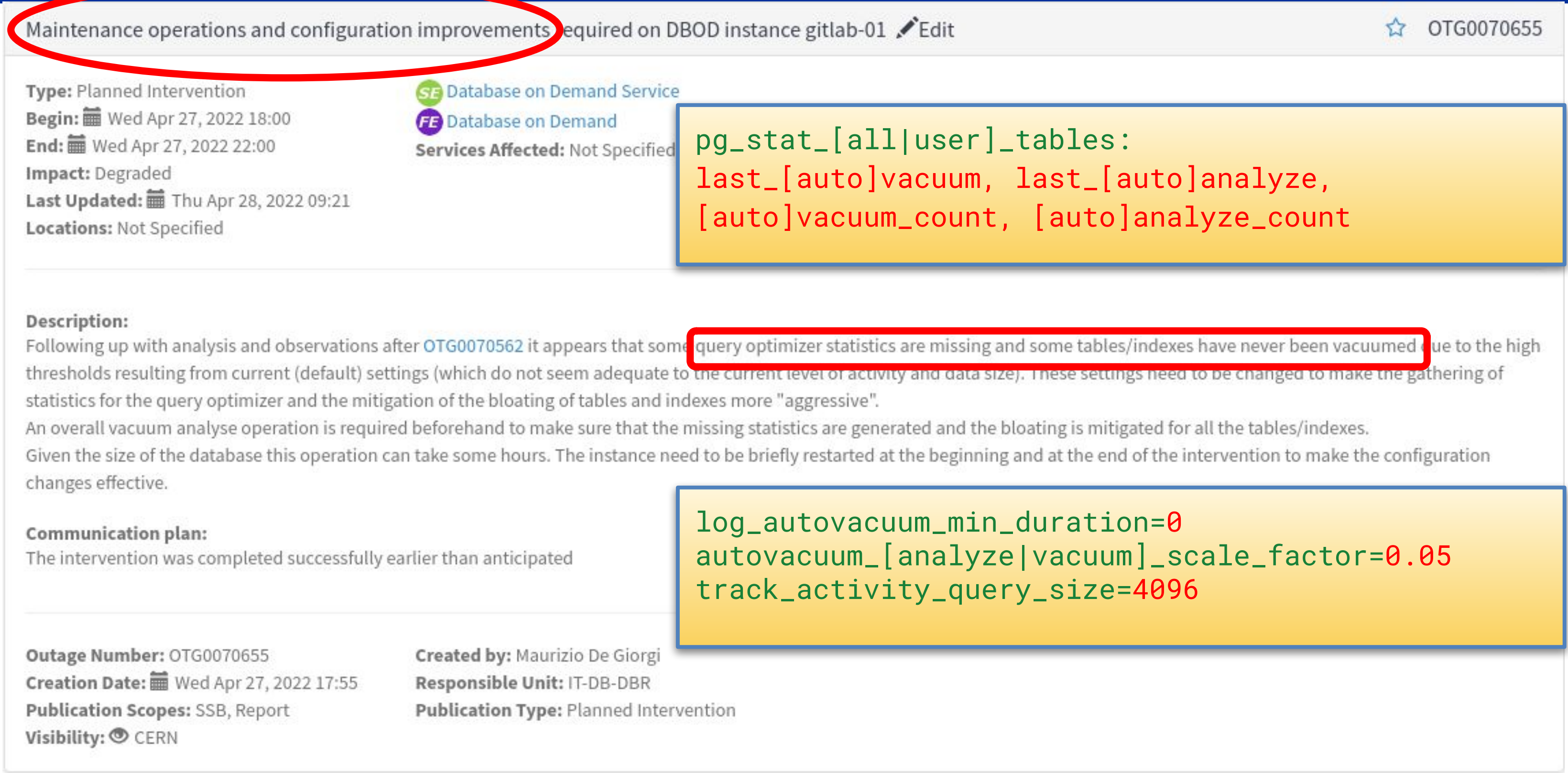
Communication plan:
The intervention was completed successfully but we will need monitoring the instance for some hours to check if the issues encountered are also solved.

Outage Number: OTG0070562
Creation Date: Thu Apr 21, 2022 20:30
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

Upgrading to latest major and/or minor version that you can afford, depending on your circumstances, is a good practice to deal with bugs and security fixes

Troubleshooting GitLab issues



Maintenance operations and configuration improvements required on DBOD instance gitlab-01 ✎Edit ☆ OTG0070655

Type: Planned Intervention
Begin: Wed Apr 27, 2022 18:00
End: Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up with analysis and observations after OTG0070562 it appears that some query optimizer statistics are missing and some tables/indexes have never been vacuumed due to the high thresholds resulting from current (default) settings (which do not seem adequate to the current level of activity and data size). These settings need to be changed to make the gathering of statistics for the query optimizer and the mitigation of the bloating of tables and indexes more "aggressive".
An overall vacuum analyse operation is required beforehand to make sure that the missing statistics are generated and the bloating is mitigated for all the tables/indexes.
Given the size of the database this operation can take some hours. The instance need to be briefly restarted at the beginning and at the end of the intervention to make the configuration changes effective.

Communication plan:
The intervention was completed successfully earlier than anticipated

Outage Number: OTG0070655
Creation Date: Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: CERN

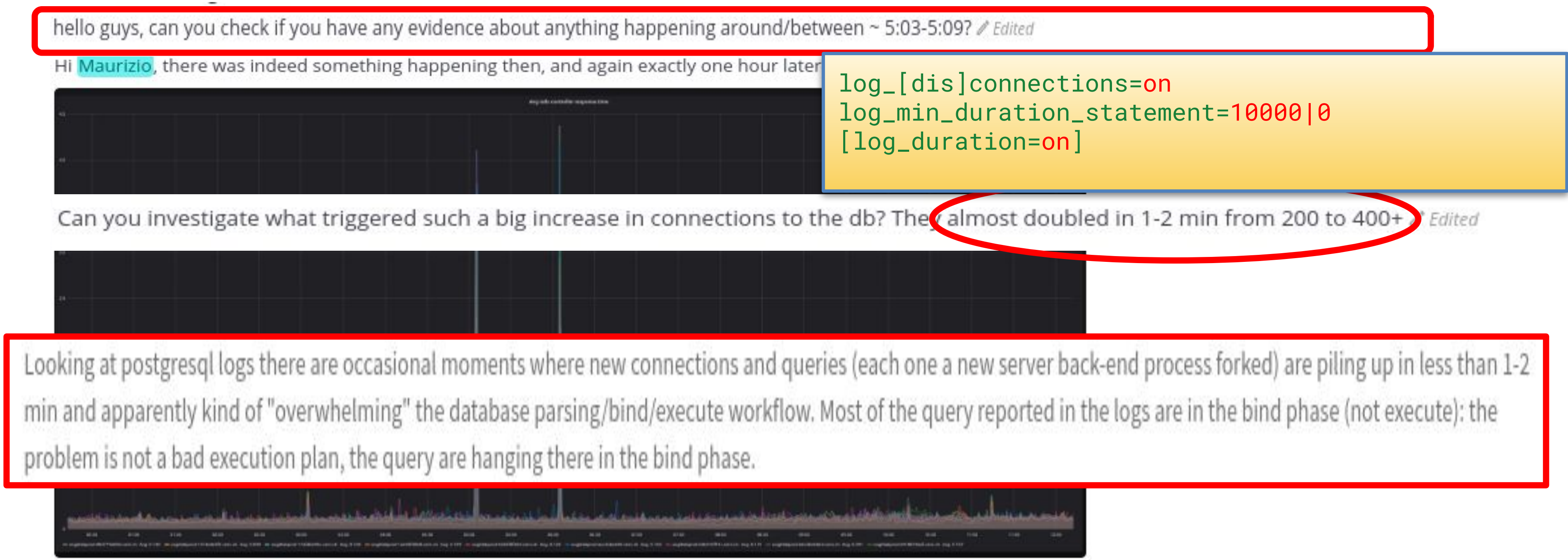
Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

```
pg_stat_[all|user]_tables:
last_[auto]vacuum, last_[auto]analyze,
[auto]vacuum_count, [auto]analyze_count
```

```
log_autovacuum_min_duration=0
autovacuum_[analyze|vacuum]_scale_factor=0.05
track_activity_query_size=4096
```

Troubleshooting GitLab issues

The first clues



hello guys, can you check if you have any evidence about anything happening around/between ~ 5:03-5:09? Edited

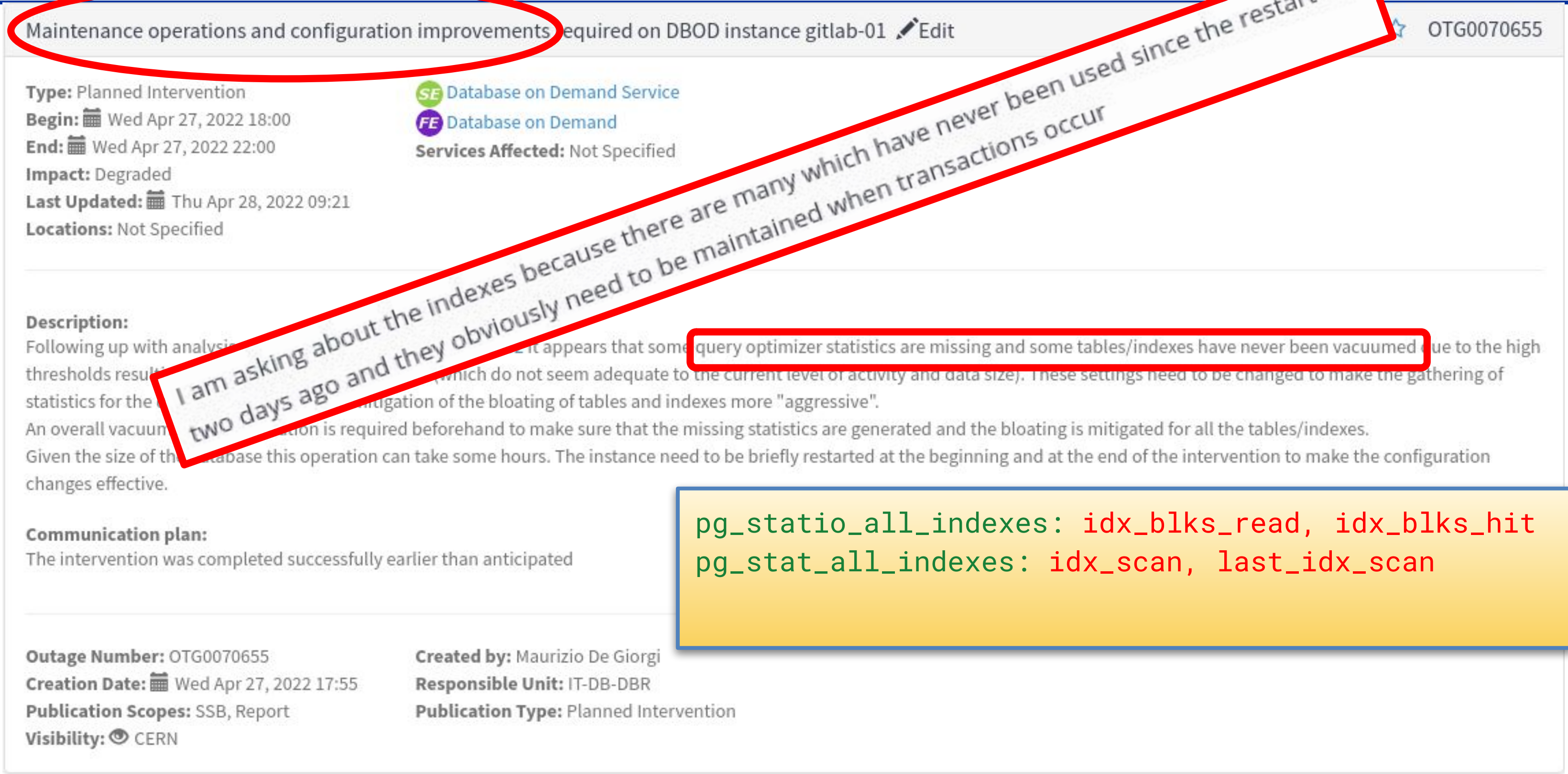
Hi Maurizio, there was indeed something happening then, and again exactly one hour later

```
log_[dis]connections=on
log_min_duration_statement=10000|0
[log_duration=on]
```

Can you investigate what triggered such a big increase in connections to the db? They almost doubled in 1-2 min from 200 to 400+ Edited

Looking at postgresql logs there are occasional moments where new connections and queries (each one a new server back-end process forked) are piling up in less than 1-2 min and apparently kind of "overwhelming" the database parsing/bind/execute workflow. Most of the query reported in the logs are in the bind phase (not execute): the problem is not a bad execution plan, the query are hanging there in the bind phase.

Troubleshooting GitLab issues



Maintenance operations and configuration improvements required on DBOD instance gitlab-01 ✎Edit ☆ OTG0070655

Type: Planned Intervention
Begin: Wed Apr 27, 2022 18:00
End: Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up with analysi[illegible] it appears that some query optimizer statistics are missing and some tables/indexes have never been vacuumed due to the high thresholds resul[illegible] which do not seem adequate to the current level of activity and data size). These settings need to be changed to make the gathering of statistics for the [illegible] mitigation of the bloating of tables and indexes more "aggressive".
An overall vacuum [illegible] ion is required beforehand to make sure that the missing statistics are generated and the bloating is mitigated for all the tables/indexes.
Given the size of th[illegible]atabase this operation can take some hours. The instance need to be briefly restarted at the beginning and at the end of the intervention to make the configuration changes effective.

Communication plan:
The intervention was completed successfully earlier than anticipated

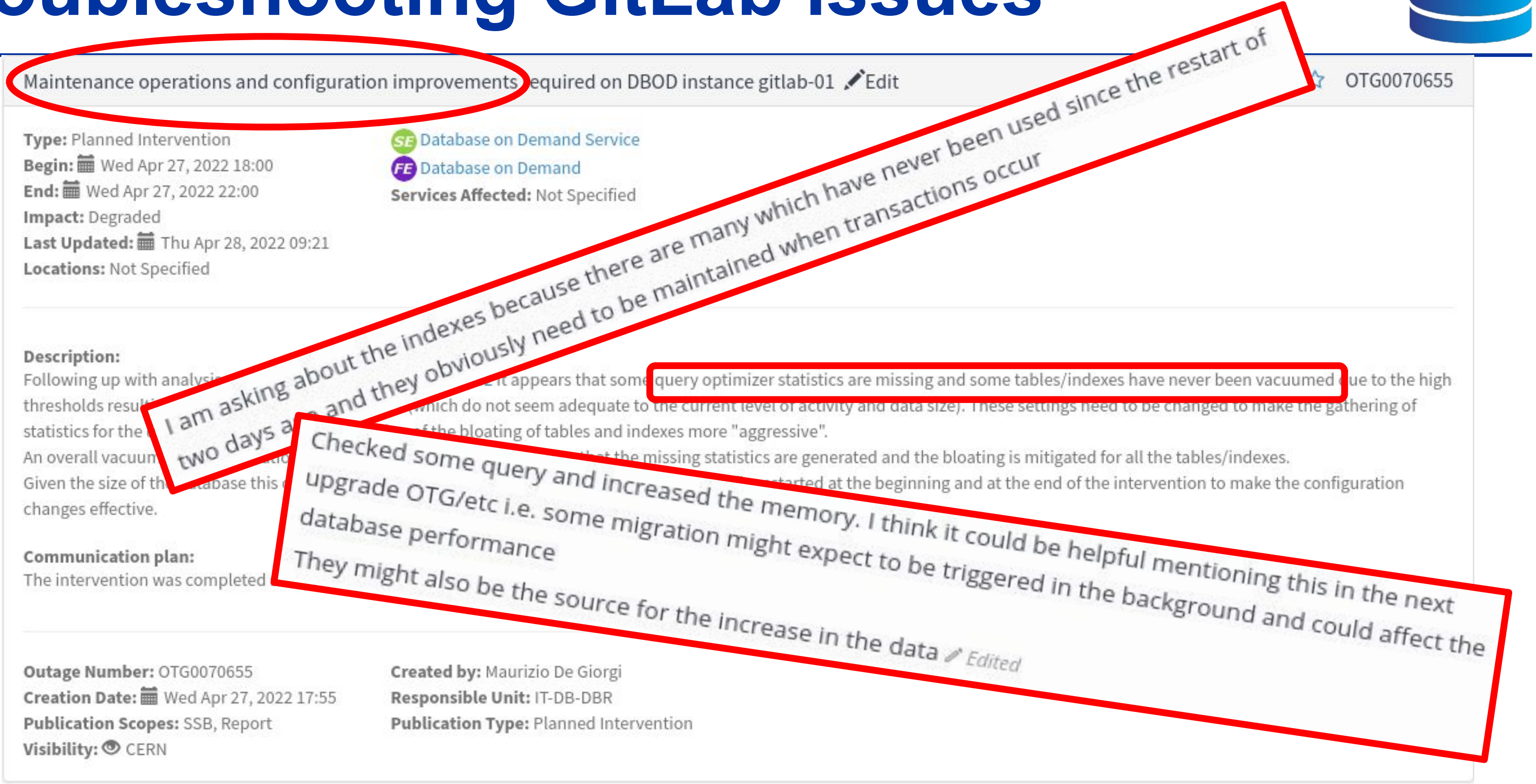
Outage Number: OTG0070655
Creation Date: Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

I am asking about the indexes because there are many which have never been used since the restart of two days ago and they obviously need to be maintained when transactions occur

```
pg_statio_all_indexes: idx_blks_read, idx_blks_hit
pg_stat_all_indexes: idx_scan, last_idx_scan
```

Troubleshooting GitLab issues



Maintenance operations and configuration improvements equired on DBOD instance gitlab-01 Edit ☆ OTG0070655

Type: Planned Intervention
Begin: Wed Apr 27, 2022 18:00
End: Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

I am asking about the indexes because there are many which have never been used since the restart of two days a and they obviously need to be maintained when transactions occur

Description:
Following up with analysi it appears that some query optimizer statistics are missing and some tables/indexes have never been vacuumed ue to the high thresholds resul which do not seem adequate to the current level of activity and data size). These settings need to be changed to make the gathering of statistics for the of the bloating of tables and indexes more "aggressive".
An overall vacuum that the missing statistics are generated and the bloating is mitigated for all the tables/indexes.
Given the size of th atabase this tarted at the beginning and at the end of the intervention to make the configuration changes effective.

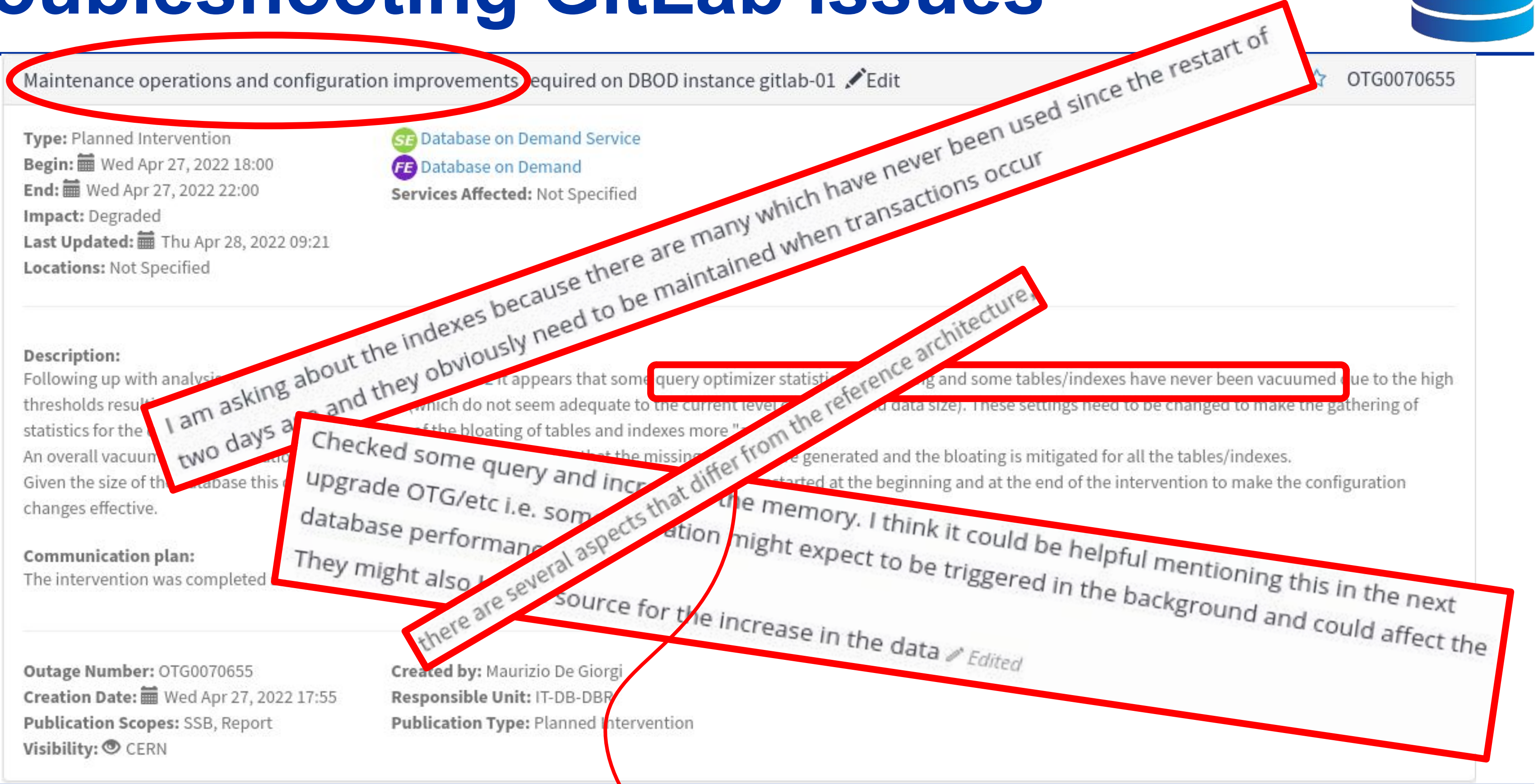
Checked some query and increased the memory. I think it could be helpful mentioning this in the next upgrade OTG/etc i.e. some migration might expect to be triggered in the background and could affect the database performance
They might also be the source for the increase in the data Edited

Communication plan:
The intervention was completed

Outage Number: OTG0070655
Creation Date: Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

Troubleshooting GitLab issues



Maintenance operations and configuration improvements required on DBOD instance gitlab-01 ✎Edit ☆ OTG0070655

Type: Planned Intervention
Begin: Wed Apr 27, 2022 18:00
End: Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up with analysi… it appears that some query optimizer statisti… g and some tables/indexes have never been vacuumed due to the high thresholds resul… which do not seem adequate to the current level … d data size). These settings need to be changed to make the gathering of statistics for the … of the bloating of tables and indexes more "…
An overall vacuum … that the missing … e generated and the bloating is mitigated for all the tables/indexes.
Given the size of the database this … started at the beginning and at the end of the intervention to make the configuration changes effective.

Communication plan:
The intervention was completed …

Outage Number: OTG0070655
Creation Date: Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

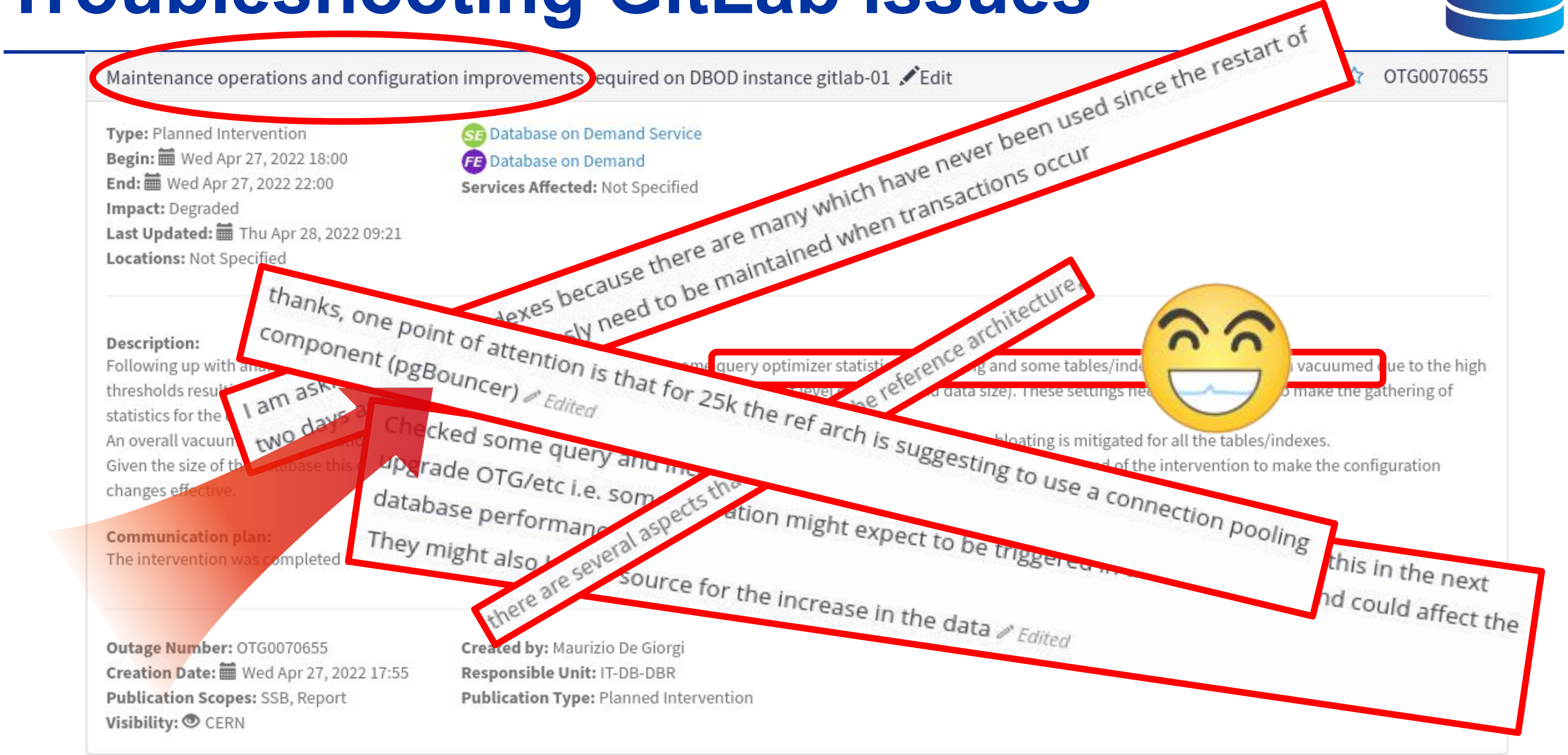
I am asking about the indexes because there are many which have never been used since the restart of two days ago and they obviously need to be maintained when transactions occur

Checked some query and incr… the memory. I think it could be helpful mentioning this in the next upgrade OTG/etc i.e. som… ation might expect to be triggered in the background and could affect the database performan…
They might also … source for the increase in the data ✎ Edited

there are several aspects that differ from the reference architecture,

https://docs.gitlab.com/ee/administration/reference_architectures/index.html

Troubleshooting GitLab issues



Maintenance operations and configuration improvements required on DBOD instance gitlab-01 Edit

OTG0070655

Type: Planned Intervention

Begin: Wed Apr 27, 2022 18:00

End: Wed Apr 27, 2022 22:00

Impact: Degraded

Last Updated: Thu Apr 28, 2022 09:21

Locations: Not Specified

SE Database on Demand Service

FE Database on Demand

Services Affected: Not Specified

Description:

Following up with ana… query optimizer statisti… g and some tables/inde… vacuumed due to the high thresholds resul… level … d data size). These settings ne… o make the gathering of statistics for the …

An overall vacuum … bloating is mitigated for all the tables/indexes.

