Closed (moved) Opened 3 years ago by **Pablo Carranza [GitLab]** 6 of 9 tasks completed

Database Outage on 2016/11/28 when project_authorizations had too much bloat

TL;DR

On Monday 2016/11/28 we had an outage that took GitLab.com down. The initial culprit was high database load, further investigation pointed in the direction of a large amount of slow queries which eventually pointed in the direction of a large amount of table bloat in the project_authorizations table.

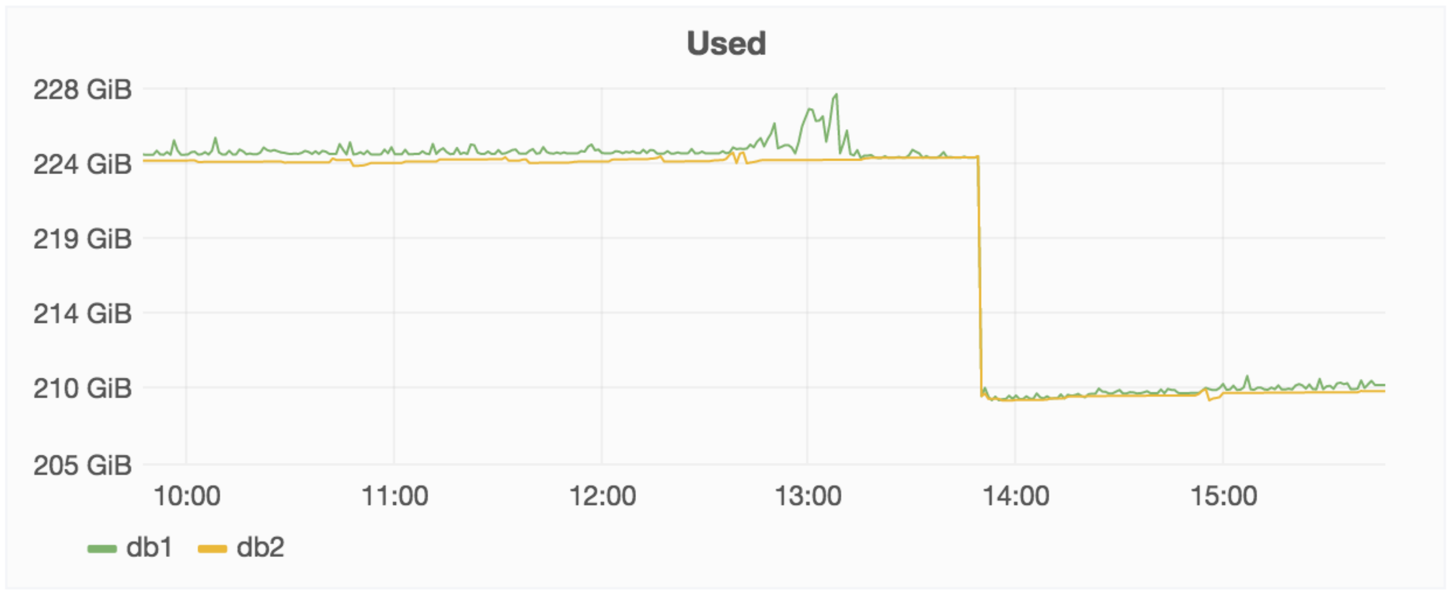
The outage was resolved by issuing a `VACUUM FULL project_authorizations` command in the database, which instantly removed all the bloat away and returned the DB to normal operational levels.