Given the size of the database this … d of the intervention to make the configuration changes effective.

Communication plan:

The intervention was completed …

Outage Number: OTG0070655

Creation Date: Wed Apr 27, 2022 17:55

Publication Scopes: SSB, Report

Visibility: CERN

Created by: Maurizio De Giorgi

Responsible Unit: IT-DB-DBR

Publication Type: Planned Intervention

… dexes because there are many which have never been used since the restart of
… sly need to be maintained when transactions occur

thanks, one point of attention is that for 25k the ref arch is suggesting to use a connection pooling component (pgBouncer) Edited

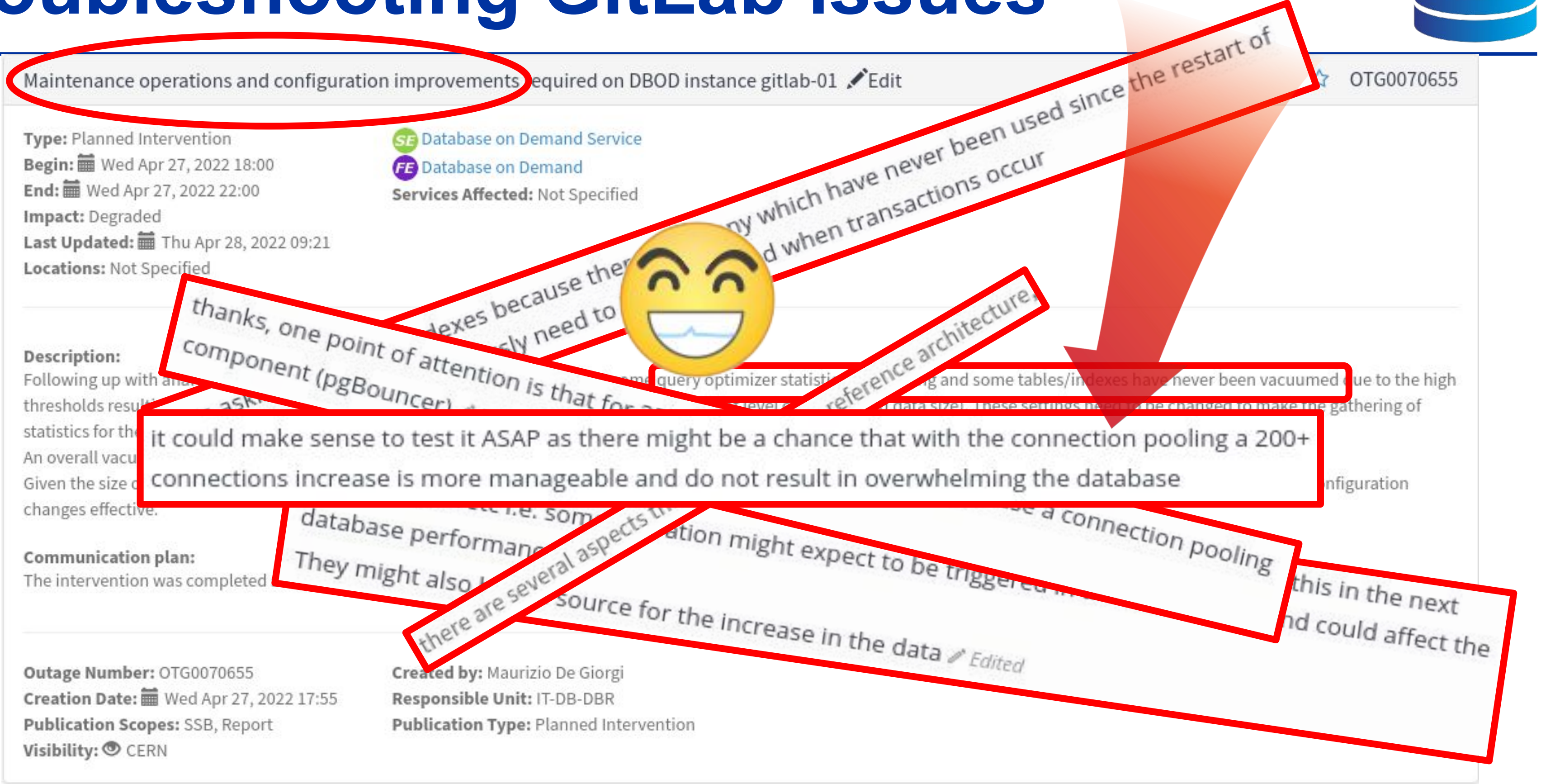
… he reference architecture,

I am ask…
two days a…

Checked some query and in…
upgrade OTG/etc i.e. som… ation might expect to be triggered in … this in the next
database performan… nd could affect the
They might also … source for the increase in the data Edited

there are several aspects tha…

Troubleshooting GitLab issues



Maintenance operations and configuration improvements required on DBOD instance gitlab-01 Edit ☆ OTG0070655

Type: Planned Intervention
Begin: ▦ Wed Apr 27, 2022 18:00
End: ▦ Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: ▦ Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up with an … some query optimizer statisti… g and some tables/indexes have never been vacuumed due to the high thresholds resul… level … data size). These settings need to be changed to make the gathering of statistics for the …
An overall vacu…
Given the size o… nfiguration changes effective.

Communication plan:
The intervention was completed …

Outage Number: OTG0070655
Creation Date: ▦ Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: ◉ CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

…dexes because the… ny which have never been used since the restart of
…sly need to … d when transactions occur

thanks, one point of attention is that for a…
component (pgBouncer) …

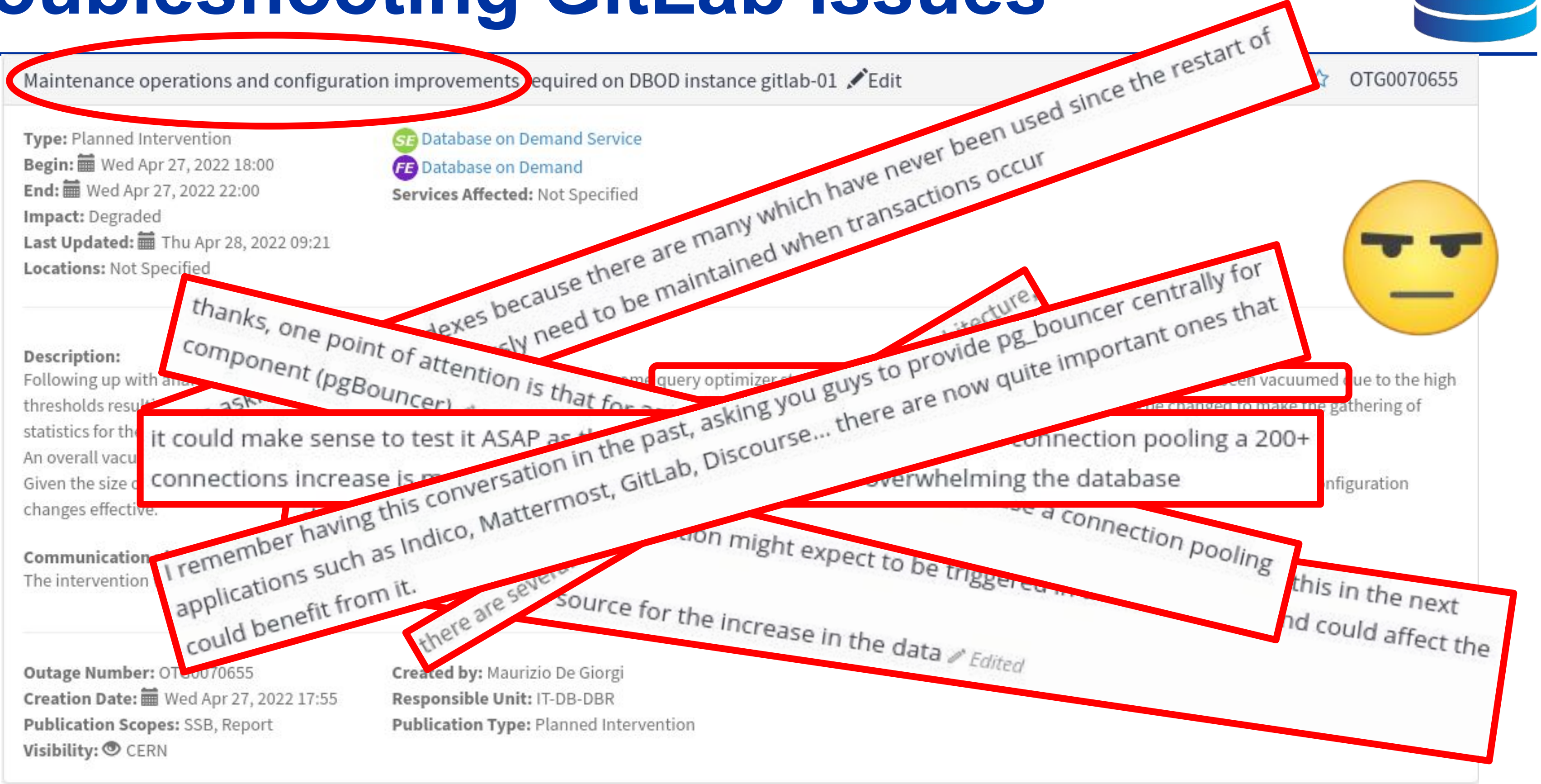
…reference architecture,

there are several aspects t…

it could make sense to test it ASAP as there might be a chance that with the connection pooling a 200+ connections increase is more manageable and do not result in overwhelming the database

… i.e. som… e a connection pooling
database performan… ation might expect to be triggered …
They might also … this in the next
… source for the increase in the data ✎ Edited … nd could affect the

Troubleshooting GitLab issues



Maintenance operations and configuration improvements equired on DBOD instance gitlab-01 ✎Edit ☆ OTG0070655

Type: Planned Intervention
Begin: ▦ Wed Apr 27, 2022 18:00
End: ▦ Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: ▦ Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up with an … query optimizer … een vacuumed due to the high thresholds resu… e changed to make the gathering of statistics for th…
An overall vacu…
Given the size … nfiguration changes effective.

Communication…
The intervention …

Outage Number: OT…0070655
Creation Date: ▦ Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: 👁 CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

…dexes because there are many which have never been used since the restart of …ly need to be maintained when transactions occur

thanks, one point of attention is that for a… component (pgBouncer) …

…itecture, … ask…

it could make sense to test it ASAP as t… connection pooling a 200+ connections increase is … overwhelming the database

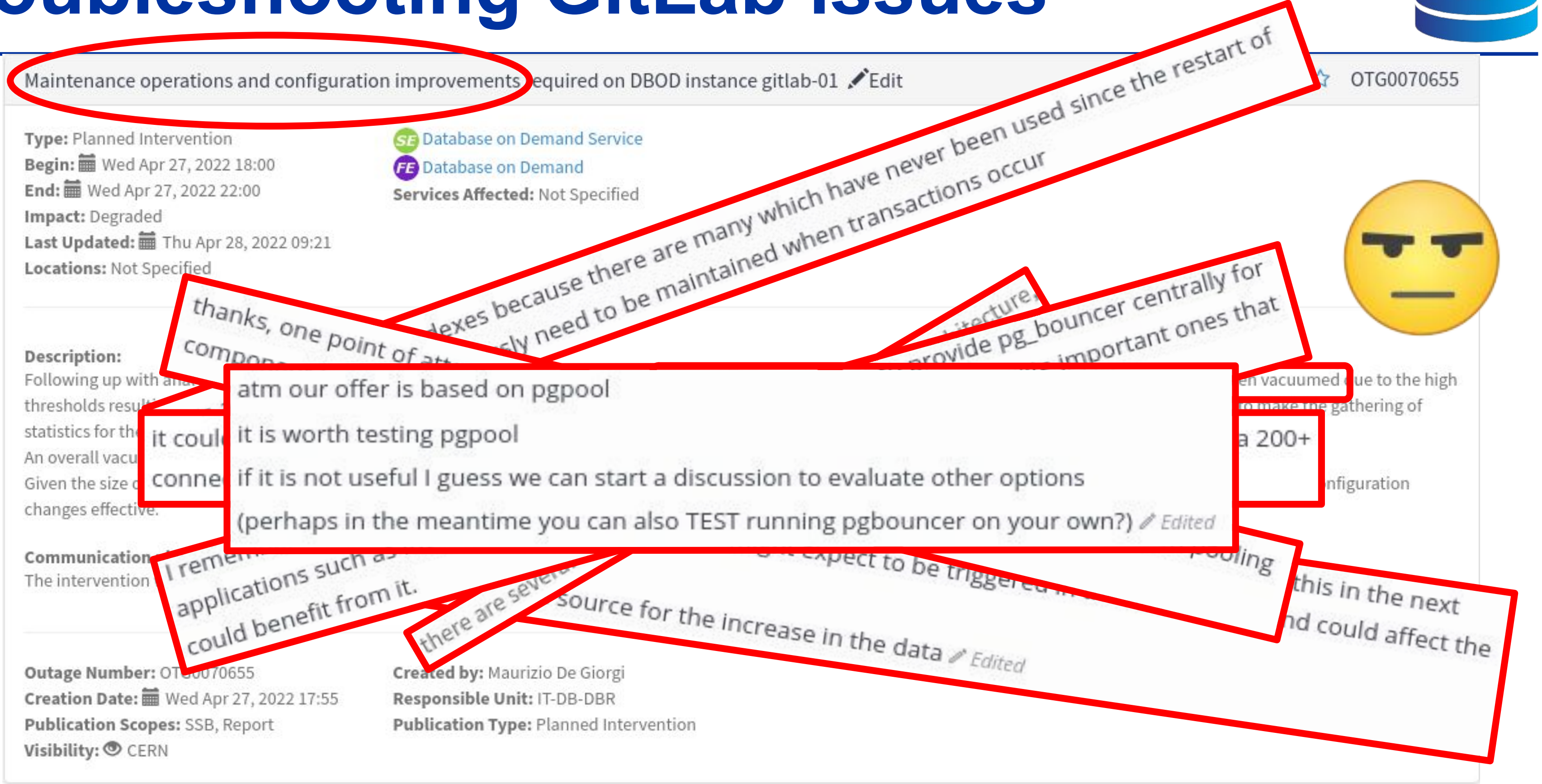
I remember having this conversation in the past, asking you guys to provide pg_bouncer centrally for applications such as Indico, Mattermost, GitLab, Discourse... there are now quite important ones that could benefit from it.

…e a connection pooling …ion might expect to be triggered in…

there are seve…

…this in the next …nd could affect the … source for the increase in the data ✎ Edited

Troubleshooting GitLab issues



Maintenance operations and configuration improvements equired on DBOD instance gitlab-01 Edit ☆ OTG0070655

Type: Planned Intervention
Begin: Wed Apr 27, 2022 18:00
End: Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up with an… thresholds resul… statistics for th… An overall vacu… Given the size o… changes effective.

…en vacuumed due to the high …to make the gathering of … …nfiguration

Communication:
The intervention …

Outage Number: OTG0070655
Creation Date: Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

…dexes because there are many which have never been used since the restart of …ly need to be maintained when transactions occur

thanks, one point of att… compon…

…itecture… …provide pg_bouncer centrally for … important ones that

it coul… conne… …a 200+

atm our offer is based on pgpool

it is worth testing pgpool

if it is not useful I guess we can start a discussion to evaluate other options

(perhaps in the meantime you can also TEST running pgbouncer on your own?) Edited

I remem… applications such as… could benefit from it.

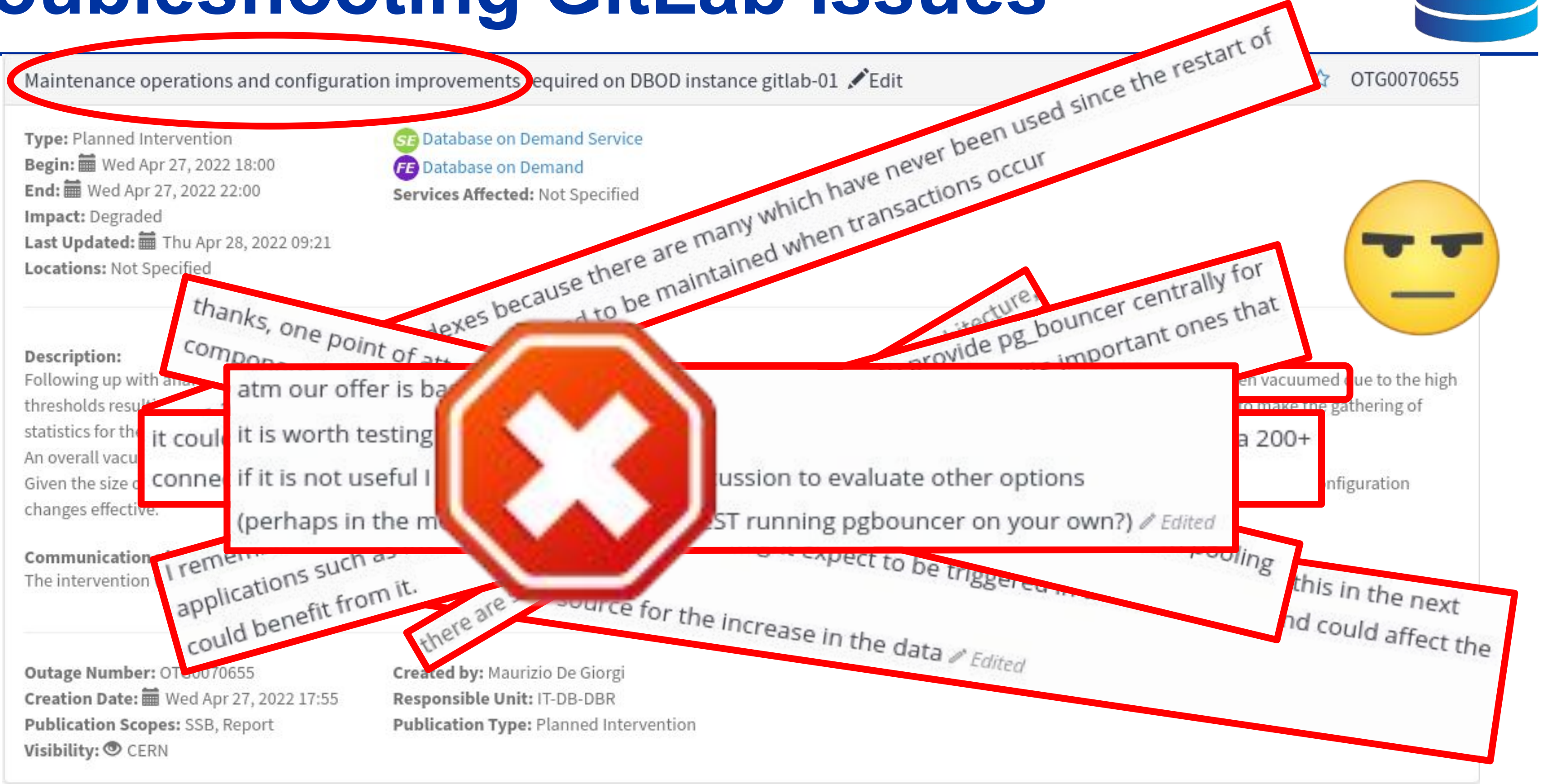
there are sever…

…expect to be triggered… …pooling

…source for the increase in the data Edited

…this in the next …nd could affect the

Troubleshooting GitLab issues



Maintenance operations and configuration improvements required on DBOD instance gitlab-01 Edit

OTG0070655

Type: Planned Intervention
Begin: Wed Apr 27, 2022 18:00
End: Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: Thu Apr 28, 2022 09:21
Locations: Not Specified

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up with an
thresholds resul
statistics for th
An overall vacu
Given the size c
changes effective.

en vacuumed due to the high
o make the gathering of
nfiguration

Communication
The intervention

dexes because there are many which have never been used since the restart of
to be maintained when transactions occur

thanks, one point of att
compon

itecture
provide pg_bouncer centrally for
important ones that

atm our offer is ba
it is worth testing
if it is not useful I … ussion to evaluate other options
(perhaps in the m … ST running pgbouncer on your own?) Edited

it coul … a 200+
conne

I remem
applications such as
could benefit from it.

there are
source for the increase in the data Edited

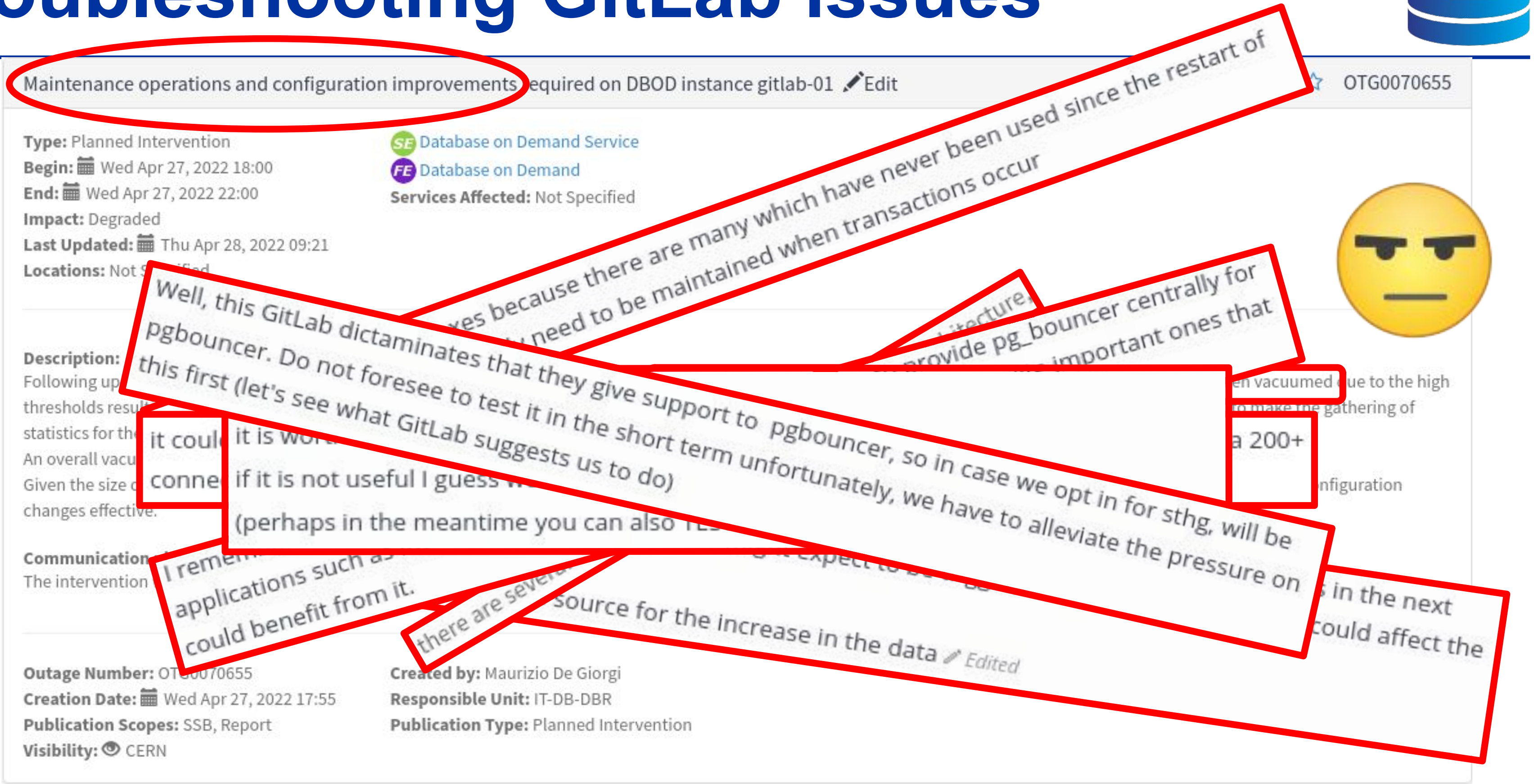
expect to be triggered
pooling

this in the next
nd could affect the

Outage Number: OTG0070655
Creation Date: Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

Troubleshooting GitLab issues



Maintenance operations and configuration improvements equired on DBOD instance gitlab-01 ✎Edit ☆ OTG0070655

Type: Planned Intervention
Begin: Wed Apr 27, 2022 18:00
End: Wed Apr 27, 2022 22:00
Impact: Degraded
Last Updated: Thu Apr 28, 2022 09:21
Locations: Not S

SE Database on Demand Service
FE Database on Demand
Services Affected: Not Specified

Description:
Following up … en vacuumed ue to the high
thresholds resu … o make the gathering of
statistics for th
An overall vacu
Given the size o … nfiguration
changes effective.

Communication
The intervention

Outage Number: OT 70655
Creation Date: Wed Apr 27, 2022 17:55
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Maurizio De Giorgi
Responsible Unit: IT-DB-DBR
Publication Type: Planned Intervention

es because there are many which have never been used since the restart of
need to be maintained when transactions occur

itecture,

provide pg_bouncer centrally for
important ones that

Well, this GitLab dictaminates that they give support to pgbouncer, so in case we opt in for sthg, will be
pgbouncer. Do not foresee to test it in the short term unfortunately, we have to alleviate the pressure on
this first (let's see what GitLab suggests us to do)

it coul
conne

it is wo
if it is not useful I guess
(perhaps in the meantime you can also TE

a 200+

I remem
applications such as
could benefit from it.

there are sever

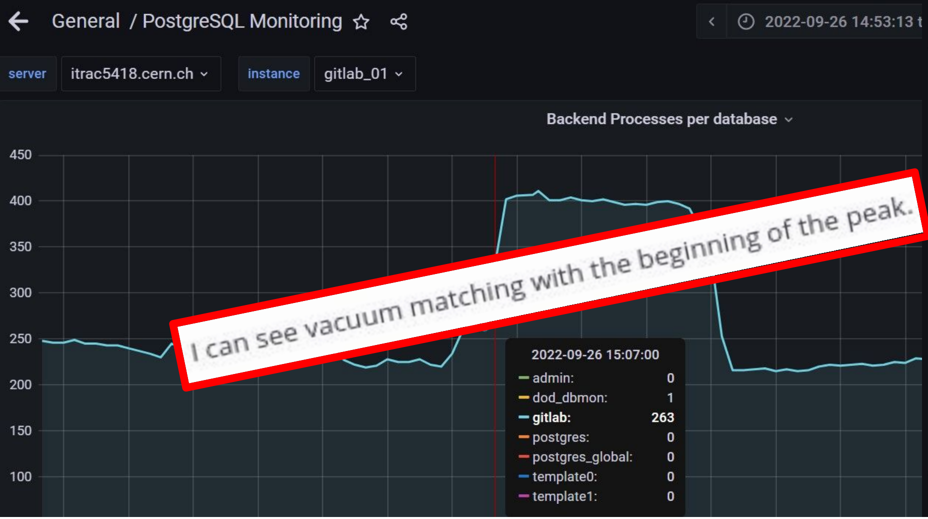
source for the increase in the data ✎ Edited

s in the next
could affect the

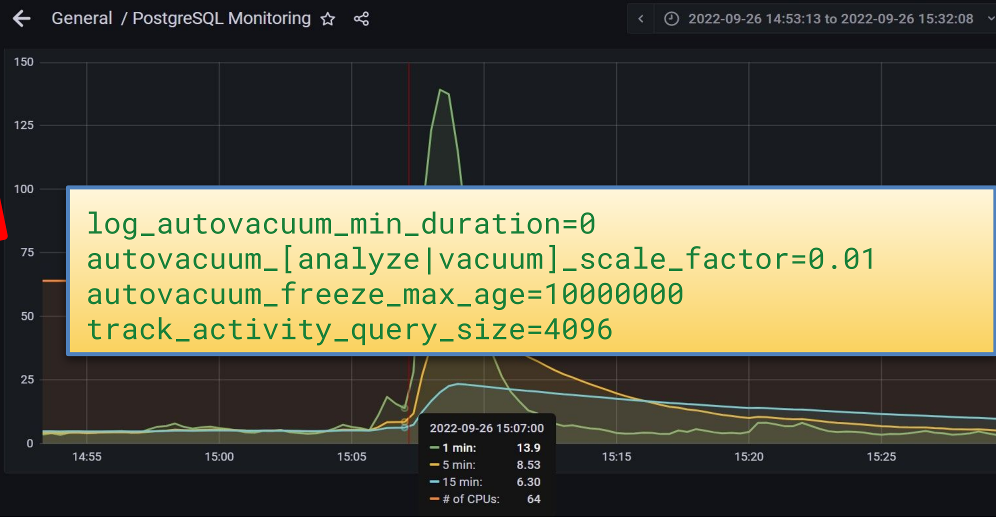
Troubleshooting GitLab issues

Back to square one... looking for a culprit

```
[2022-09-26 16:03:36.187 CEST][PID:163179][SID:6331b104.27d6b][DB:] LOG:  automatic analyze of table "gitlab.public.project_authorizations" system usage: CPU: user: 0.35 s, system: 0.44 s, elapsed: 47.54 s
[2022-09-26 16:04:06.221 CEST][PID:163566][SID:6331b14f.27eee][DB:] LOG:  automatic analyze of table "gitlab.public.namespaces" system usage: CPU: user: 0.50 s, system: 0.06 s, elapsed: 5.82 s
[2022-09-26 16:04:11.891 CEST][PID:163602][SID:6331b150.27f12][DB:gitlab] LOG:  duration: 10063.487 ms  bind <unnamed>: /*application:web,correlation_id:01GDX1S9VENZ19H288EKTR10R5,db_config_name:main*/
```



I can see vacuum matching with the beginning of the peak.

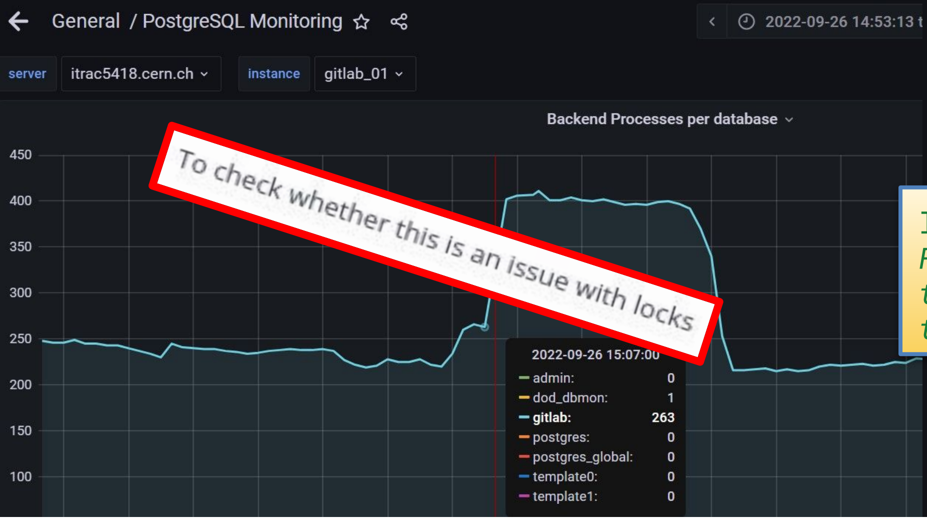


```
log_autovacuum_min_duration=0
autovacuum_[analyze|vacuum]_scale_factor=0.01
autovacuum_freeze_max_age=10000000
track_activity_query_size=4096
```

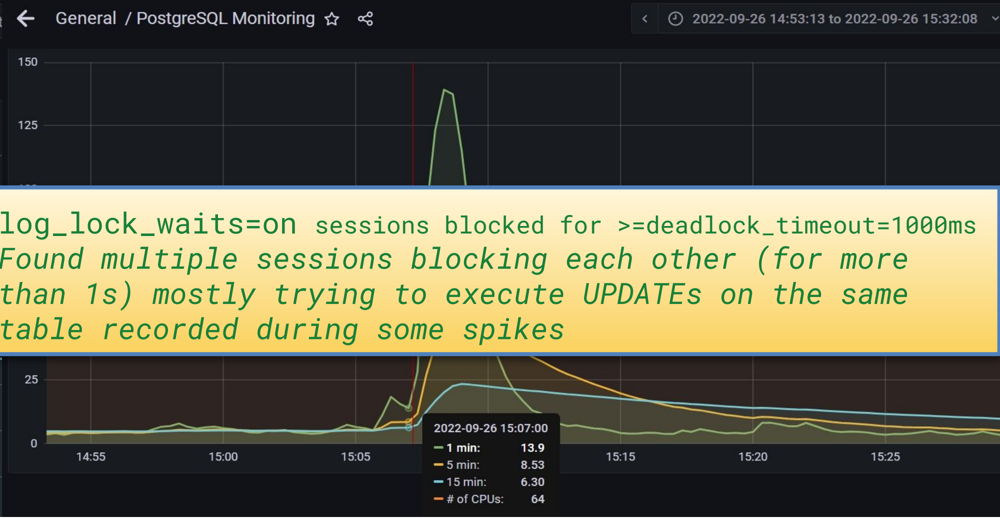
Troubleshooting GitLab issues

Back to square one... looking for a culprit

```
[2022-09-26 16:03:36.187 CEST][PID:163179][SID:6331b104.27d6b][DB:] LOG:  automatic analyze of table "gitlab.public.project_authorizations" system
usage: CPU: user: 0.35 s, system: 0.44 s, elapsed: 47.54 s
[2022-09-26 16:04:06.221 CEST][PID:163566][SID:6331b14f.27eee][DB:] LOG:  automatic analyze of table "gitlab.public.namespaces" system usage: CPU:
user: 0.50 s, system: 0.06 s, elapsed: 5.82 s
[2022-09-26 16:04:11.891 CEST][PID:163602][SID:6331b150.27f12][DB:gitlab] LOG:  duration: 10063.487 ms  bind <unnamed>:
/*application:web,correlation_id:01GDX1S9VENZ19H288EKTR10R5,db_config_name:main*/
```



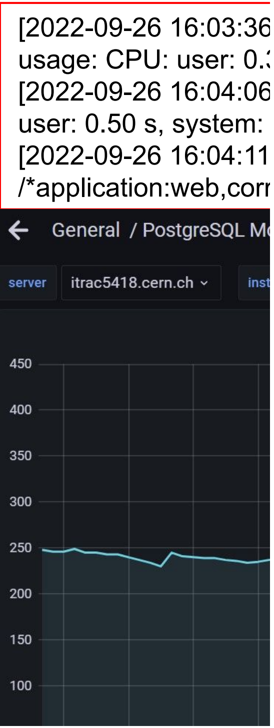
To check whether this is an issue with locks



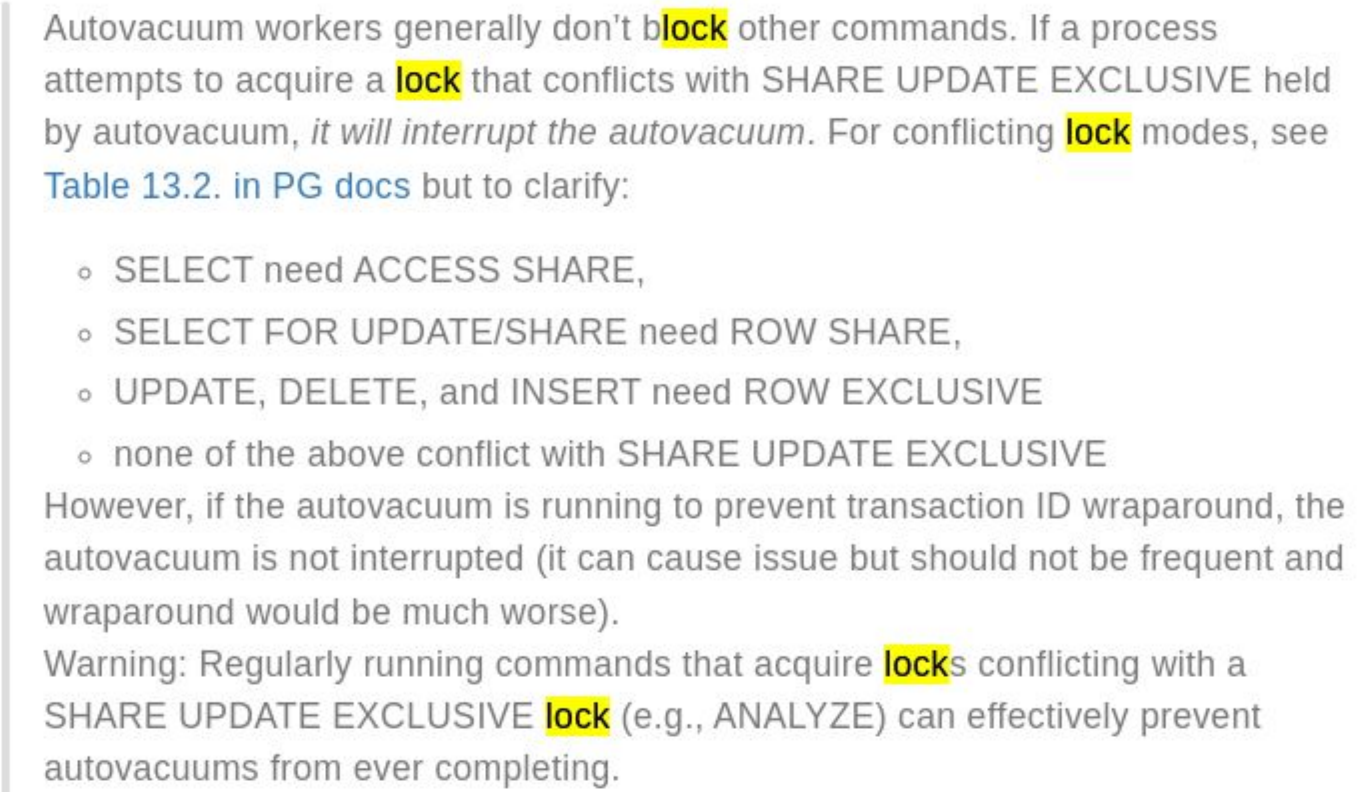
`log_lock_waits=on` sessions blocked for >=deadlock_timeout=1000ms

Found multiple sessions blocking each other (for more than 1s) mostly trying to execute UPDATEs on the same table recorded during some spikes

Troubleshooting GitLab issues



```
[2022-09-26 16:03:36
usage: CPU: user: 0.
[2022-09-26 16:04:06
user: 0.50 s, system:
[2022-09-26 16:04:11
/*application:web,corr
```



Autovacuum workers generally don't block other commands. If a process attempts to acquire a lock that conflicts with SHARE UPDATE EXCLUSIVE held by autovacuum, *it will interrupt the autovacuum*. For conflicting lock modes, see Table 13.2. in PG docs but to clarify:

- SELECT need ACCESS SHARE,
- SELECT FOR UPDATE/SHARE need ROW SHARE,
- UPDATE, DELETE, and INSERT need ROW EXCLUSIVE
- none of the above conflict with SHARE UPDATE EXCLUSIVE

However, if the autovacuum is running to prevent transaction ID wraparound, the autovacuum is not interrupted (it can cause issue but should not be frequent and wraparound would be much worse).

Warning: Regularly running commands that acquire locks conflicting with a SHARE UPDATE EXCLUSIVE lock (e.g., ANALYZE) can effectively prevent autovacuums from ever completing.

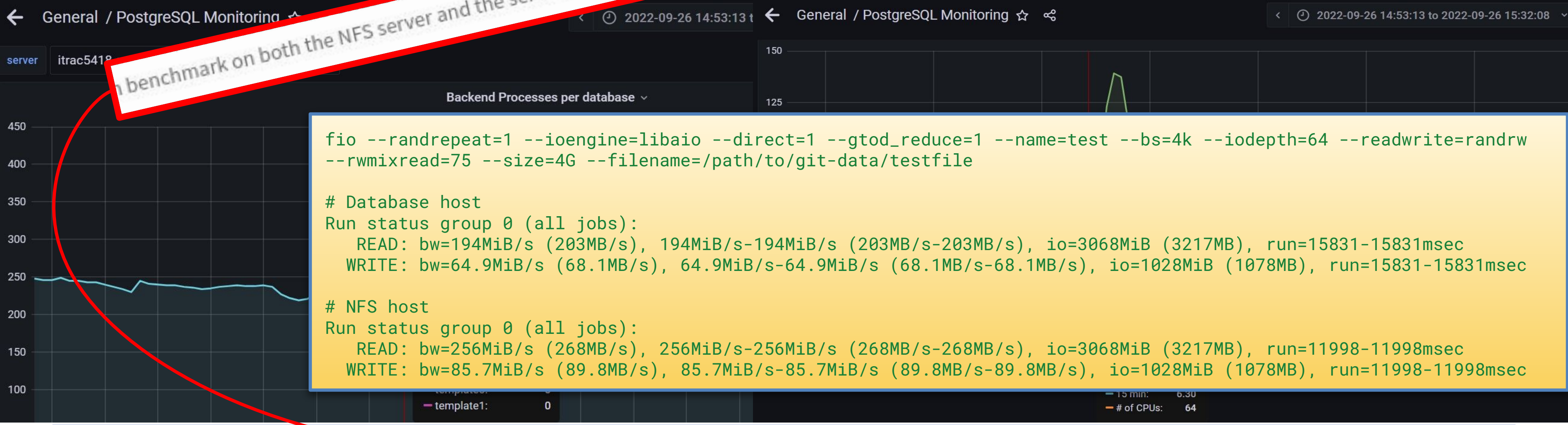
```
ations" system
stem usage: CPU:
```

Troubleshooting GitLab issues

Back to square one... looking for a culprit

```
[2022-09-26 16:03:36.187 CEST][PID:163179][SID:6331b104.27d6b][DB:] LOG:  automatic analyze of table "gitlab.public.project_authorizations" system
usage: CPU: user: 0.35 s, system: 0.44 s, elapsed: 47.54 s
[2022-09-26 16:04:06.221 CEST][PID:163566][SID:6331b14f.27eee][DB:] LOG:  f table "gitlab.public.namespaces" system usage: CPU:
user: 0.50 s, system: 0.06 s, elapsed: 5.82 s
[2022-09-26 16:04:11.891 CEST][PID:163602][SID:6331b15 G:  duration: 10063.487 ms  bind <unnamed>:
/*application:web,correlation_id:01GDX1S9VEN b_config_name:main*/
```

n benchmark on both the NFS server and the server that hosts the database with the NFS mounts,



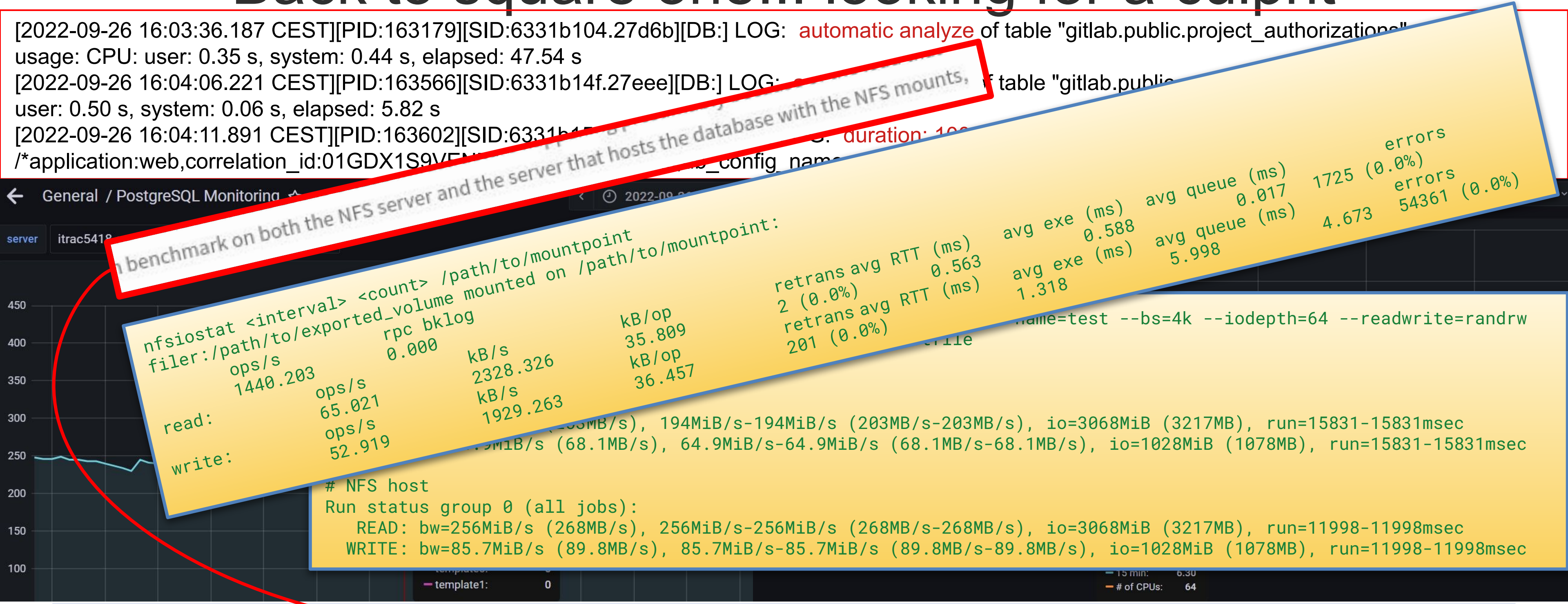
General / PostgreSQL Monitoring
2022-09-26 14:53:13 t
server
itrac5418
Backend Processes per database
450
400
350
300
250
200
150
100
template1: 0
General / PostgreSQL Monitoring
2022-09-26 14:53:13 to 2022-09-26 15:32:08
150
125
15 min: 6.30
# of CPUs: 64

```
fio --randrepeat=1 --ioengine=libaio --direct=1 --gtod_reduce=1 --name=test --bs=4k --iodepth=64 --readwrite=randrw
--rwmixread=75 --size=4G --filename=/path/to/git-data/testfile

# Database host
Run status group 0 (all jobs):
   READ: bw=194MiB/s (203MB/s), 194MiB/s-194MiB/s (203MB/s-203MB/s), io=3068MiB (3217MB), run=15831-15831msec
  WRITE: bw=64.9MiB/s (68.1MB/s), 64.9MiB/s-64.9MiB/s (68.1MB/s-68.1MB/s), io=1028MiB (1078MB), run=15831-15831msec

# NFS host
Run status group 0 (all jobs):
   READ: bw=256MiB/s (268MB/s), 256MiB/s-256MiB/s (268MB/s-268MB/s), io=3068MiB (3217MB), run=11998-11998msec
  WRITE: bw=85.7MiB/s (89.8MB/s), 85.7MiB/s-85.7MiB/s (89.8MB/s-89.8MB/s), io=1028MiB (1078MB), run=11998-11998msec
```

fio benchmarking: https://docs.gitlab.com/ee/administration/operations/filesystem_benchmarking.html

Troubleshooting GitLab issues

Back to square one... looking for a culprit



```
[2022-09-26 16:03:36.187 CEST][PID:163179][SID:6331b104.27d6b][DB:] LOG:  automatic analyze of table "gitlab.public.project_authorizations"
usage: CPU: user: 0.35 s, system: 0.44 s, elapsed: 47.54 s
[2022-09-26 16:04:06.221 CEST][PID:163566][SID:6331b14f.27eee][DB:] LOG: f table "gitlab.public
user: 0.50 s, system: 0.06 s, elapsed: 5.82 s
[2022-09-26 16:04:11.891 CEST][PID:163602][SID:6331b15 G:  duration: 10
/*application:web,correlation_id:01GDX1S9VEN b_config_name
```

n benchmark on both the NFS server and the server that hosts the database with the NFS mounts,

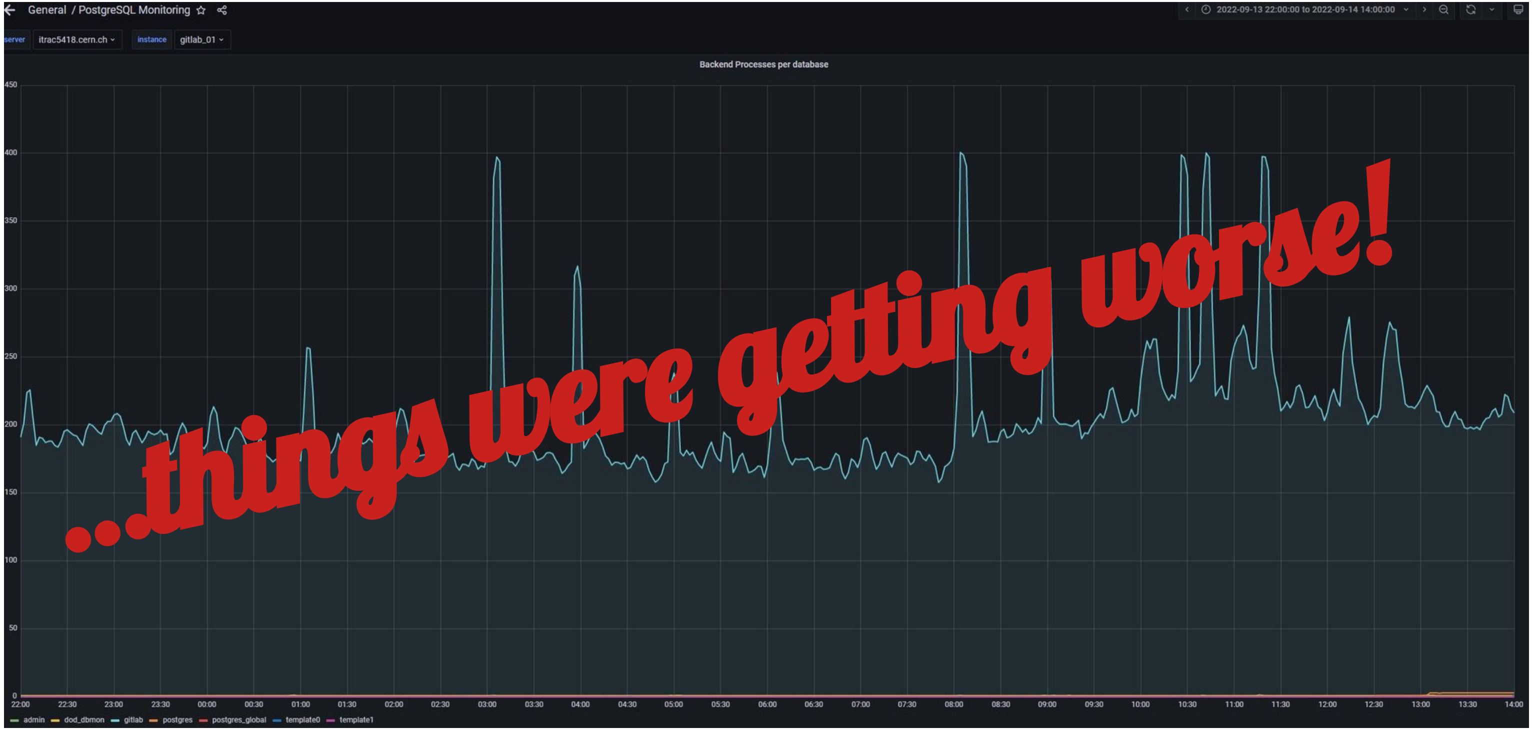
```
nfsiostat <interval> <count> /path/to/mountpoint
filer:/path/to/exported_volume mounted on /path/to/mountpoint:
           ops/s       rpc bklog
        1440.203           0.000

read:      ops/s            kB/s           kB/op         retrans    avg RTT (ms)    avg exe (ms)  avg queue (ms)          errors
          65.021        2328.326          35.809        2 (0.0%)           0.563           0.588           0.017     1725 (0.0%)
write:     ops/s            kB/s           kB/op         retrans    avg RTT (ms)    avg exe (ms)  avg queue (ms)          errors
          52.919        1929.263          36.457      201 (0.0%)           1.318           5.998           4.673    54361 (0.0%)
```

```
ame=test --bs=4k --iodepth=64 --readwrite=randrw
file

3MB/s), 194MiB/s-194MiB/s (203MB/s-203MB/s), io=3068MiB (3217MB), run=15831-15831msec
.9MiB/s (68.1MB/s), 64.9MiB/s-64.9MiB/s (68.1MB/s-68.1MB/s), io=1028MiB (1078MB), run=15831-15831msec

# NFS host
Run status group 0 (all jobs):
   READ: bw=256MiB/s (268MB/s), 256MiB/s-256MiB/s (268MB/s-268MB/s), io=3068MiB (3217MB), run=11998-11998msec
  WRITE: bw=85.7MiB/s (89.8MB/s), 85.7MiB/s-85.7MiB/s (89.8MB/s-89.8MB/s), io=1028MiB (1078MB), run=11998-11998msec
```

fio benchmarking: https://docs.gitlab.com/ee/administration/operations/filesystem_benchmarking.html

Troubleshooting GitLab issues

In the meantime…



General / PostgreSQL Monitoring

2022-09-13 22:00:00 to 2022-09-14 14:00:00

server itrac5418.cern.ch

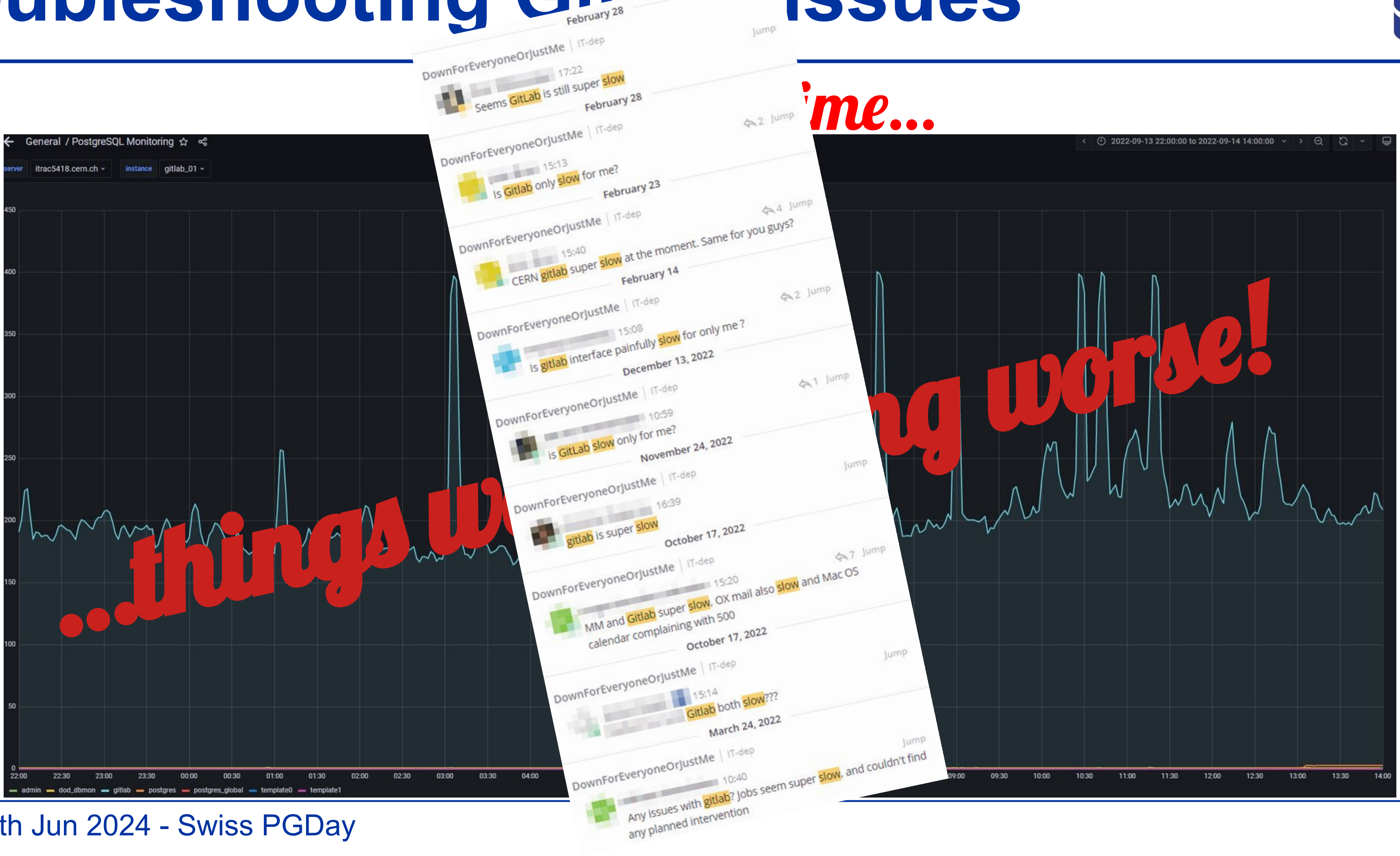
instance gitlab_01

Backend Processes per database

…things were getting worse!