Timeline

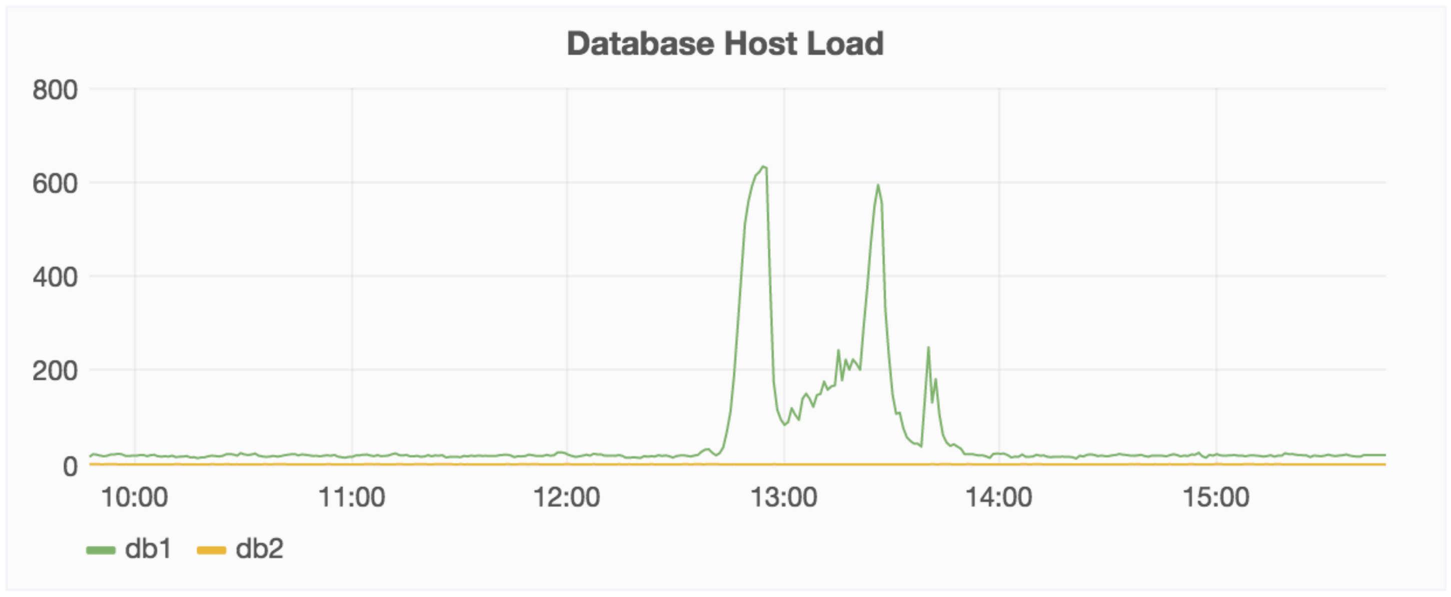
- 12:50 UTC - we got alerted by @jnijhof in slack that the DB had a load of 400+
- 13:03 UTC - we ssh'd into the db1 server to try to rule out what the problem was. At this point we executed a set of queries to understand if there were slow and/or blocked queries. There were a number of those all around.
- At this point we reviewed if Vacuum was executing to find that `VACUUM ANALYZE public.ci_runners`, `ANALYZE public.projects`, `ANALYZE public.users` and `VACUUM ANALYZE public.user_activities` where executing, some of them for a while already. This turned out to be a red herring.
- We also saw that there were a bunch of different queries executing for a long time in the database already, circa 500 to 600 slow queries were already in the database. So we decided to kill these slow queries to get the database in a better shape and unlock Vacuum so it could finish.
- After killing queries the main offender in slow queries turned to be `SELECT  "projects".* FROM "projects" INNER JOIN "project_authorizations" ON "projects"."id" = "project_authorizations"...`
- Still load was not going down so we kept killing queries with no form of success. Our reasoning was that vacuum was blocking the database as we have seen before, but even we were killing queries all the time the situation was not improving.
- We left a `while true; ./kill_blocking_queries; sleep 1; done` executing in a terminal to keep slow and blocking queries down while vacuum was doing it's job.
- At some point all the vacuum process finished, we cancelled the killing loop and load started going high again.
- 13:40 UTC - Here we took a step back trying to understand what was actually going on, as Vacuum was not to be blamed anymore. We were still piling slow queries and load was not going down.
  - using `htop` we realized that memory usage was actually low (32G out of 100G assigned to shared buffers)
  - We double checked this in the database itself by issuing the command `show ALL;` to then look up the value `shared_buffers                      | 112896MB`
  - All the cores where at 100% usage in user space.
- We checked the graphs for postgresql and followed the lead of the table `project_authorizations` having a spike up to 8M dead tuples at 12:40 UTC, even though the graph was showing a low number now.
- We started taking a wider look at the graphs and realized that the database was using 220Gb of storage space, which was particularly high for what we were used to (180G)
- 13:49 UTC - After some discussion we decided to issue a `VACUUM FULL` on project_authorizations as it was the main offender and was not being vacuumed for some reason.
- 13:50 UTC
  - storage usage in the database drops 20G,
  - load normalizes at 20 and connections start to recover.
  - Slow queries count goes away only showing vacuum process as being slow (normal behavior)
- We remove killing scripts and wait a minute monitoring the database to check that it's responding correctly.
- Everything is calmed, the outage is resolved.