Troubleshooting Git… issues

…me…



February 28

DownForEveryoneOrJustMe | IT-dep — Jump

[redacted] 17:22
Seems GitLab is still super slow

February 28

DownForEveryoneOrJustMe | IT-dep — 2 Jump

[redacted] 15:13
Is Gitlab only slow for me?

February 23

DownForEveryoneOrJustMe | IT-dep — 4 Jump

[redacted] 15:40
CERN gitlab super slow at the moment. Same for you guys?

February 14

DownForEveryoneOrJustMe | IT-dep — 2 Jump

[redacted] 15:08
Is gitlab interface painfully slow for only me ?

December 13, 2022

DownForEveryoneOrJustMe | IT-dep — 1 Jump

[redacted] 10:59
is GitLab slow only for me?

November 24, 2022

DownForEveryoneOrJustMe | IT-dep — Jump

[redacted] 16:39
gitlab is super slow

October 17, 2022

DownForEveryoneOrJustMe | IT-dep — 7 Jump

[redacted] 15:20
MM and Gitlab super slow, OX mail also slow and Mac OS calendar complaining with 500

October 17, 2022

DownForEveryoneOrJustMe | IT-dep — Jump

[redacted] 15:14
Gitlab both slow???

March 24, 2022

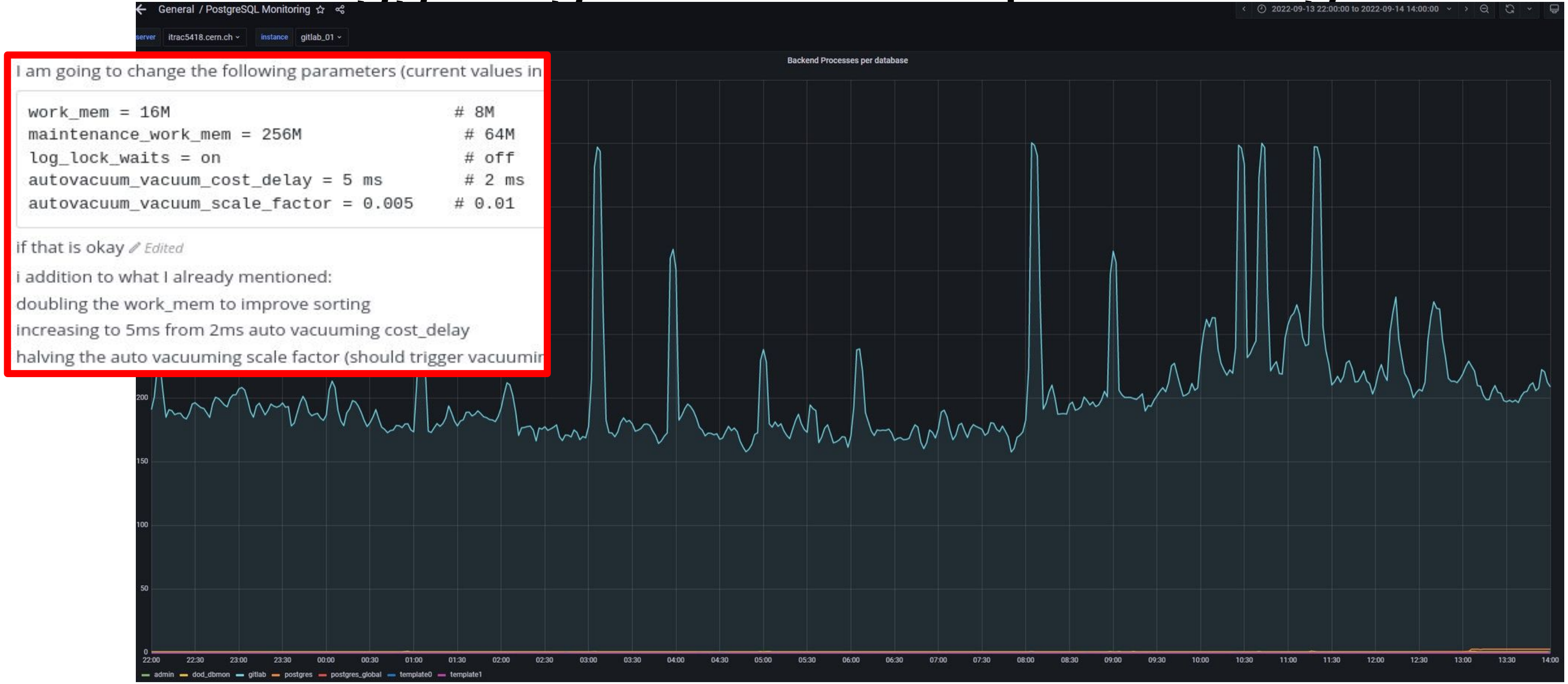
DownForEveryoneOrJustMe | IT-dep — Jump

[redacted] 10:40
Any issues with gitlab? Jobs seem super slow, and couldn't find any planned intervention

…things w… ng worse!

Troubleshooting GitLab issues

… triggering further attempts of mitigation



I am going to change the following parameters (current values in

```
work_mem = 16M                               # 8M
maintenance_work_mem = 256M                   # 64M
log_lock_waits = on                           # off
autovacuum_vacuum_cost_delay = 5 ms           # 2 ms
autovacuum_vacuum_scale_factor = 0.005      # 0.01
```

if that is okay *Edited*

i addition to what I already mentioned:

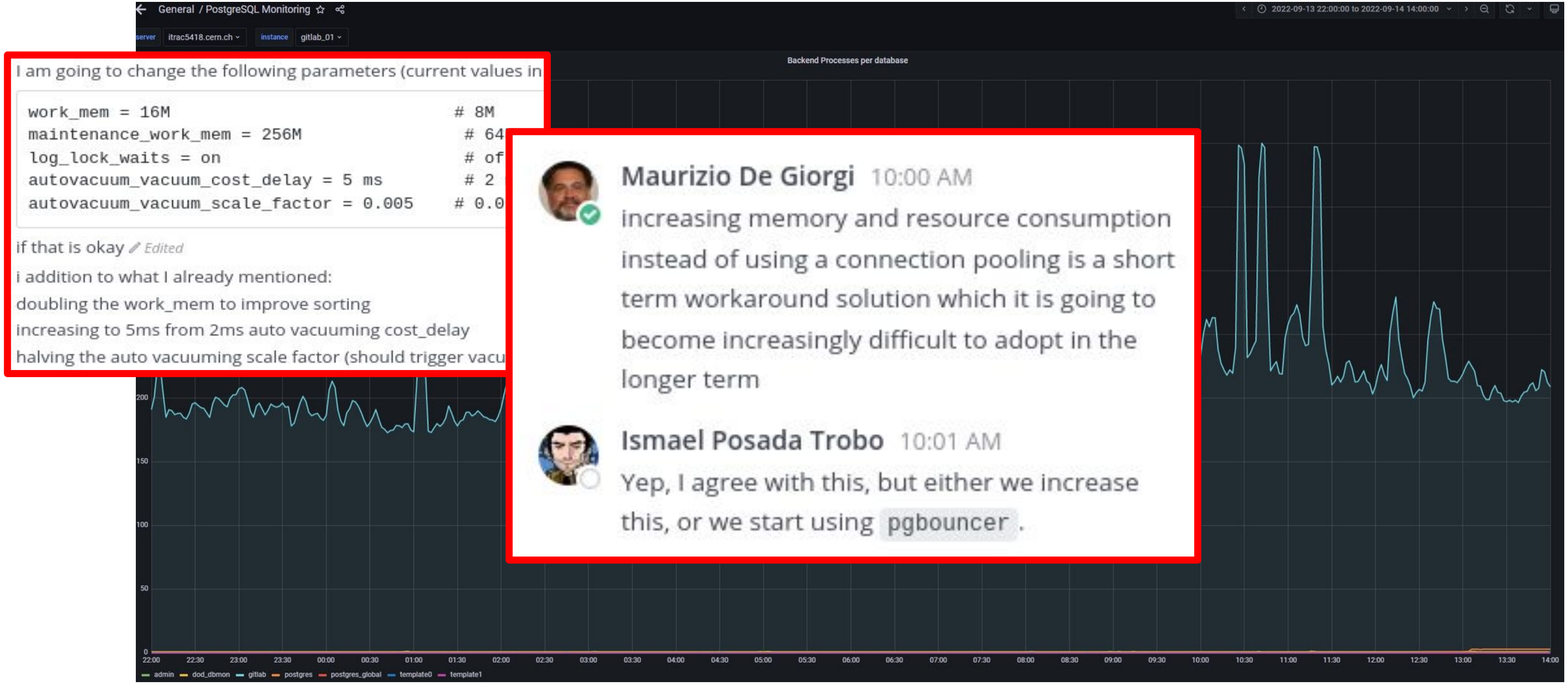
doubling the work_mem to improve sorting

increasing to 5ms from 2ms auto vacuuming cost_delay

halving the auto vacuuming scale factor (should trigger vacuumir

Troubleshooting GitLab issues

...based on *lateral* measures



I am going to change the following parameters (current values in

```
work_mem = 16M                                # 8M
maintenance_work_mem = 256M                    # 64
log_lock_waits = on                            # of
autovacuum_vacuum_cost_delay = 5 ms            # 2
autovacuum_vacuum_scale_factor = 0.005       # 0.0
```

if that is okay Edited

i addition to what I already mentioned:

doubling the work_mem to improve sorting

increasing to 5ms from 2ms auto vacuuming cost_delay

halving the auto vacuuming scale factor (should trigger vacu

Maurizio De Giorgi 10:00 AM

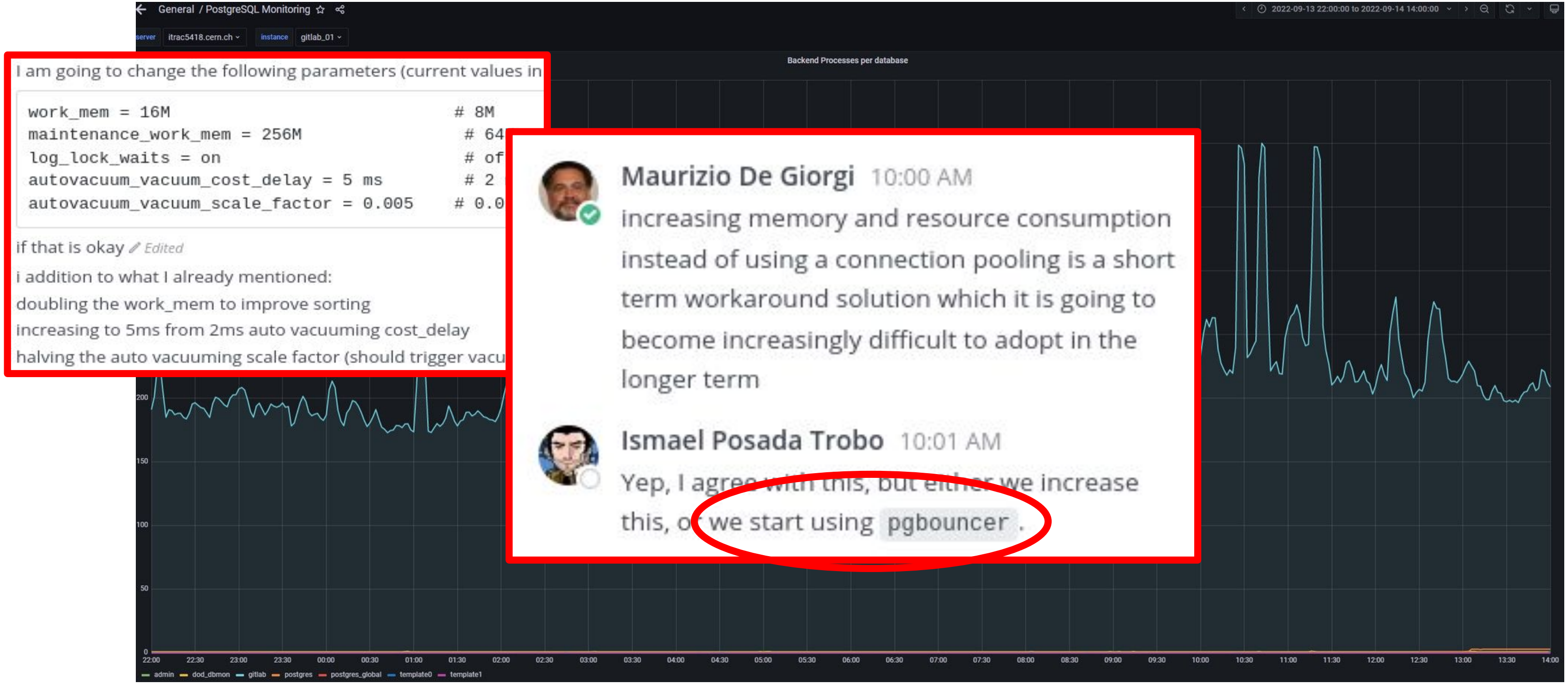
increasing memory and resource consumption instead of using a connection pooling is a short term workaround solution which it is going to become increasingly difficult to adopt in the longer term

Ismael Posada Trobo 10:01 AM

Yep, I agree with this, but either we increase this, or we start using `pgbouncer`.

Troubleshooting GitLab issues

...awareness started to come back



I am going to change the following parameters (current values in

```
work_mem = 16M                              # 8M
maintenance_work_mem = 256M                  # 64
log_lock_waits = on                          # of
autovacuum_vacuum_cost_delay = 5 ms          # 2
autovacuum_vacuum_scale_factor = 0.005      # 0.0
```

if that is okay *Edited*

i addition to what I already mentioned:

doubling the work_mem to improve sorting

increasing to 5ms from 2ms auto vacuuming cost_delay

halving the auto vacuuming scale factor (should trigger vacu

Maurizio De Giorgi 10:00 AM

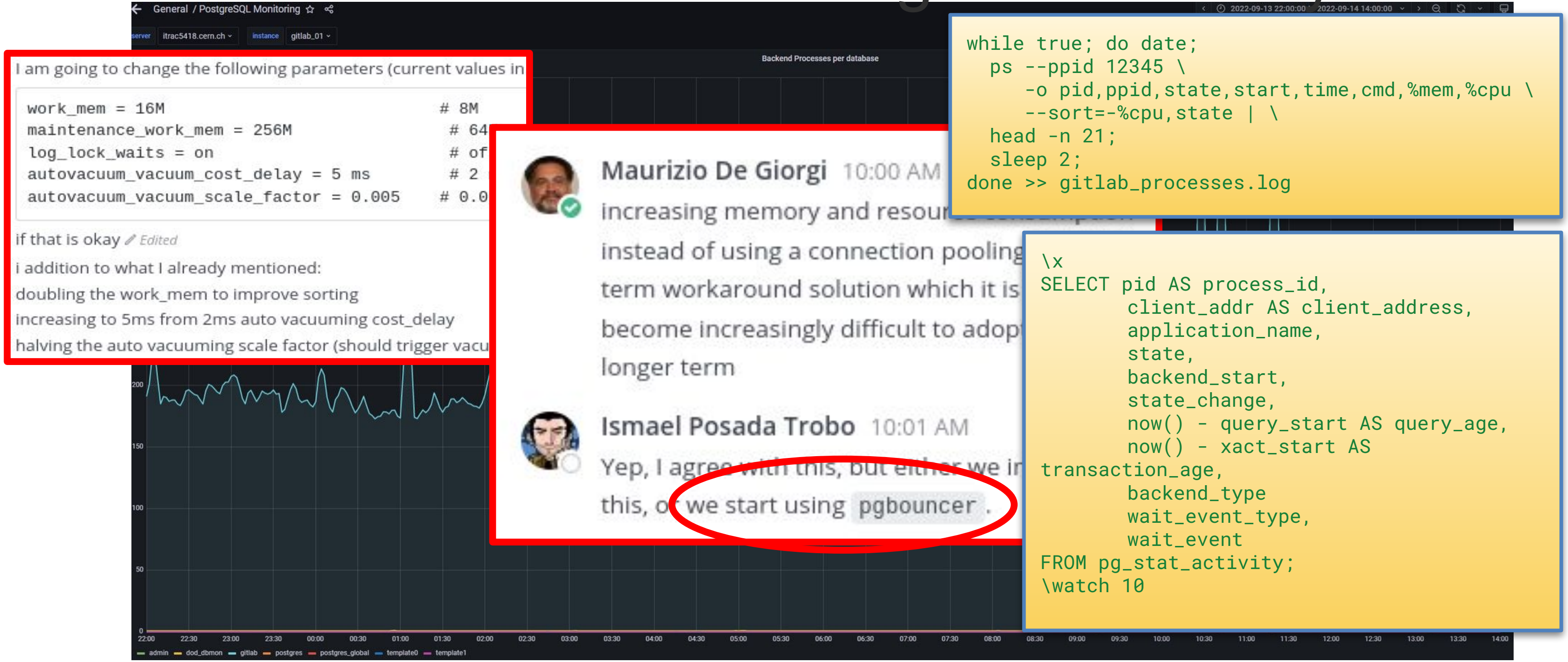
increasing memory and resource consumption instead of using a connection pooling is a short term workaround solution which it is going to become increasingly difficult to adopt in the longer term

Ismael Posada Trobo 10:01 AM

Yep, I agree with this, but either we increase this, or we start using `pgbouncer`.

Troubleshooting GitLab issues

…but with more diagnostic activity



I am going to change the following parameters (current values in

```
work_mem = 16M                            # 8M
maintenance_work_mem = 256M                # 64
log_lock_waits = on                        # of
autovacuum_vacuum_cost_delay = 5 ms        # 2
autovacuum_vacuum_scale_factor = 0.005    # 0.0
```

if that is okay ✎ *Edited*

i addition to what I already mentioned:

doubling the work_mem to improve sorting

increasing to 5ms from 2ms auto vacuuming cost_delay

halving the auto vacuuming scale factor (should trigger vacu

Maurizio De Giorgi 10:00 AM

increasing memory and resour

instead of using a connection pooling

term workaround solution which it is

become increasingly difficult to adop

longer term

Ismael Posada Trobo 10:01 AM

Yep, I agree with this, but either we i

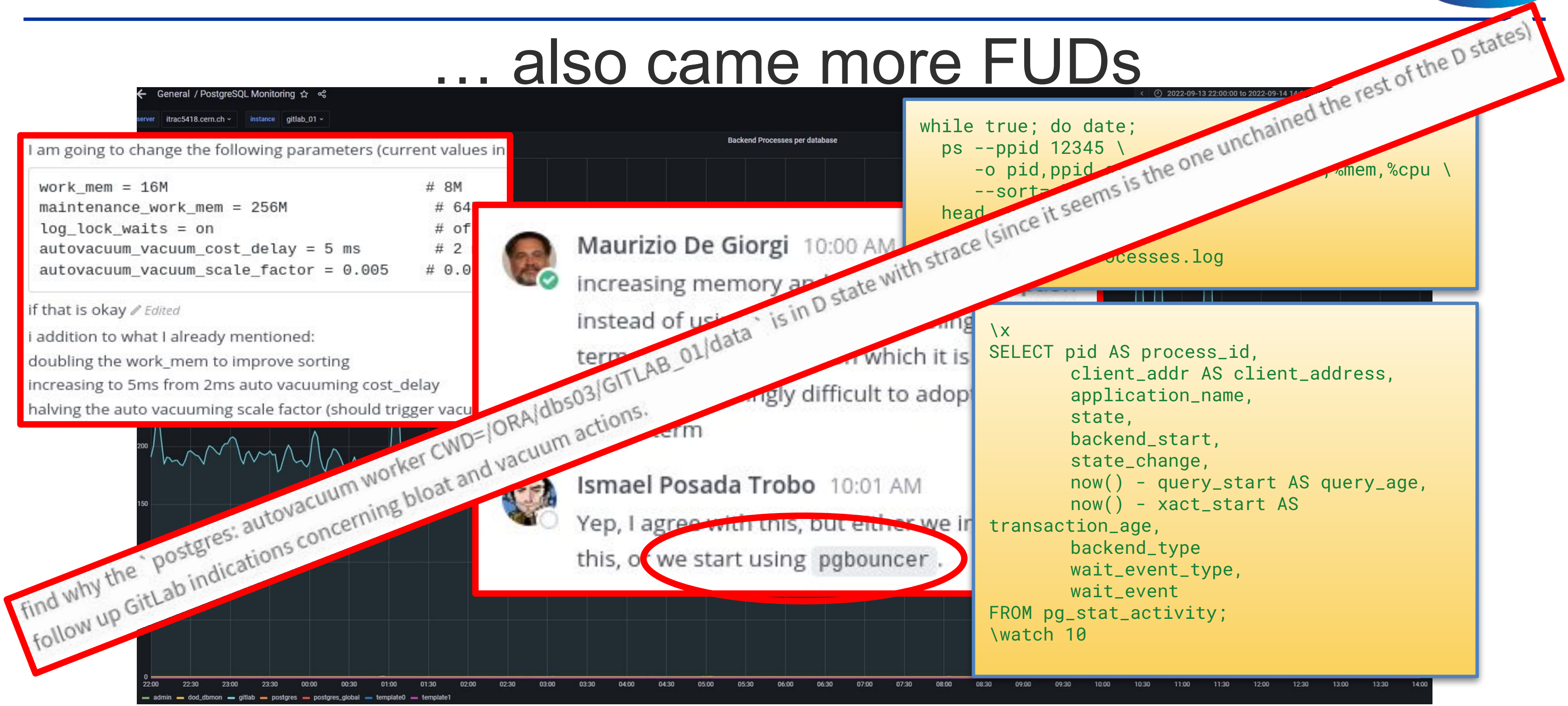
this, or we start using `pgbouncer`.

```
while true; do date;
  ps --ppid 12345 \
     -o pid,ppid,state,start,time,cmd,%mem,%cpu \
     --sort=-%cpu,state | \
  head -n 21;
  sleep 2;
done >> gitlab_processes.log
```

```
\x
SELECT pid AS process_id,
       client_addr AS client_address,
       application_name,
       state,
       backend_start,
       state_change,
       now() - query_start AS query_age,
       now() - xact_start AS
transaction_age,
       backend_type
       wait_event_type,
       wait_event
FROM pg_stat_activity;
\watch 10
```

Troubleshooting GitLab issues

… also came more FUDs



I am going to change the following parameters (current values in

```
work_mem = 16M                               # 8M
maintenance_work_mem = 256M                   # 64
log_lock_waits = on                           # of
autovacuum_vacuum_cost_delay = 5 ms           # 2
autovacuum_vacuum_scale_factor = 0.005       # 0.0
```

if that is okay *Edited*

i addition to what I already mentioned:

doubling the work_mem to improve sorting

increasing to 5ms from 2ms auto vacuuming cost_delay

halving the auto vacuuming scale factor (should trigger vacu

Maurizio De Giorgi 10:00 AM

increasing memory an

instead of usi

term which it is

ngly difficult to adopt

term

Ismael Posada Trobo 10:01 AM

Yep, I agree with this, but either we in

this, or we start using pgbouncer .

find why the ` postgres: autovacuum worker CWD=/ORA/dbs03/GITLAB_01/data ` is in D state with strace (since it seems is the one unchained the rest of the D states)

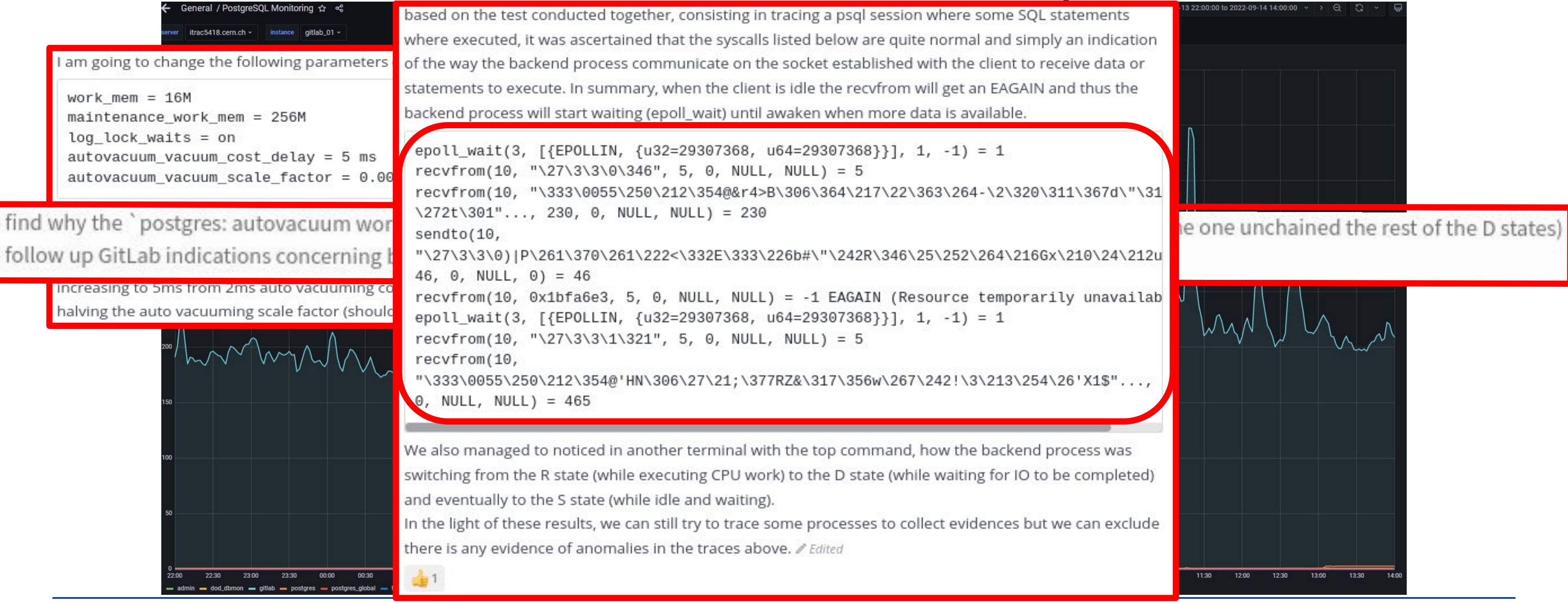
follow up GitLab indications concerning bloat and vacuum actions.

```
while true; do date;
  ps --ppid 12345 \
     -o pid,ppid                      ,%mem,%cpu \
     --sort=
  head

                   ocesses.log
```

```
\x
SELECT pid AS process_id,
       client_addr AS client_address,
       application_name,
       state,
       backend_start,
       state_change,
       now() - query_start AS query_age,
       now() - xact_start AS
transaction_age,
       backend_type
       wait_event_type,
       wait_event
FROM pg_stat_activity;
\watch 10
```

Troubleshooting GitLab issues

…which we had to analyze



I am going to change the following parameters

```
work_mem = 16M
maintenance_work_mem = 256M
log_lock_waits = on
autovacuum_vacuum_cost_delay = 5 ms
autovacuum_vacuum_scale_factor = 0.00
```

increasing to 5ms from 2ms auto vacuuming co

halving the auto vacuuming scale factor (should

find why the `postgres: autovacuum wor

follow up GitLab indications concerning b

based on the test conducted together, consisting in tracing a psql session where some SQL statements where executed, it was ascertained that the syscalls listed below are quite normal and simply an indication of the way the backend process communicate on the socket established with the client to receive data or statements to execute. In summary, when the client is idle the recvfrom will get an EAGAIN and thus the backend process will start waiting (epoll_wait) until awaken when more data is available.

```
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(10, "\27\3\3\0\346", 5, 0, NULL, NULL) = 5
recvfrom(10, "\333\0055\250\212\354@&r4>B\306\364\217\22\363\264-\2\320\311\367d\"\31
\272t\301"..., 230, 0, NULL, NULL) = 230
sendto(10,
"\27\3\3\0)|P\261\370\261\222<\332E\333\226b#\"\242R\346\25\252\264\216Gx\210\24\212u
46, 0, NULL, 0) = 46
recvfrom(10, 0x1bfa6e3, 5, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailab
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(10, "\27\3\3\1\321", 5, 0, NULL, NULL) = 5
recvfrom(10,
"\333\0055\250\212\354@'HN\306\27\21;\377RZ&\317\356w\267\242!\3\213\254\26'X1$"...,
0, NULL, NULL) = 465
```

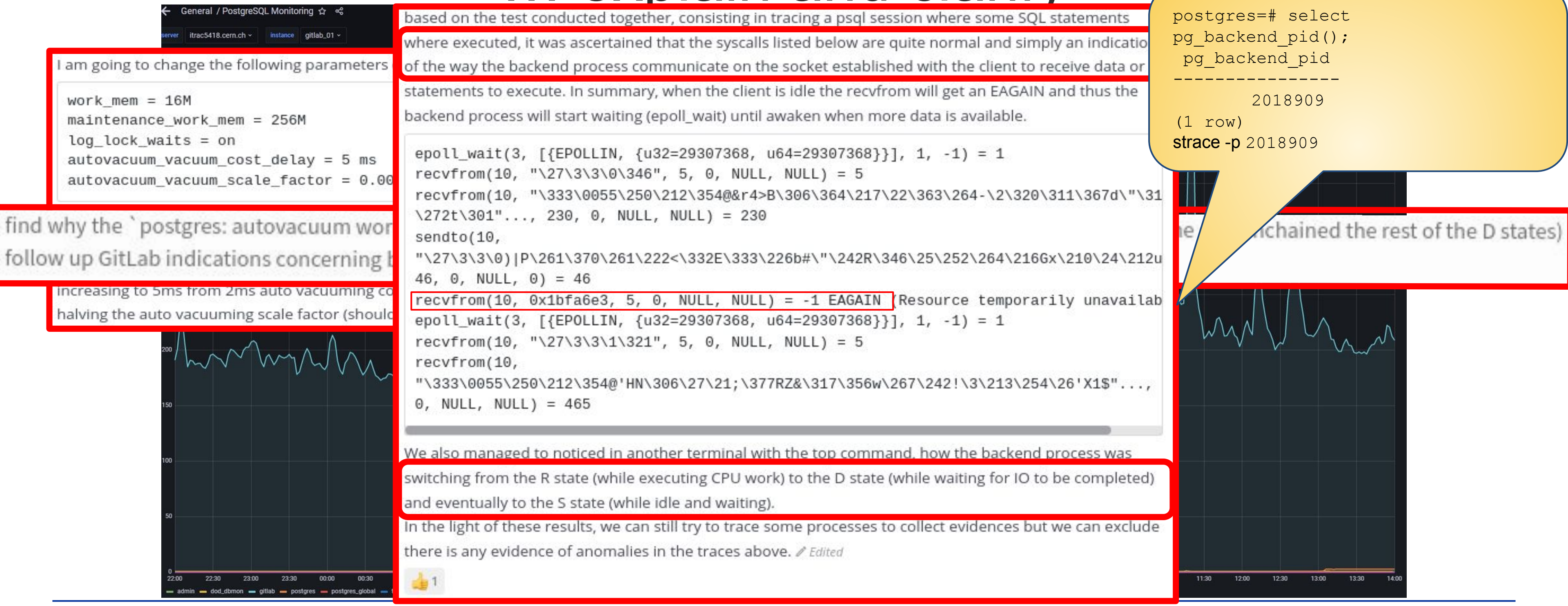
We also managed to noticed in another terminal with the top command, how the backend process was switching from the R state (while executing CPU work) to the D state (while waiting for IO to be completed) and eventually to the S state (while idle and waiting).

In the light of these results, we can still try to trace some processes to collect evidences but we can exclude there is any evidence of anomalies in the traces above.

e one unchained the rest of the D states)

Troubleshooting GitLab issues

… explain and clarify



I am going to change the following parameters

```
work_mem = 16M
maintenance_work_mem = 256M
log_lock_waits = on
autovacuum_vacuum_cost_delay = 5 ms
autovacuum_vacuum_scale_factor = 0.00
```

increasing to 5ms from 2ms auto vacuuming co

halving the auto vacuuming scale factor (should

find why the `postgres: autovacuum wor

follow up GitLab indications concerning I

based on the test conducted together, consisting in tracing a psql session where some SQL statements where executed, it was ascertained that the syscalls listed below are quite normal and simply an indicatio of the way the backend process communicate on the socket established with the client to receive data or statements to execute. In summary, when the client is idle the recvfrom will get an EAGAIN and thus the backend process will start waiting (epoll_wait) until awaken when more data is available.

```
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(10, "\27\3\3\0\346", 5, 0, NULL, NULL) = 5
recvfrom(10, "\333\0055\250\212\354@&r4>B\306\364\217\22\363\264-\2\320\311\367d\"\31
\272t\301"..., 230, 0, NULL, NULL) = 230
sendto(10,
"\27\3\3\0)|P\261\370\261\222<\332E\333\226b#\"\242R\346\25\252\264\216Gx\210\24\212u
46, 0, NULL, 0) = 46
recvfrom(10, 0x1bfa6e3, 5, 0, NULL, NULL) = -1 EAGAIN (Resource temporarily unavailab
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(10, "\27\3\3\1\321", 5, 0, NULL, NULL) = 5
recvfrom(10,
"\333\0055\250\212\354@'HN\306\27\21;\377RZ&\317\356w\267\242!\3\213\254\26'X1$"...,
0, NULL, NULL) = 465
```

We also managed to noticed in another terminal with the top command, how the backend process was switching from the R state (while executing CPU work) to the D state (while waiting for IO to be completed) and eventually to the S state (while idle and waiting).

In the light of these results, we can still try to trace some processes to collect evidences but we can exclude there is any evidence of anomalies in the traces above. Edited

👍 1

he

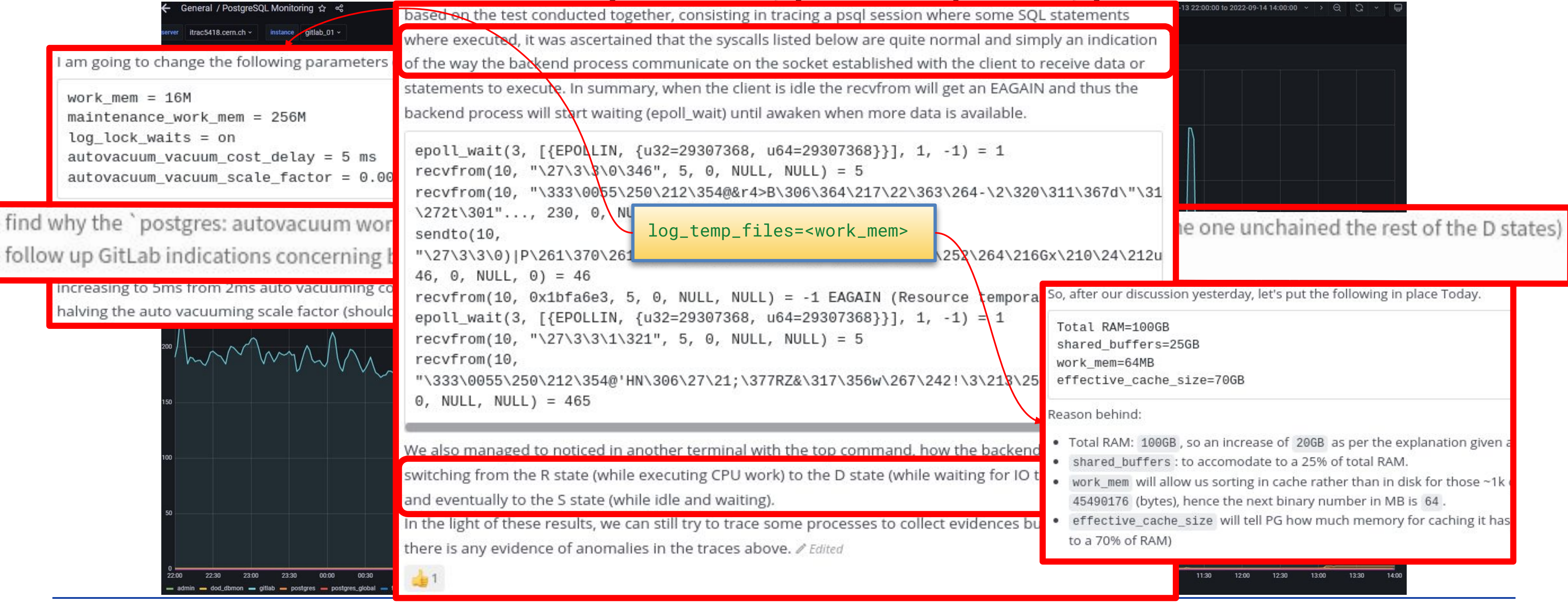
chained the rest of the D states)

```
postgres=# select
pg_backend_pid();
 pg_backend_pid
----------------
        2018909
(1 row)
```

strace -p 2018909

Troubleshooting GitLab issues

...while improving everything else



I am going to change the following parameters

```
work_mem = 16M
maintenance_work_mem = 256M
log_lock_waits = on
autovacuum_vacuum_cost_delay = 5 ms
autovacuum_vacuum_scale_factor = 0.00
```

increasing to 5ms from 2ms auto vacuuming co

halving the auto vacuuming scale factor (should

find why the `postgres: autovacuum wor

follow up GitLab indications concerning b

based on the test conducted together, consisting in tracing a psql session where some SQL statements where executed, it was ascertained that the syscalls listed below are quite normal and simply an indication of the way the backend process communicate on the socket established with the client to receive data or statements to execute. In summary, when the client is idle the recvfrom will get an EAGAIN and thus the backend process will start waiting (epoll_wait) until awaken when more data is available.

```
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(10, "\27\3\3\0\346", 5, 0, NULL, NULL) = 5
recvfrom(10, "\333\0055\250\212\354@&r4>B\306\364\217\22\363\264-\2\320\311\367d\"\31
\272t\301"..., 230, 0, NU
sendto(10,
"\27\3\3\0)|P\261\370\261                              \252\264\216Gx\210\24\212u
46, 0, NULL, 0) = 46
recvfrom(10, 0x1bfa6e3, 5, 0, NULL, NULL) = -1 EAGAIN (Resource tempora
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(10, "\27\3\3\1\321", 5, 0, NULL, NULL) = 5
recvfrom(10,
"\333\0055\250\212\354@'HN\306\27\21;\377RZ&\317\356w\267\242!\3\213\25
0, NULL, NULL) = 465
```

log_temp_files=<work_mem>

We also managed to noticed in another terminal with the top command, how the backend switching from the R state (while executing CPU work) to the D state (while waiting for IO t and eventually to the S state (while idle and waiting).

In the light of these results, we can still try to trace some processes to collect evidences bu there is any evidence of anomalies in the traces above. Edited

👍 1

e one unchained the rest of the D states)

So, after our discussion yesterday, let's put the following in place Today.

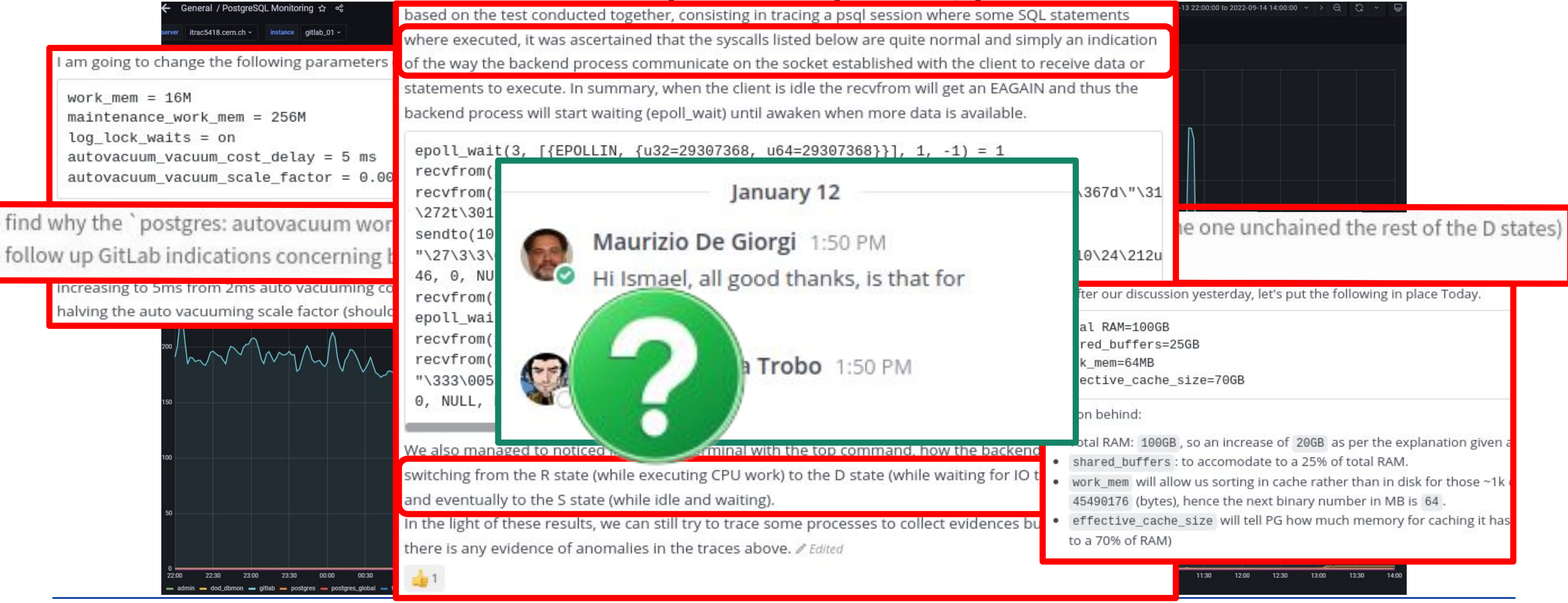
```
Total RAM=100GB
shared_buffers=25GB
work_mem=64MB
effective_cache_size=70GB
```

Reason behind:

- Total RAM: `100GB`, so an increase of `20GB` as per the explanation given a
- `shared_buffers`: to accomodate to a 25% of total RAM.
- `work_mem` will allow us sorting in cache rather than in disk for those ~1k `45490176` (bytes), hence the next binary number in MB is `64`.
- `effective_cache_size` will tell PG how much memory for caching it has to a 70% of RAM)

Troubleshooting GitLab issues

...until one day everything was clear!



I am going to change the following parameters

```
work_mem = 16M
maintenance_work_mem = 256M
log_lock_waits = on
autovacuum_vacuum_cost_delay = 5 ms
autovacuum_vacuum_scale_factor = 0.00
```

find why the `postgres: autovacuum wor

follow up GitLab indications concerning l

increasing to 5ms from 2ms auto vacuuming co

halving the auto vacuuming scale factor (should

based on the test conducted together, consisting in tracing a psql session where some SQL statements where executed, it was ascertained that the syscalls listed below are quite normal and simply an indication of the way the backend process communicate on the socket established with the client to receive data or statements to execute. In summary, when the client is idle the recvfrom will get an EAGAIN and thus the backend process will start waiting (epoll_wait) until awaken when more data is available.

```
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(
recvfrom(                                                   \367d\"\31
\272t\301
sendto(10
"\27\3\3\                                                   10\24\212u
46, 0, NU
recvfrom(
epoll_wai
recvfrom(
recvfrom(
"\333\005
0, NULL,
```

Hi Ismael, all good thanks, is that for

We also managed to noticed ... rminal with the top command, how the backend switching from the R state (while executing CPU work) to the D state (while waiting for IO t and eventually to the S state (while idle and waiting).

In the light of these results, we can still try to trace some processes to collect evidences bu there is any evidence of anomalies in the traces above. Edited

👍 1

e one unchained the rest of the D states)

fter our discussion yesterday, let's put the following in place Today.

```
al RAM=100GB
red_buffers=25GB
k_mem=64MB
ective_cache_size=70GB
```

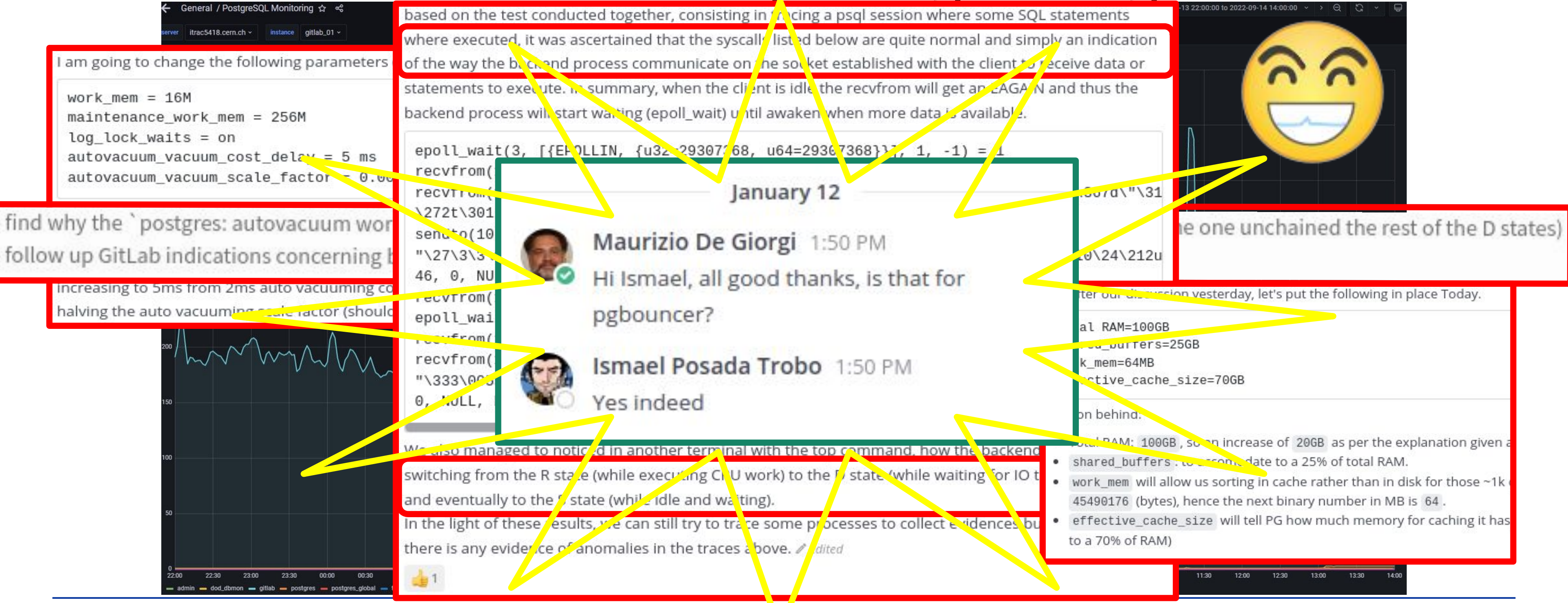
on behind:

otal RAM: 100GB, so an increase of 20GB as per the explanation given a

- shared_buffers: to accomodate to a 25% of total RAM.
- work_mem will allow us sorting in cache rather than in disk for those ~1k 45490176 (bytes), hence the next binary number in MB is 64.
- effective_cache_size will tell PG how much memory for caching it has to a 70% of RAM)

Troubleshooting GitLab issues

...and the connection pooling testing started!



I am going to change the following parameters

```
work_mem = 16M
maintenance_work_mem = 256M
log_lock_waits = on
autovacuum_vacuum_cost_delay = 5 ms
autovacuum_vacuum_scale_factor = 0.00
```

increasing to 5ms from 2ms auto vacuuming co

halving the auto vacuuming scale factor (should

find why the `postgres: autovacuum wor

follow up GitLab indications concerning b

based on the test conducted together, consisting in tracing a psql session where some SQL statements where executed, it was ascertained that the syscalls listed below are quite normal and simply an indication of the way the backend process communicate on the socket established with the client to receive data or statements to execute. In summary, when the client is idle the recvfrom will get an EAGAIN and thus the backend process will start waiting (epoll_wait) until awaken when more data is available.

```
epoll_wait(3, [{EPOLLIN, {u32=29307368, u64=29307368}}], 1, -1) = 1
recvfrom(
recvfrom(                                                   307d\"\31
\272t\301
sendto(10
"\27\3\3                                                    0\24\212u
46, 0, NU
recvfrom(
epoll_wai
recvfrom(
recvfrom(
"\333\003
0, NULL,
```

We also managed to noticed in another terminal with the top command, how the backend switching from the R state (while executing CPU work) to the D state (while waiting for IO t and eventually to the S state (while idle and waiting).

In the light of these results, we can still try to trace some processes to collect evidences bu there is any evidence of anomalies in the traces above. edited

👍 1

January 12

Maurizio De Giorgi 1:50 PM

Hi Ismael, all good thanks, is that for pgbouncer?

Ismael Posada Trobo 1:50 PM

Yes indeed

he one unchained the rest of the D states)

fter our discussion yesterday, let's put the following in place Today.

```
al RAM=100GB
red_buffers=25GB
k_mem=64MB
ctive_cache_size=70GB
```

on behind:

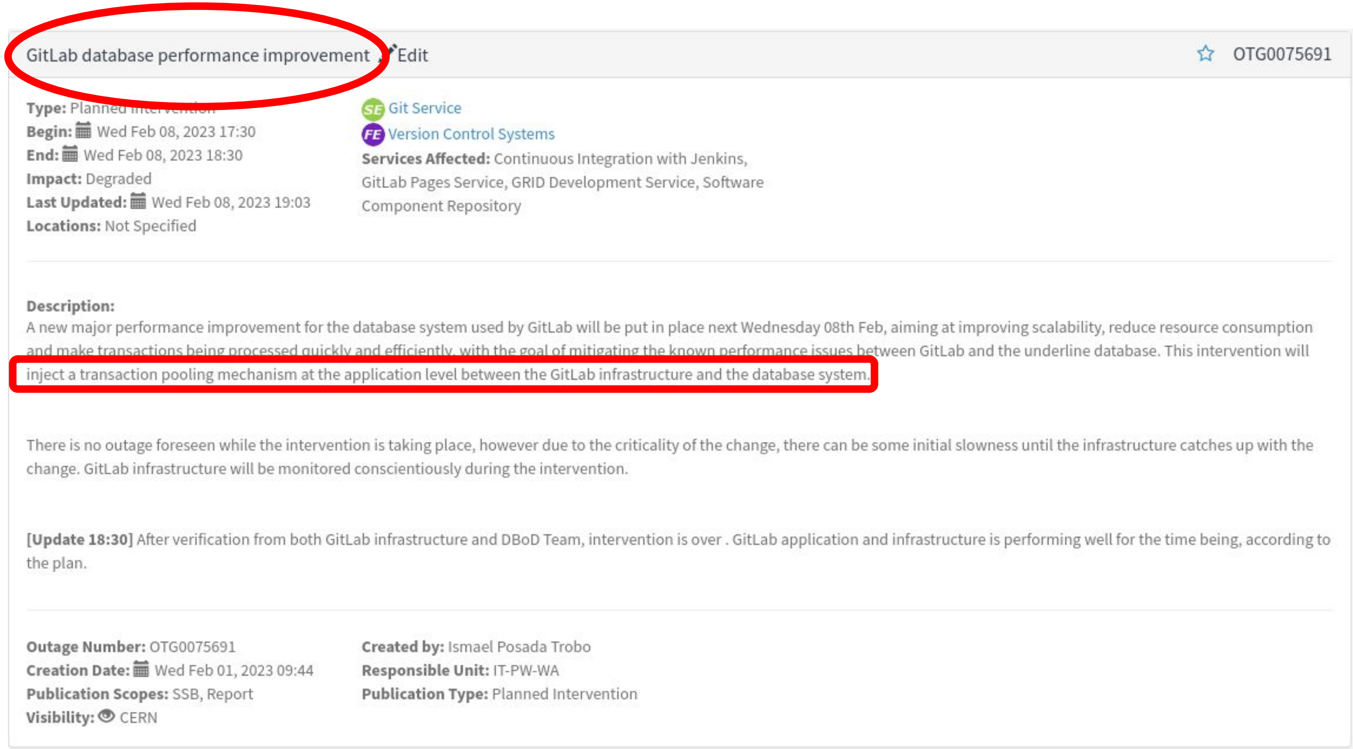
Total RAM: `100GB`, so an increase of `20GB` as per the explanation given a

- `shared_buffers`: to accomodate to a 25% of total RAM.
- `work_mem` will allow us sorting in cache rather than in disk for those ~1k `45490176` (bytes), hence the next binary number in MB is `64`.
- `effective_cache_size` will tell PG how much memory for caching it has to a 70% of RAM)

Agenda

- **An MVCC primer (boring things everyone knows but it is worth refreshing)**
    - (ACID) Transactions, Isolation, Concurrency, Serializable Snapshot Isolation
- **Connection scalability (showing the problem and its causes)**
    - Benchmarking & bottleneck analysis
- **Troubleshooting GitLab issues (talking about that time when we all had a lot of fun)**
    - The journey to enlightenment
    - The joy of enlightenment
- **The great effects of connection pooling on connection scalability**

Troubleshooting GitLab issues



GitLab database performance improvement Edit ☆ OTG0075691

Type: Planned Intervention
Begin: Wed Feb 08, 2023 17:30
End: Wed Feb 08, 2023 18:30
Impact: Degraded
Last Updated: Wed Feb 08, 2023 19:03
Locations: Not Specified

SE Git Service
FE Version Control Systems
Services Affected: Continuous Integration with Jenkins, GitLab Pages Service, GRID Development Service, Software Component Repository

Description:
A new major performance improvement for the database system used by GitLab will be put in place next Wednesday 08th Feb, aiming at improving scalability, reduce resource consumption and make transactions being processed quickly and efficiently, with the goal of mitigating the known performance issues between GitLab and the underline database. This intervention will inject a transaction pooling mechanism at the application level between the GitLab infrastructure and the database system.

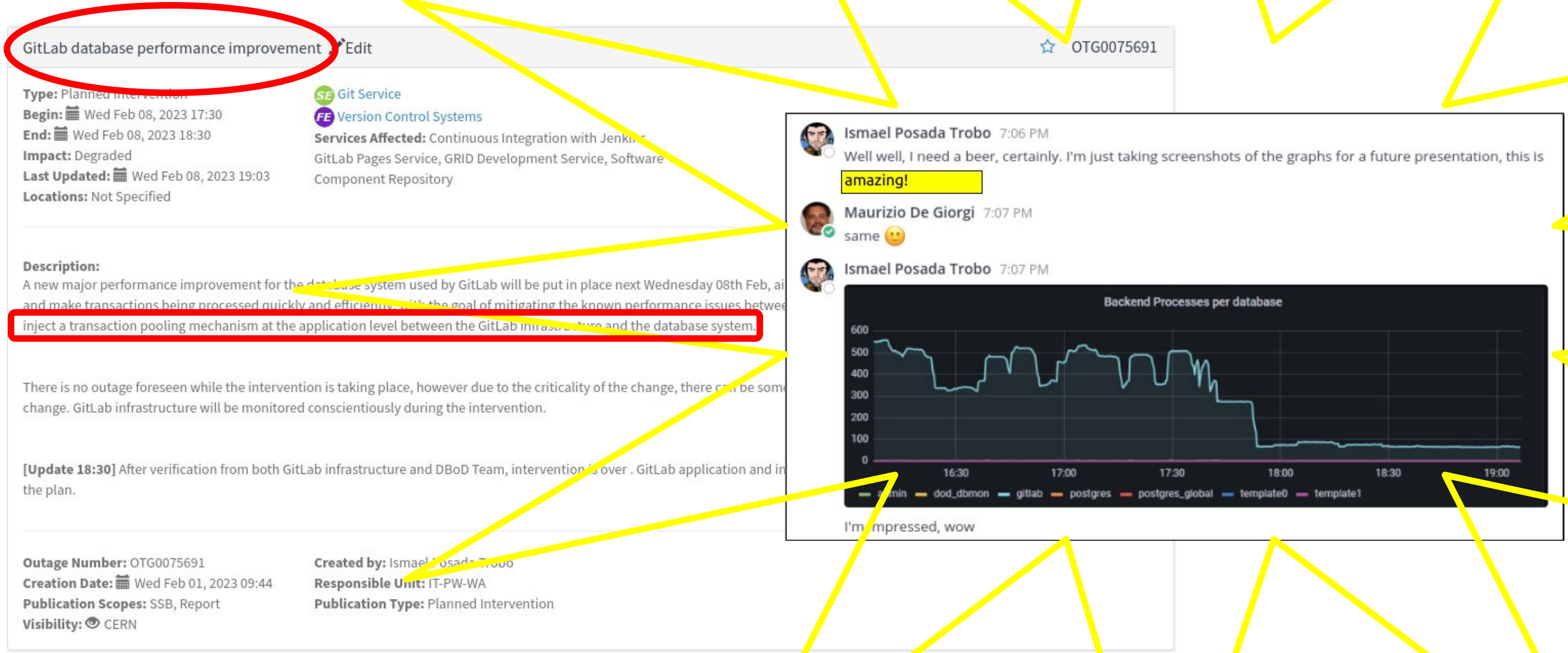
There is no outage foreseen while the intervention is taking place, however due to the criticality of the change, there can be some initial slowness until the infrastructure catches up with the change. GitLab infrastructure will be monitored conscientiously during the intervention.

[Update 18:30] After verification from both GitLab infrastructure and DBoD Team, intervention is over . GitLab application and infrastructure is performing well for the time being, according to the plan.

Outage Number: OTG0075691
Creation Date: Wed Feb 01, 2023 09:44
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Ismael Posada Trobo
Responsible Unit: IT-PW-WA
Publication Type: Planned Intervention

Troubleshooting GitLab issues



GitLab database performance improvement Edit

OTG0075691

Type: Planned intervention
Begin: Wed Feb 08, 2023 17:30
End: Wed Feb 08, 2023 18:30
Impact: Degraded
Last Updated: Wed Feb 08, 2023 19:03
Locations: Not Specified

Git Service
Version Control Systems
Services Affected: Continuous Integration with Jenkins, GitLab Pages Service, GRID Development Service, Software Component Repository

Description:
A new major performance improvement for the database system used by GitLab will be put in place next Wednesday 08th Feb, ai
and make transactions being processed quickly and efficiently, with the goal of mitigating the known performance issues betwee
inject a transaction pooling mechanism at the application level between the GitLab infrastructure and the database system.

There is no outage foreseen while the intervention is taking place, however due to the criticality of the change, there can be some
change. GitLab infrastructure will be monitored conscientiously during the intervention.

[Update 18:30] After verification from both GitLab infrastructure and DBoD Team, intervention is over . GitLab application and in
the plan.

Outage Number: OTG0075691
Creation Date: Wed Feb 01, 2023 09:44
Publication Scopes: SSB, Report
Visibility: CERN

Created by: Ismael Posada Trobo
Responsible Unit: IT-PW-WA
Publication Type: Planned Intervention

Ismael Posada Trobo 7:06 PM
Well well, I need a beer, certainly. I'm just taking screenshots of the graphs for a future presentation, this is amazing!

Maurizio De Giorgi 7:07 PM
same 🙂

Ismael Posada Trobo 7:07 PM

I'm impressed, wow

Troubleshooting GitLab issues

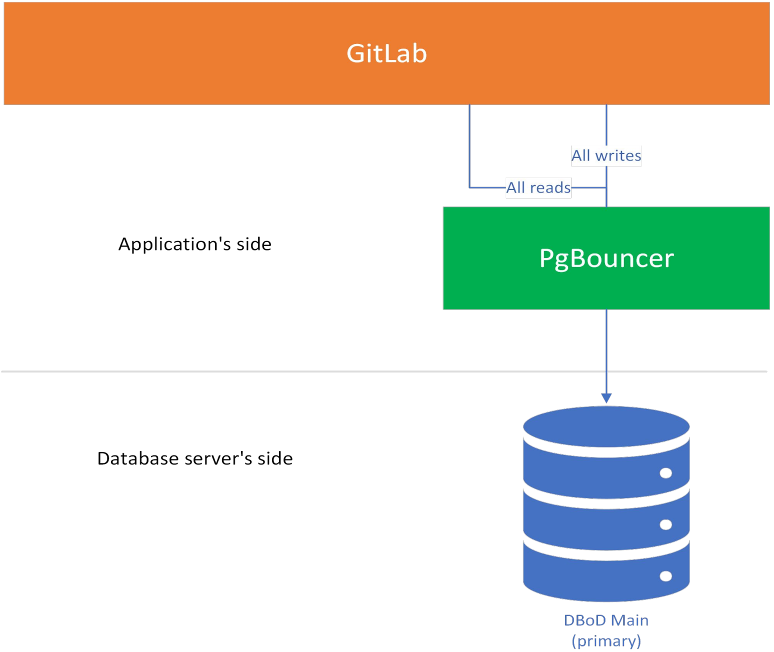
What is PgBouncer?

A lightweight connection pooler for PostgreSQL

- "near" the application and/or "near" the database

PgBouncer modes:

- Session:
  Assigns 1 client connection to a dedicated session, supports all PostgreSQL features, default mode
- **Transaction:**
  Creates a new connection for each transaction, returning the connection to the pool when the transaction is complete, break some features [1]
- Statement:
  Multi-statement transactions disallowed, enforce "autocommit" mode on the client, mostly targeted at PL/Proxy



[1] https://www.pgbouncer.org/features.html

Troubleshooting GitLab issues

Implementation and integration

PgBouncer Helm Chart

- Some existing implementations, but none of them are official nor supported by GitLab.

Created our own

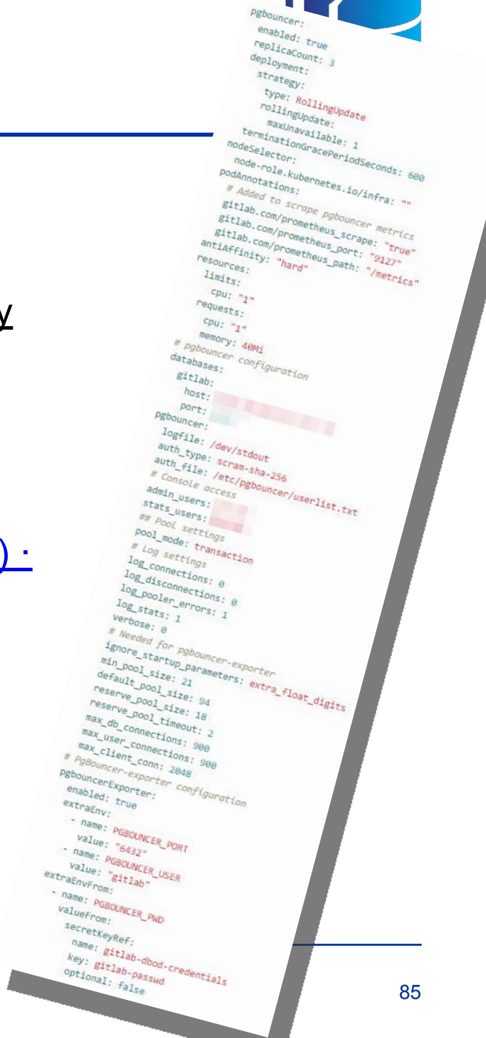
Contribution to GitLab

- Add CERN pgbouncer chart support (&39) · Epics · charts · GitLab
- Document how to integrate GitLab chart and CERN PGBouncer chart (#5527) · Issues · GitLab.org / charts / GitLab Chart · GitLab
- License and maintenance issues

"Click-and-go" for Kubernetes (Incl. monitoring)

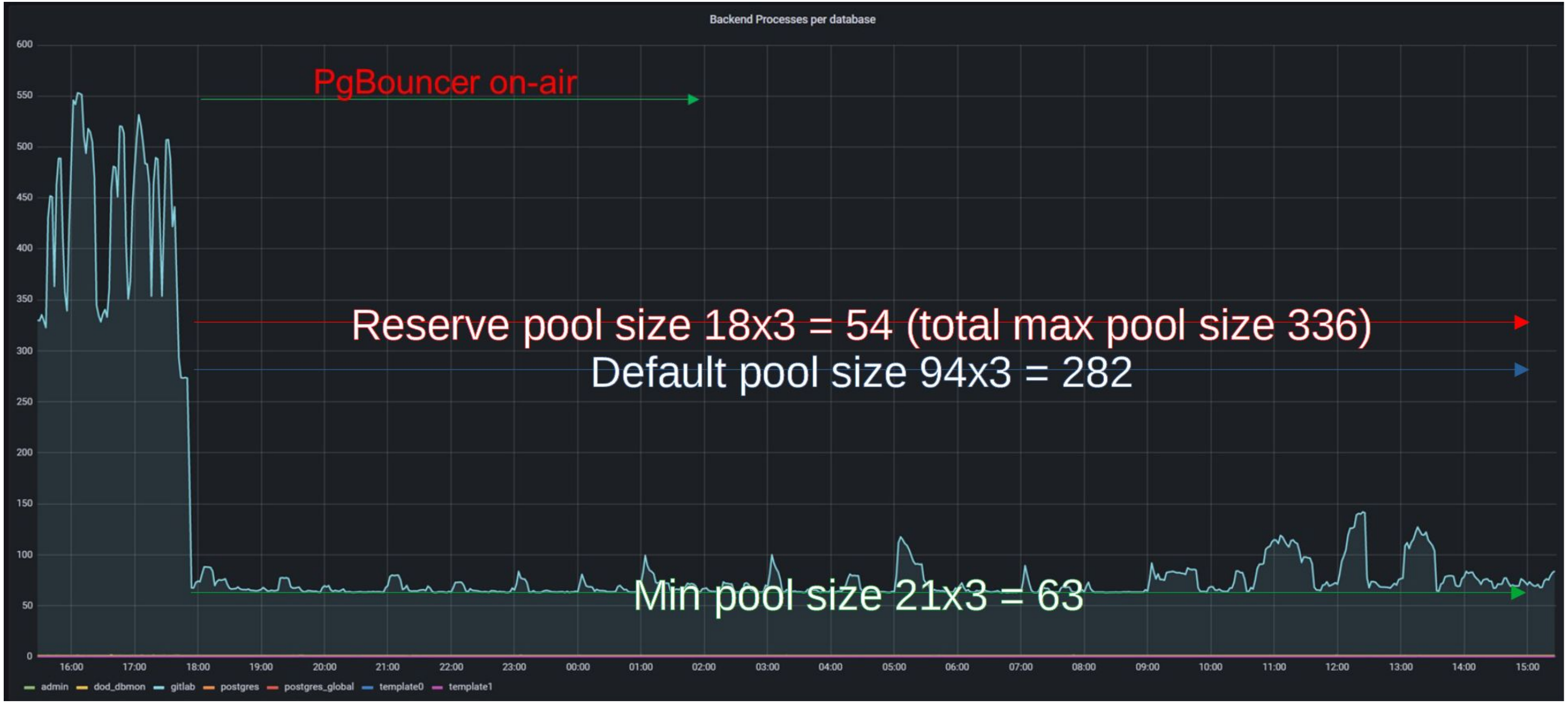
GitLab at CERN integration

- Puma (app server) and Sidekiq (job dispatcher) going through PgBouncer
- Migrations not going through PgBouncer to avoid long-running transactions.
- 3 replicas (one per AVZ)



```yaml
pgbouncer:
  enabled: true
  replicaCount: 3
  deployment:
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxUnavailable: 1
  terminationGracePeriodSeconds: 600
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  podAnnotations:
    # Added to scrape pgbouncer metrics
    gitlab.com/prometheus_scrape: "true"
    gitlab.com/prometheus_port: "9127"
    gitlab.com/prometheus_path: "/metrics"
  antiAffinity: "hard"
  resources:
    limits:
      cpu: "1"
    requests:
      cpu: "1"
      memory: 40Mi
  # pgbouncer configuration
  databases:
    gitlab:
      host: [redacted]
      port:
  pgbouncer:
    logfile: /dev/stdout
    auth_type: scram-sha-256
    auth_file: /etc/pgbouncer/userlist.txt
    # Console access
    admin_users: [redacted]
    stats_users:
    ## Pool settings
    pool_mode: transaction
    # Log settings
    log_connections: 0
    log_disconnections: 0
    log_pooler_errors: 1
    log_stats: 1
    verbose: 0
    # Needed for pgbouncer-exporter
    ignore_startup_parameters: extra_float_digits
    min_pool_size: 21
    default_pool_size: 94
    reserve_pool_size: 18
    reserve_pool_timeout: 2
    max_db_connections: 900
    max_user_connections: 900
    max_client_conn: 2048
  # PgBouncer-exporter configuration
  pgbouncerExporter:
    enabled: true
    extraEnv:
      - name: PGBOUNCER_PORT
        value: "6432"
      - name: PGBOUNCER_USER
        value: "gitlab"
    extraEnvFrom:
      - name: PGBOUNCER_PWD
        valueFrom:
          secretKeyRef:
            name: gitlab-dbod-credentials
            key: gitlab-passwd
            optional: false
```

Troubleshooting GitLab issues



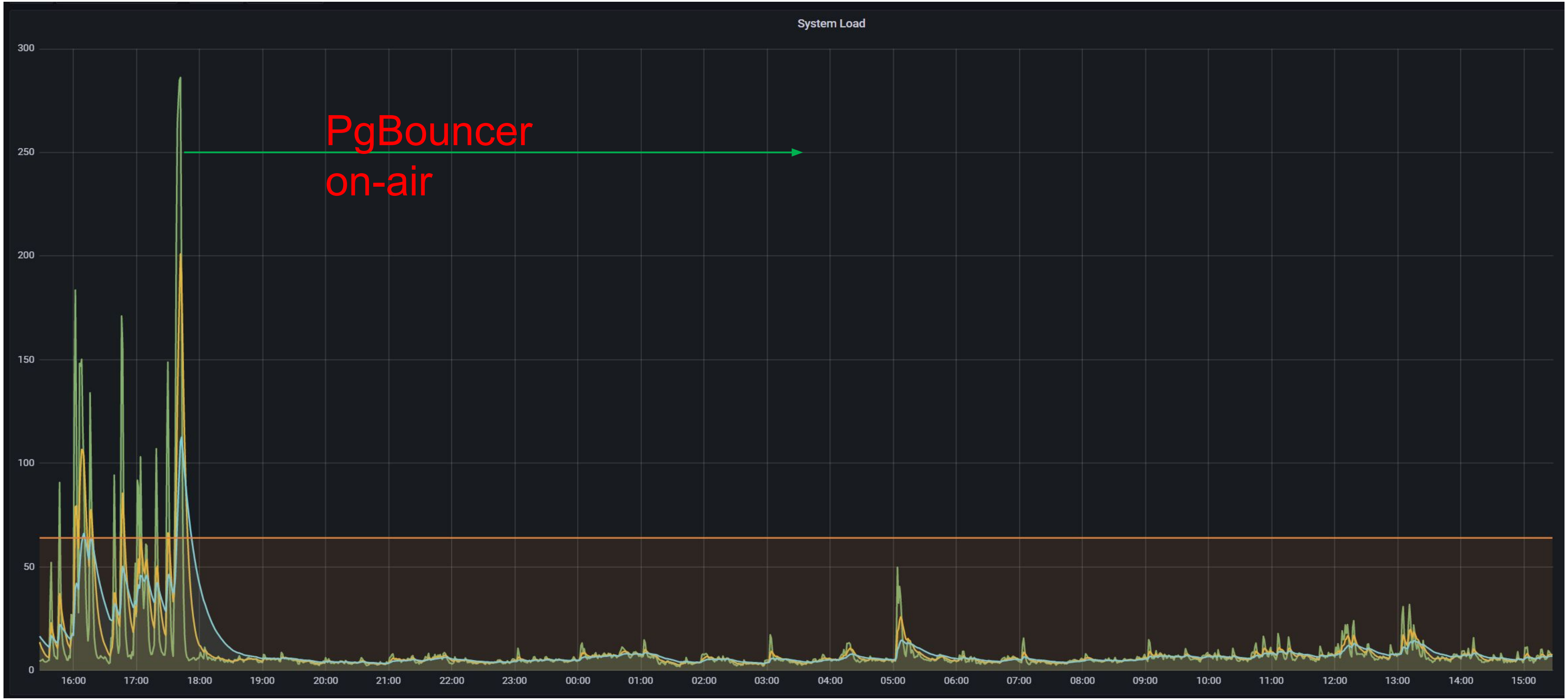
PgBouncer on-air

Reserve pool size 18x3 = 54 (total max pool size 336)

Default pool size 94x3 = 282

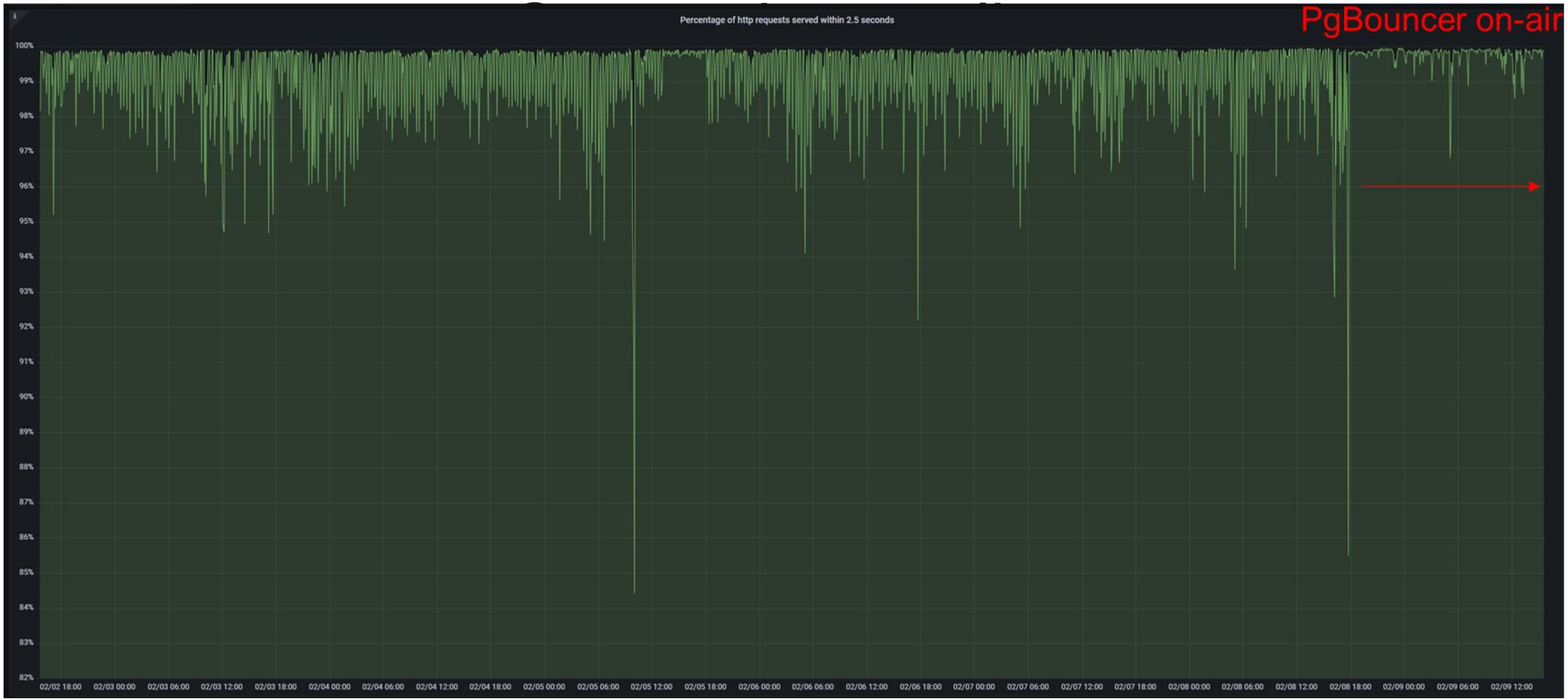
Min pool size 21x3 = 63

Troubleshooting GitLab issues

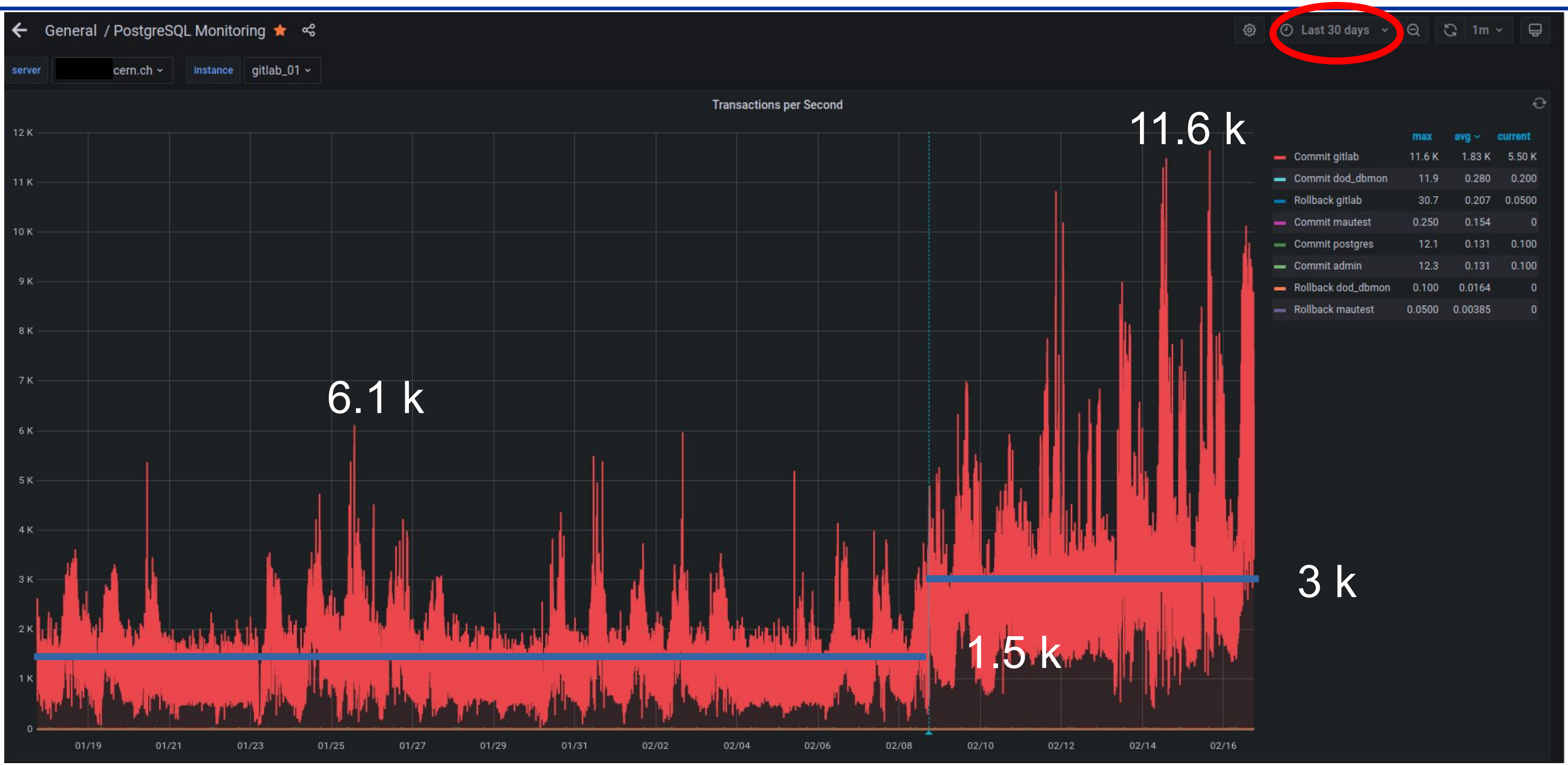


PgBouncer on-air

Troubleshooting GitLab issues



Troubleshooting GitLab issues



General / PostgreSQL Monitoring

server [redacted]cern.ch

instance gitlab_01

Last 30 days

1m

Transactions per Second

	max	avg	current
🟥 Commit gitlab	11.6 K	1.83 K	5.50 K
Commit dod_dbmon	11.9	0.280	0.200
Rollback gitlab	30.7	0.207	0.0500
Commit mautest	0.250	0.154	0
Commit postgres	12.1	0.131	0.100
Commit admin	12.3	0.131	0.100
Rollback dod_dbmon	0.100	0.0164	0
Rollback mautest	0.0500	0.00385	0

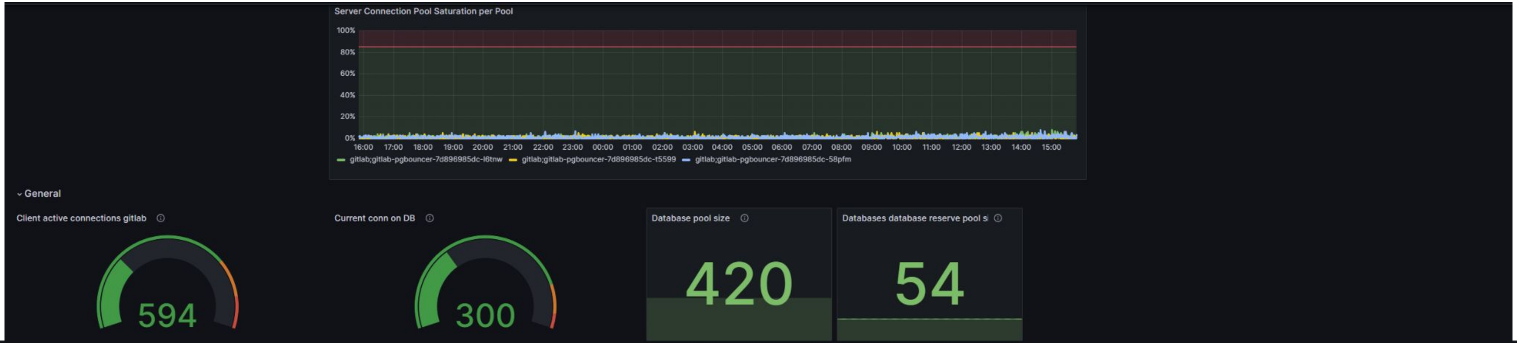
11.6 k

6.1 k

3 k

1.5 k

Troubleshooting GitLab issues



Server Connection Pool Saturation per Pool

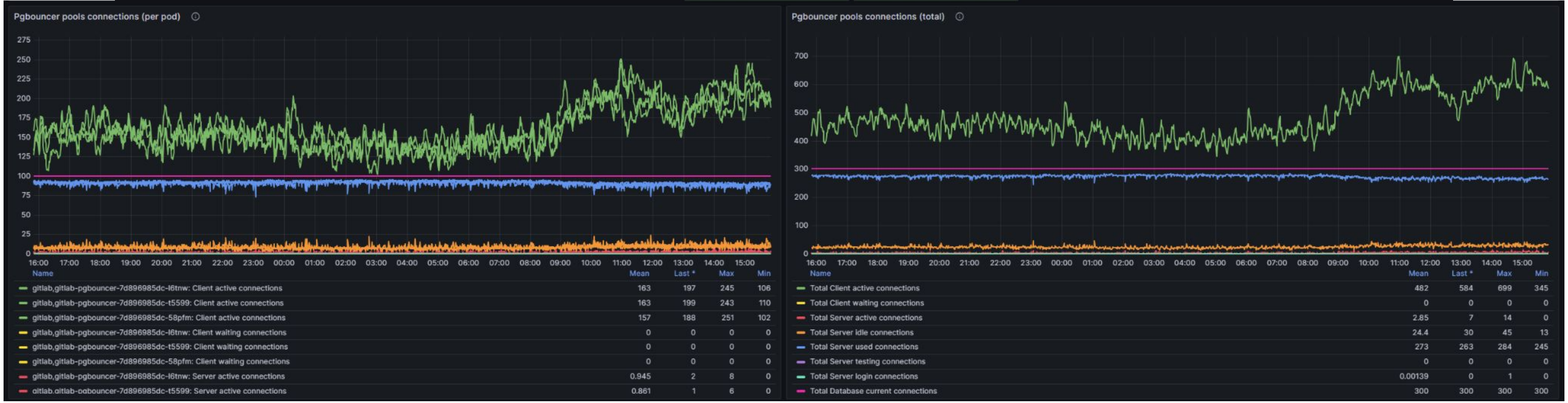
⌄ General

Client active connections gitlab ⓘ 594

Current conn on DB ⓘ 300

Database pool size ⓘ 420

Databases database reserve pool si ⓘ 54



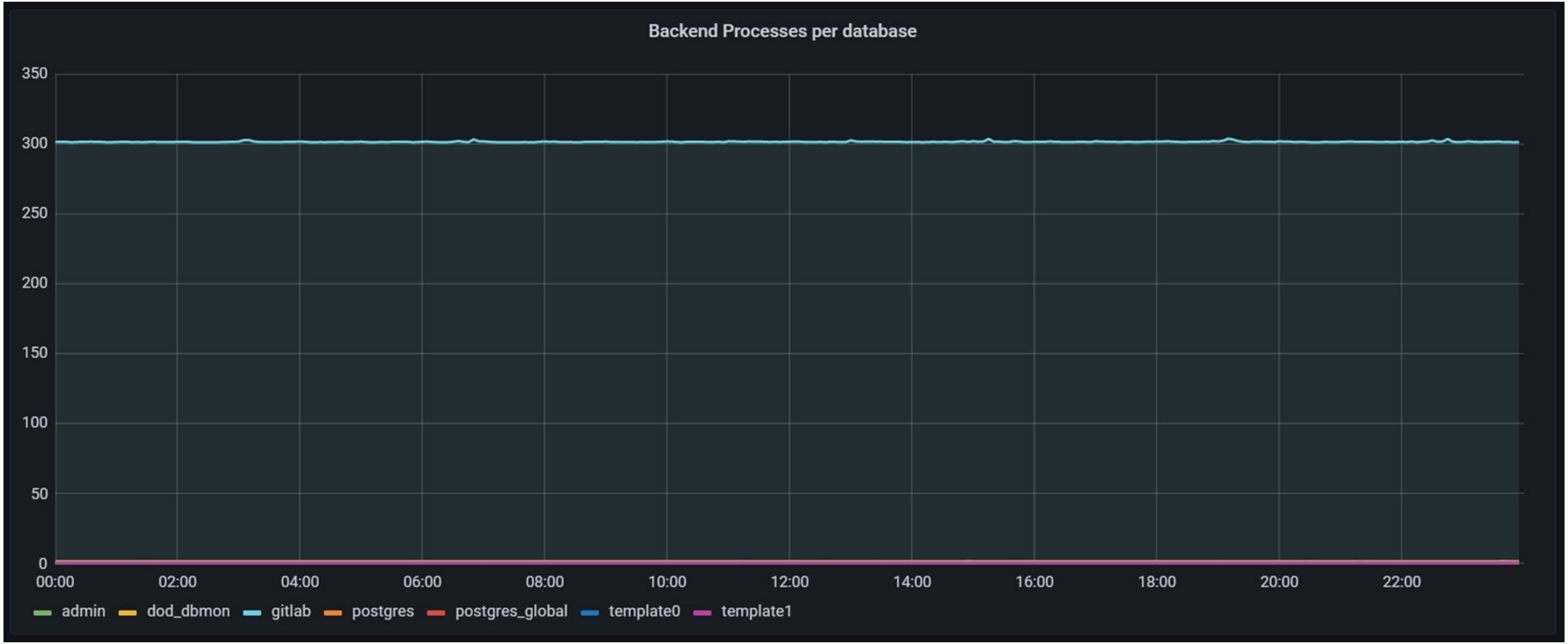
Pgbouncer pools connections (per pod) ⓘ

Name	Mean	Last *	Max	Min
🟩 gitlab,gitlab-pgbouncer-7d896985dc-l6tnw: Client active connections	163	197	245	106
🟩 gitlab,gitlab-pgbouncer-7d896985dc-t5599: Client active connections	163	199	243	110
🟩 gitlab,gitlab-pgbouncer-7d896985dc-58pfm: Client active connections	157	188	251	102
🟨 gitlab,gitlab-pgbouncer-7d896985dc-l6tnw: Client waiting connections	0	0	0	0
🟨 gitlab,gitlab-pgbouncer-7d896985dc-t5599: Client waiting connections	0	0	0	0
🟨 gitlab,gitlab-pgbouncer-7d896985dc-58pfm: Client waiting connections	0	0	0	0
🟥 gitlab,gitlab-pgbouncer-7d896985dc-l6tnw: Server active connections	0.945	2	8	0
🟥 gitlab,gitlab-pgbouncer-7d896985dc-t5599: Server active connections	0.861	1	6	0

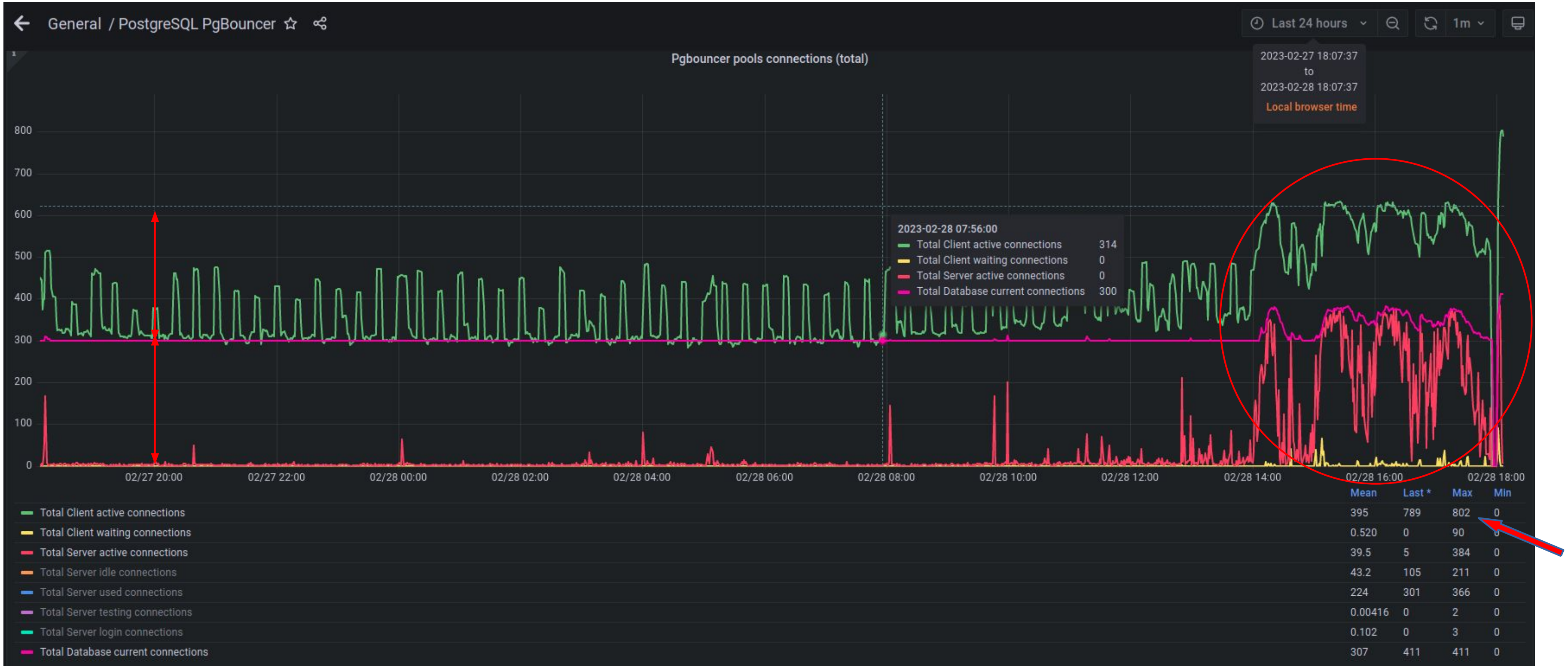
Pgbouncer pools connections (total) ⓘ

Name	Mean	Last *	Max	Min
🟩 Total Client active connections	482	584	699	345
🟨 Total Client waiting connections	0	0	0	0
🟥 Total Server active connections	2.85	7	14	0
🟧 Total Server idle connections	24.4	30	45	13
🟦 Total Server used connections	273	263	284	245
🟪 Total Server testing connections	0	0	0	0
🟩 Total Server login connections	0.00139	0	1	0
🟪 Total Database current connections	300	300	300	300

Troubleshooting GitLab issues



Troubleshooting GitLab issues



2023-02-27 18:07:37
to
2023-02-28 18:07:37
Local browser time

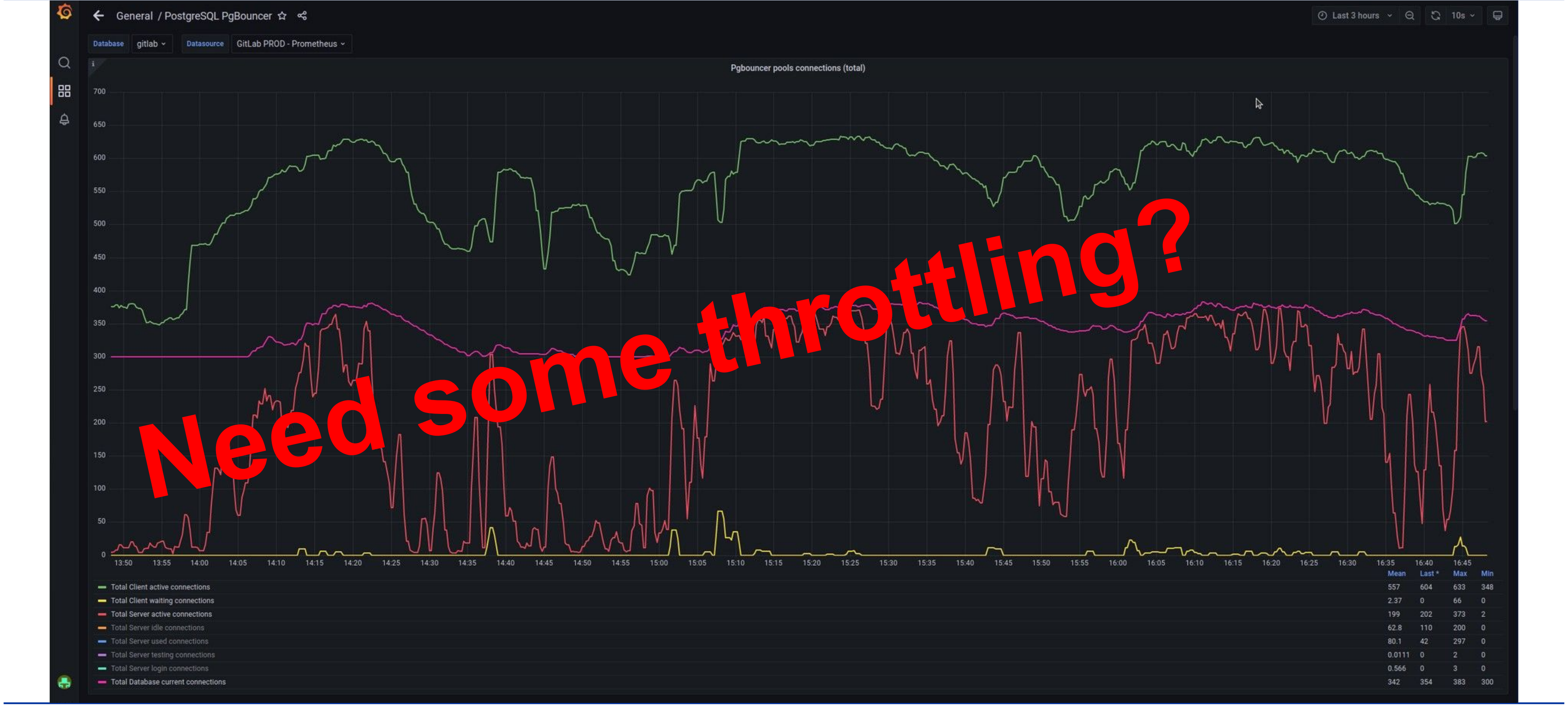
Pgbouncer pools connections (total)

2023-02-28 07:56:00

🟩 Total Client active connections	314
🟨 Total Client waiting connections	0
🟥 Total Server active connections	0
🟪 Total Database current connections	300

	Mean	Last *	Max	Min
🟩 Total Client active connections	395	789	802	0
🟨 Total Client waiting connections	0.520	0	90	0
🟥 Total Server active connections	39.5	5	384	0
🟧 Total Server idle connections	43.2	105	211	0
🟦 Total Server used connections	224	301	366	0
🟪 Total Server testing connections	0.00416	0	2	0
🟩 Total Server login connections	0.102	0	3	0
🟪 Total Database current connections	307	411	411	0

Troubleshooting GitLab issues



	Mean	Last *	Max	Min
Total Client active connections	557	604	633	348
Total Client waiting connections	2.37	0	66	0
Total Server active connections	199	202	373	2
Total Server idle connections	62.8	110	200	0
Total Server used connections	80.1	42	297	0
Total Server testing connections	0.0111	0	2	0
Total Server login connections	0.566	0	3	0
Total Database current connections	342	354	383	300

Need some throttling?

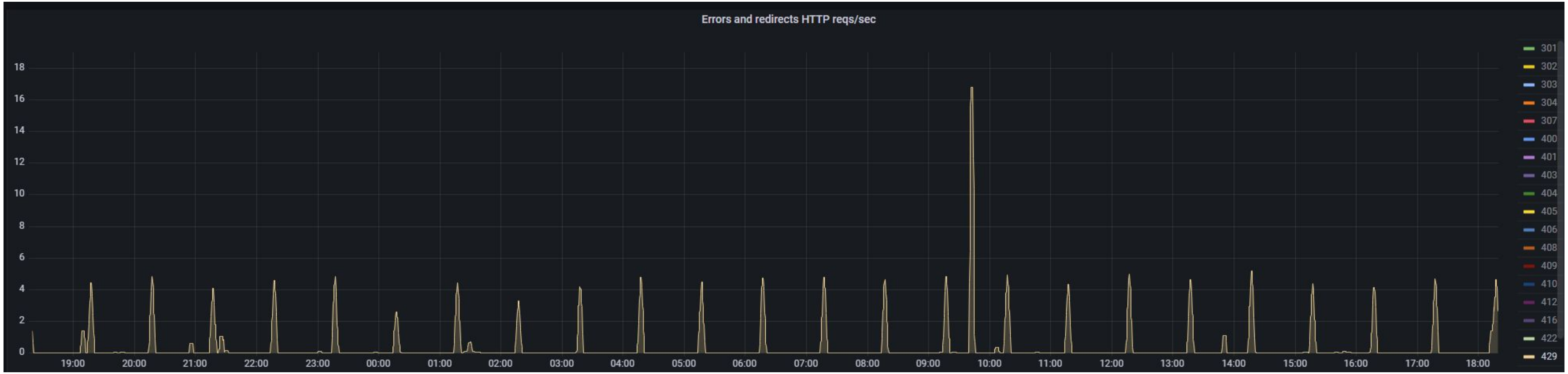
Troubleshooting GitLab issues

Throttling and Rate limits

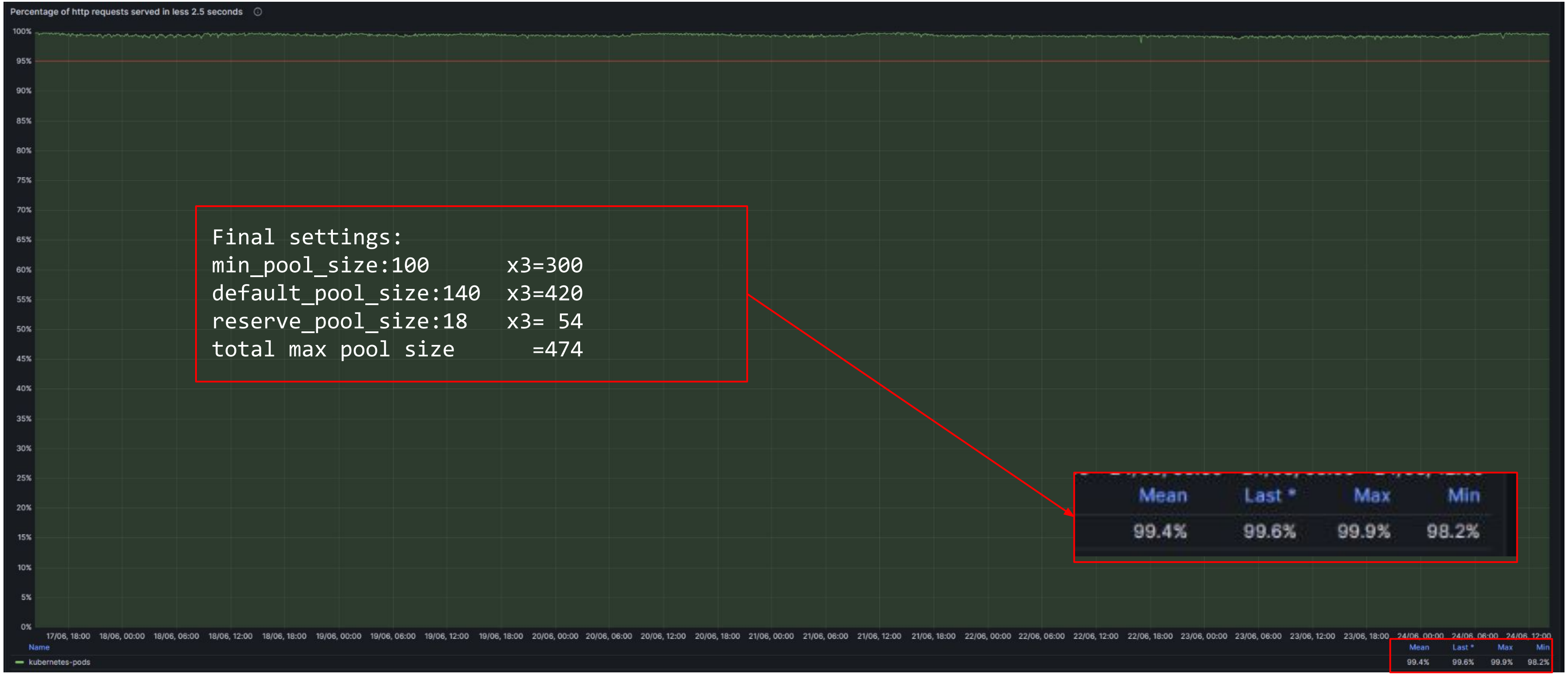
Misuse and/or abuse from some users: Too many request – Error 429

- Infinite loops hammering the API: [Set rate limit for reqs/sec]
- Huge number of jobs triggered simultaneously: Rate limit for the maximum number of jobs triggered per project

[Use response headers to make your scripts smarter]



Troubleshooting GitLab issues



```
Final settings:
min_pool_size:100      x3=300
default_pool_size:140  x3=420
reserve_pool_size:18   x3= 54
total max pool size      =474
```

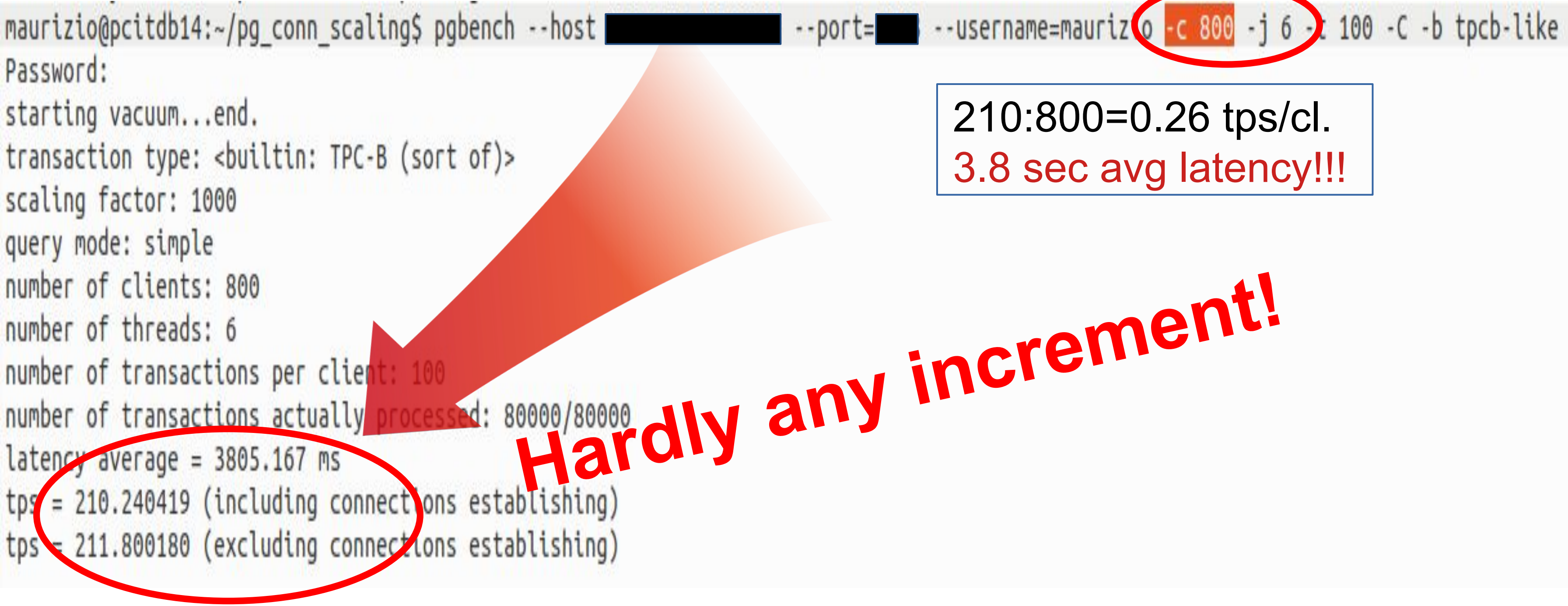
Mean	Last *	Max	Min
99.4%	99.6%	99.9%	98.2%

Agenda

- **An MVCC primer (boring things everyone knows but it is worth refreshing)**
    - (ACID) Transactions, Isolation, Concurrency, Serializable Snapshot Isolation
- **Connection scalability (showing the problem and its causes)**
    - Benchmarking & bottleneck analysis
- **Troubleshooting GitLab issues (talking about that time when we all had a lot of fun)**
    - The journey to enlightenment
    - The joy of enlightenment
- **The great effects of connection pooling on connection scalability**

Connections scalability

A **tpcb-like** run with 800 pgbench clients, 6 threads, 100 trx/client

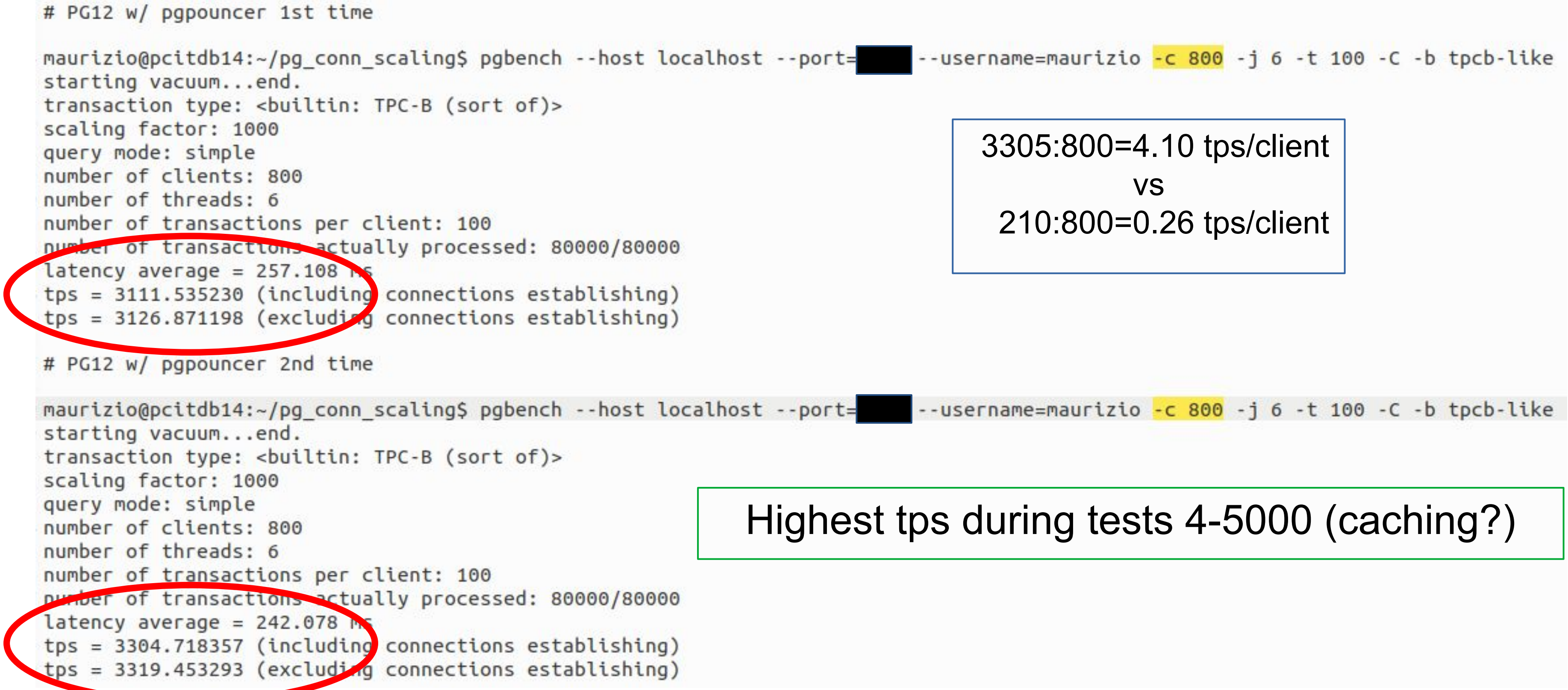


```
maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host [redacted] --port=[redacted] --username=maurizio -c 800 -j 6 -t 100 -C -b tpcb-like
Password:
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 800
number of threads: 6
number of transactions per client: 100
number of transactions actually processed: 80000/80000
latency average = 3805.167 ms
tps = 210.240419 (including connections establishing)
tps = 211.800180 (excluding connections establishing)
```

210:800=0.26 tps/cl.
3.8 sec avg latency!!!

Hardly any increment!

Effects of connection pooling



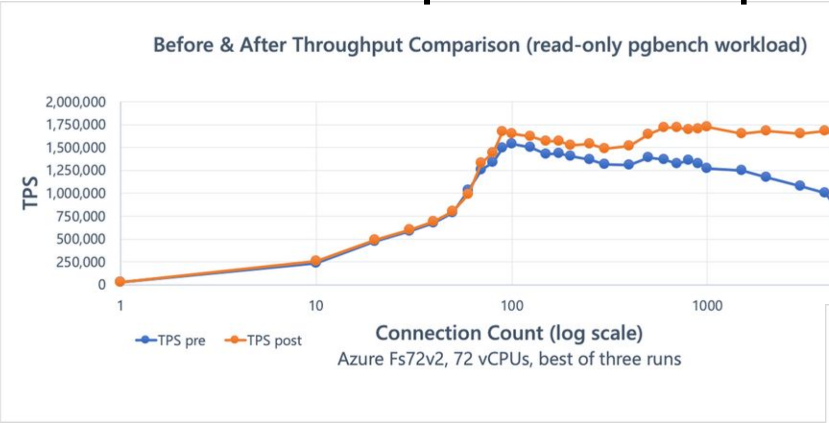
```
# PG12 w/ pgpouncer 1st time

maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host localhost --port=[redacted] --username=maurizio -c 800 -j 6 -t 100 -C -b tpcb-like
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 800
number of threads: 6
number of transactions per client: 100
number of transactions actually processed: 80000/80000
latency average = 257.108 ms
tps = 3111.535230 (including connections establishing)
tps = 3126.871198 (excluding connections establishing)

# PG12 w/ pgpouncer 2nd time

maurizio@pcitdb14:~/pg_conn_scaling$ pgbench --host localhost --port=[redacted] --username=maurizio -c 800 -j 6 -t 100 -C -b tpcb-like
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 800
number of threads: 6
number of transactions per client: 100
number of transactions actually processed: 80000/80000
latency average = 242.078 ms
tps = 3304.718357 (including connections establishing)
tps = 3319.453293 (excluding connections establishing)
```

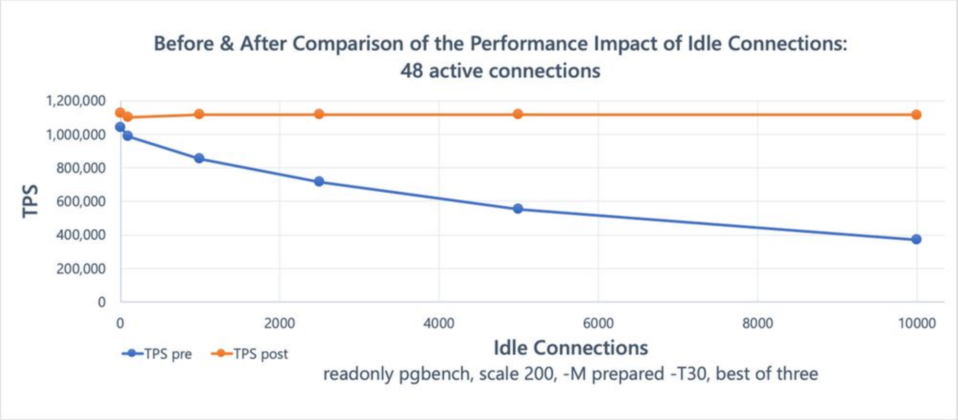
3305:800=4.10 tps/client
vs
210:800=0.26 tps/client

Highest tps during tests 4-5000 (caching?)

Removing the bottleneck

Improved snapshot scalability in PG14





… but connection pooling is still needed

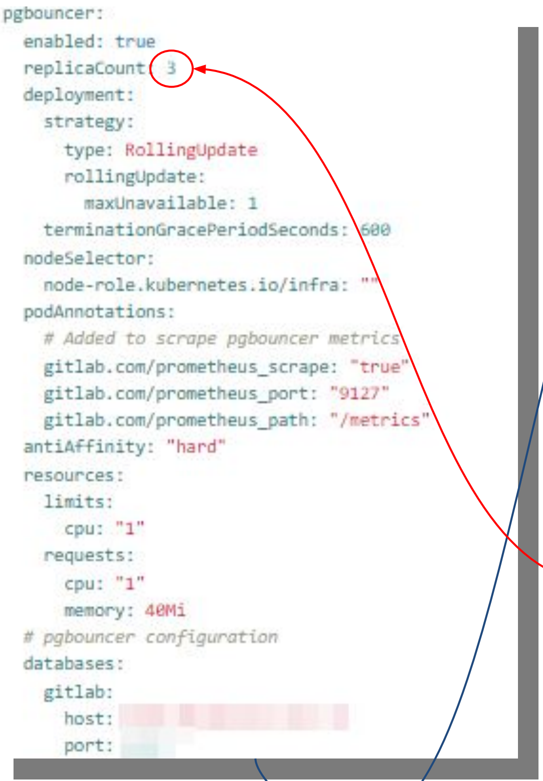
Credits http://cern.ch/go/9WRh Andres Freund@MS

Client side connection pooling: fast

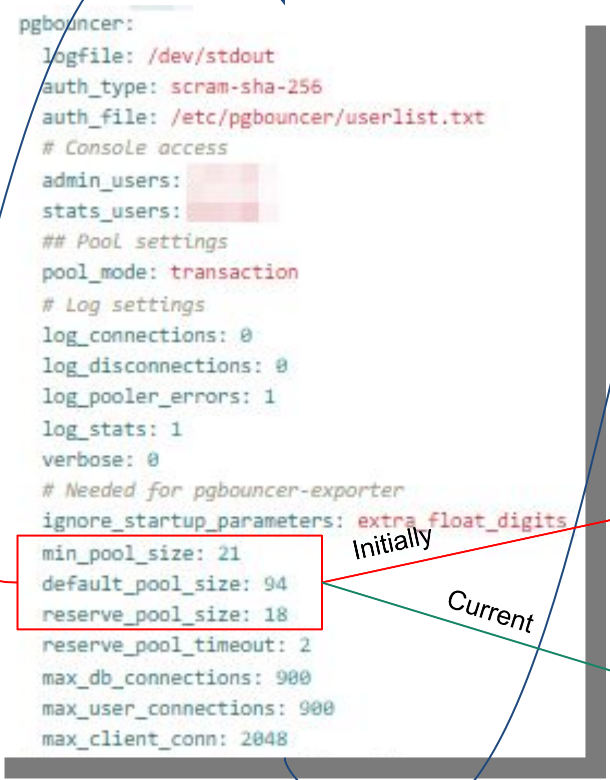
PgBouncer is a well known, flexible, reputable connection pooling software for PostgreSQL with a small footprint, which has been around for a long time

*Application **owners can setup PgBouncer on their side ("near" the application)** to establish a connection pooling layer when accessing the database with a significant number of connections or when the connections are often and suddenly going up and down by a significant number*

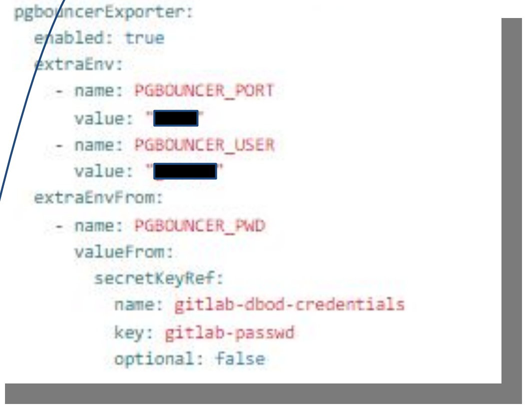
Client side connection pooling: config



```
pgbouncer:
  enabled: true
  replicaCount: 3
  deployment:
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxUnavailable: 1
    terminationGracePeriodSeconds: 600
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  podAnnotations:
    # Added to scrape pgbouncer metrics
    gitlab.com/prometheus_scrape: "true"
    gitlab.com/prometheus_port: "9127"
    gitlab.com/prometheus_path: "/metrics"
  antiAffinity: "hard"
  resources:
    limits:
      cpu: "1"
    requests:
      cpu: "1"
      memory: 40Mi
  # pgbouncer configuration
  databases:
    gitlab:
      host: [redacted]
      port: [redacted]
```



```
pgbouncer:
  logfile: /dev/stdout
  auth_type: scram-sha-256
  auth_file: /etc/pgbouncer/userlist.txt
  # Console access
  admin_users: [redacted]
  stats_users: [redacted]
  ## Pool settings
  pool_mode: transaction
  # Log settings
  log_connections: 0
  log_disconnections: 0
  log_pooler_errors: 1
  log_stats: 1
  verbose: 0
  # Needed for pgbouncer-exporter
  ignore_startup_parameters: extra_float_digits
  min_pool_size: 21
  default_pool_size: 94
  reserve_pool_size: 18
  reserve_pool_timeout: 2
  max_db_connections: 900
  max_user_connections: 900
  max_client_conn: 2048
```



```
pgbouncerExporter:
  enabled: true
  extraEnv:
    - name: PGBOUNCER_PORT
      value: "[redacted]"
    - name: PGBOUNCER_USER
      value: "[redacted]"
  extraEnvFrom:
    - name: PGBOUNCER_PWD
      valueFrom:
        secretKeyRef:
          name: gitlab-dbod-credentials
          key: gitlab-passwd
          optional: false
```

Initially

```
min_pool_size:21        x3= 63
default_pool_size:94    x3=282
reserve_pool_size:18    x3= 54
total max pool size       =336
```

Current

```
min_pool_size:100       x3=300
default_pool_size:140   x3=420
reserve_pool_size:18    x3= 54
total max pool size       =474
```

Client side connection pooling: auth

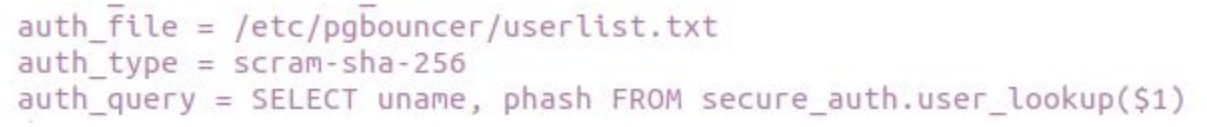
PgBouncer secure authentication in DBOD

There are different ways to authenticate users in PgBouncer including:

- authentication query returning the password hash
- authentication file with known roles and their password (clear text/hash)

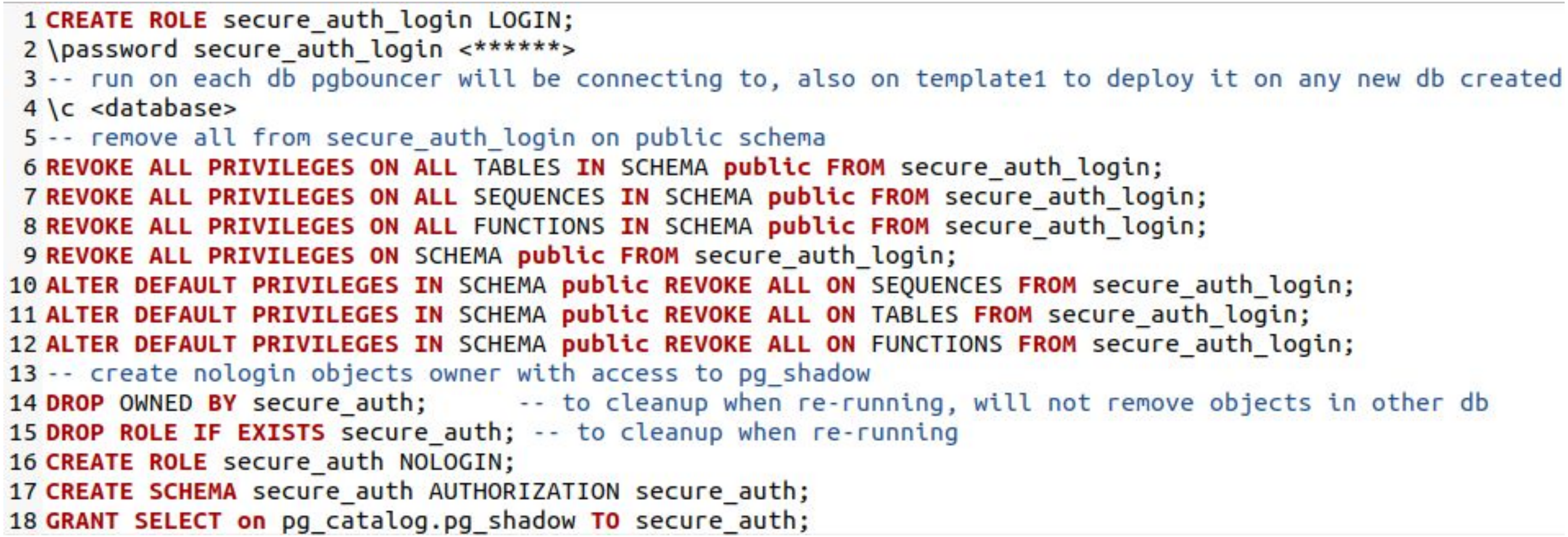
Superuser access to pg_shadow table would be required to get the hash

Secure auth setup: restricted login role + user_lookup function returning the password hash (filtering privileged and special users)



```
auth_file = /etc/pgbouncer/userlist.txt
auth_type = scram-sha-256
auth_query = SELECT uname, phash FROM secure_auth.user_lookup($1)
```

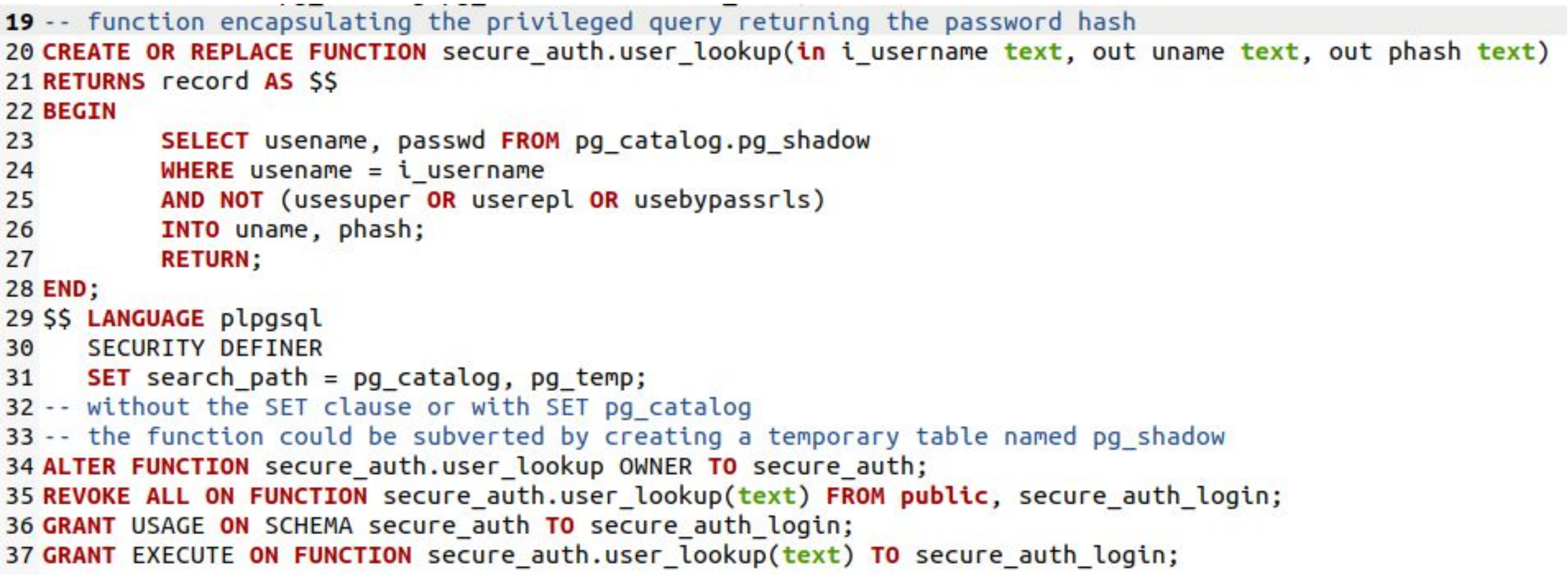
Client side connection pooling: auth



```sql
CREATE ROLE secure_auth_login LOGIN;
\password secure_auth_login <******>
-- run on each db pgbouncer will be connecting to, also on template1 to deploy it on any new db created
\c <database>
-- remove all from secure_auth_login on public schema
REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA public FROM secure_auth_login;
REVOKE ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public FROM secure_auth_login;
REVOKE ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public FROM secure_auth_login;
REVOKE ALL PRIVILEGES ON SCHEMA public FROM secure_auth_login;
ALTER DEFAULT PRIVILEGES IN SCHEMA public REVOKE ALL ON SEQUENCES FROM secure_auth_login;
ALTER DEFAULT PRIVILEGES IN SCHEMA public REVOKE ALL ON TABLES FROM secure_auth_login;
ALTER DEFAULT PRIVILEGES IN SCHEMA public REVOKE ALL ON FUNCTIONS FROM secure_auth_login;
-- create nologin objects owner with access to pg_shadow
DROP OWNED BY secure_auth;      -- to cleanup when re-running, will not remove objects in other db
DROP ROLE IF EXISTS secure_auth; -- to cleanup when re-running
CREATE ROLE secure_auth NOLOGIN;
CREATE SCHEMA secure_auth AUTHORIZATION secure_auth;
GRANT SELECT on pg_catalog.pg_shadow TO secure_auth;
```

```
# pg_hba.conf
hostssl    all   secure_auth_login    all   scram-sha-256
```

Client side connection pooling: auth



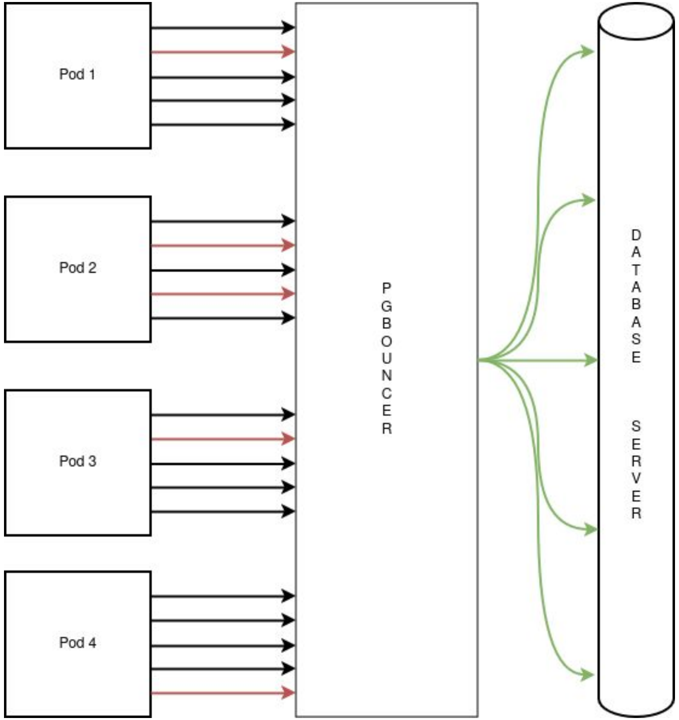
```
-- function encapsulating the privileged query returning the password hash
CREATE OR REPLACE FUNCTION secure_auth.user_lookup(in i_username text, out uname text, out phash text)
RETURNS record AS $$
BEGIN
        SELECT usename, passwd FROM pg_catalog.pg_shadow
        WHERE usename = i_username
        AND NOT (usesuper OR userepl OR usebypassrls)
        INTO uname, phash;
        RETURN;
END;
$$ LANGUAGE plpgsql
   SECURITY DEFINER
   SET search_path = pg_catalog, pg_temp;
-- without the SET clause or with SET pg_catalog
-- the function could be subverted by creating a temporary table named pg_shadow
ALTER FUNCTION secure_auth.user_lookup OWNER TO secure_auth;
REVOKE ALL ON FUNCTION secure_auth.user_lookup(text) FROM public, secure_auth_login;
GRANT USAGE ON SCHEMA secure_auth TO secure_auth_login;
GRANT EXECUTE ON FUNCTION secure_auth.user_lookup(text) TO secure_auth_login;
```

Take home: connections scalability

PostgreSQL connections scalability has been improved in recent versions but, in some cases, to achieve satisfactory results a connection pooling software is required and strongly recommended. An helm chart provided by the community for the community, would significantly facilitate the deployment and the adoption of PgBouncer, particularly for applications deployed with K8s.

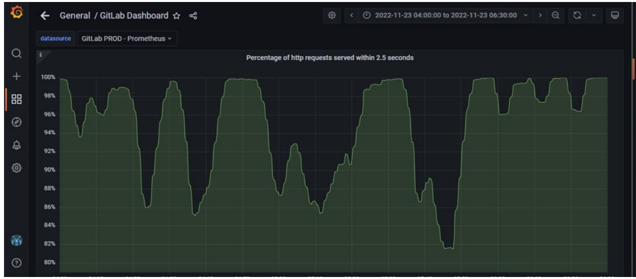
Take home: connection pooling and K8s

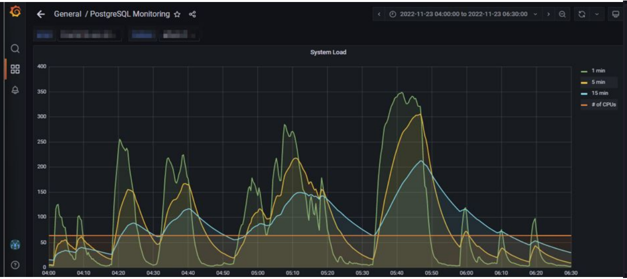
Depending on the usage pattern, this seems to be of particular importance for applications with an OLTP/OLAP load – and especially if they are deployed over multiple nodes, containers, pods - and they use more than a couple of hundreds connections, mostly idle, while opening/closing others.

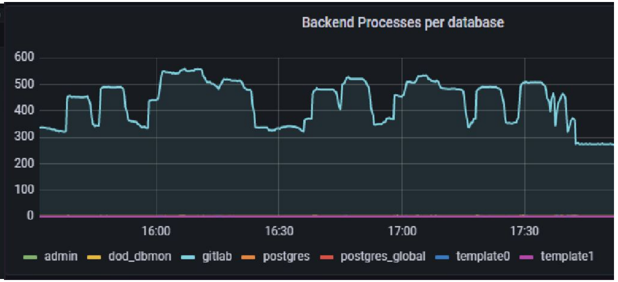


From zero… to hero

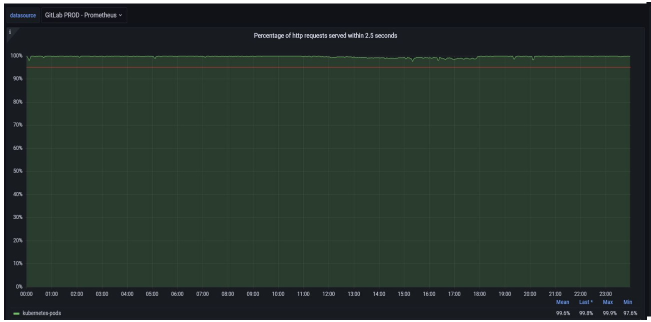
Long journey from…

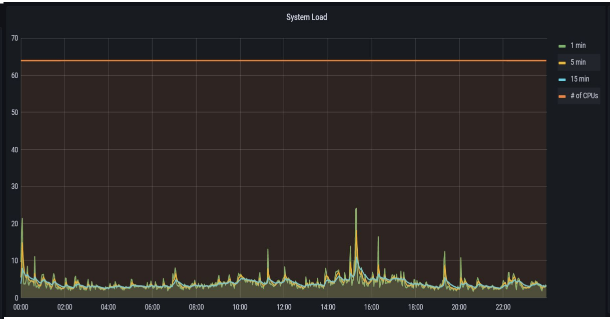


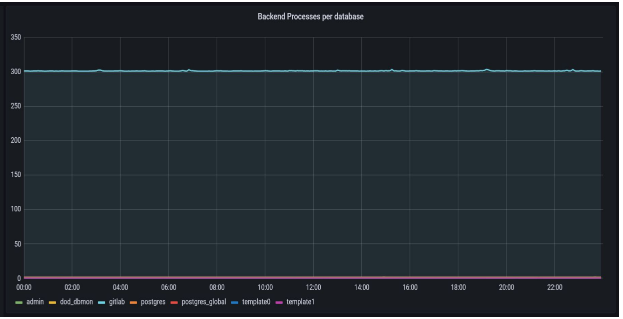




To…







DB on Demand is hiring!

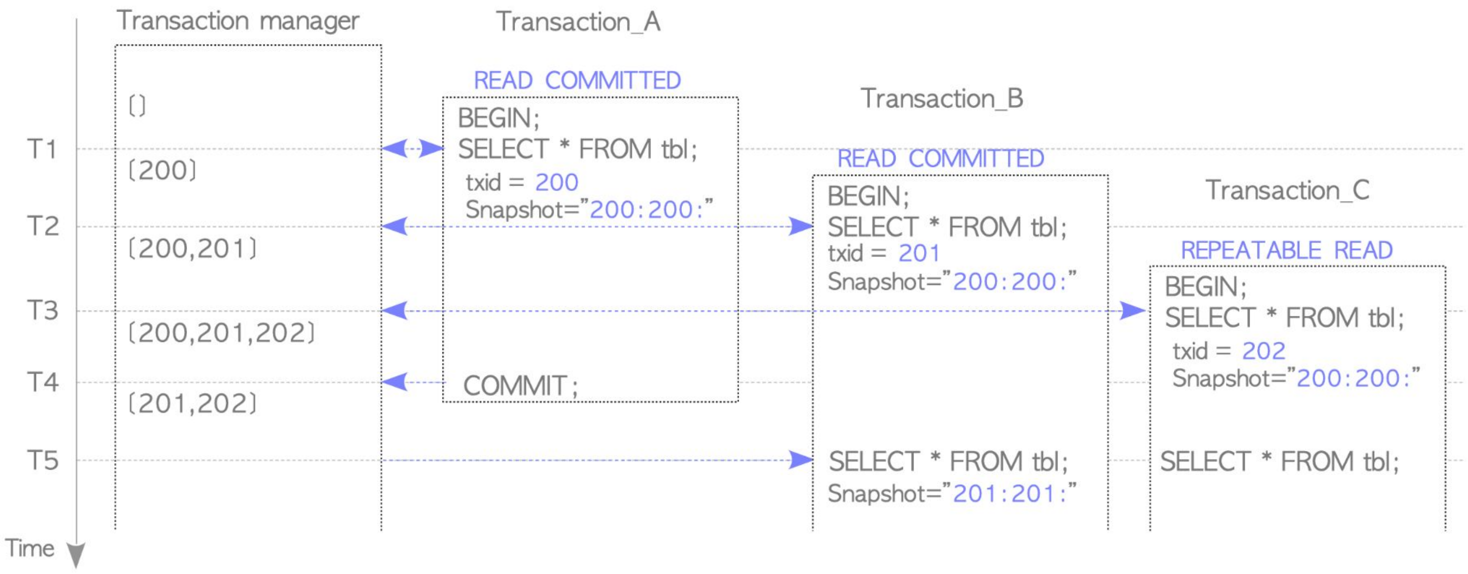
- ➡ early-career technician
- ➡ member (or associated) states individuals
- ➡ max two years of professional experience
- ➡ highest educational qualification by the application deadline: secondary education diploma
- ➡ info and application https://cern.ch/it-da-db-2024-105-grae

That’s all folks!

Maurizio De Giorgi
maurizio.degiorgi@cern.ch

Ismael Posada Trobo
ismael.posada.trobo@cern.ch

Multi Version Concurrency Control



READ COMMITTED: a snapshot for each statement

REPEATABLE READ/SERIALIZABLE: a snapshot for 1st statement

Credits Hironobu SUZUKI@InterDB https://www.interdb.jp/pg/pgsql05.html