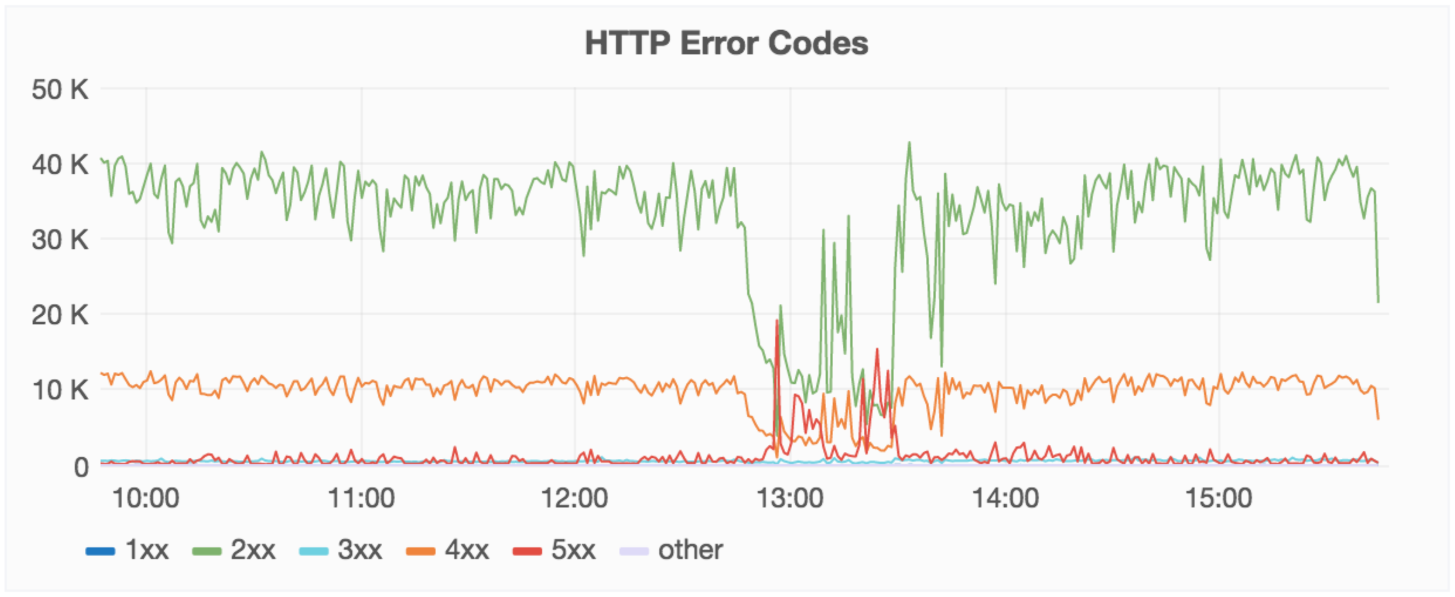
Graphs

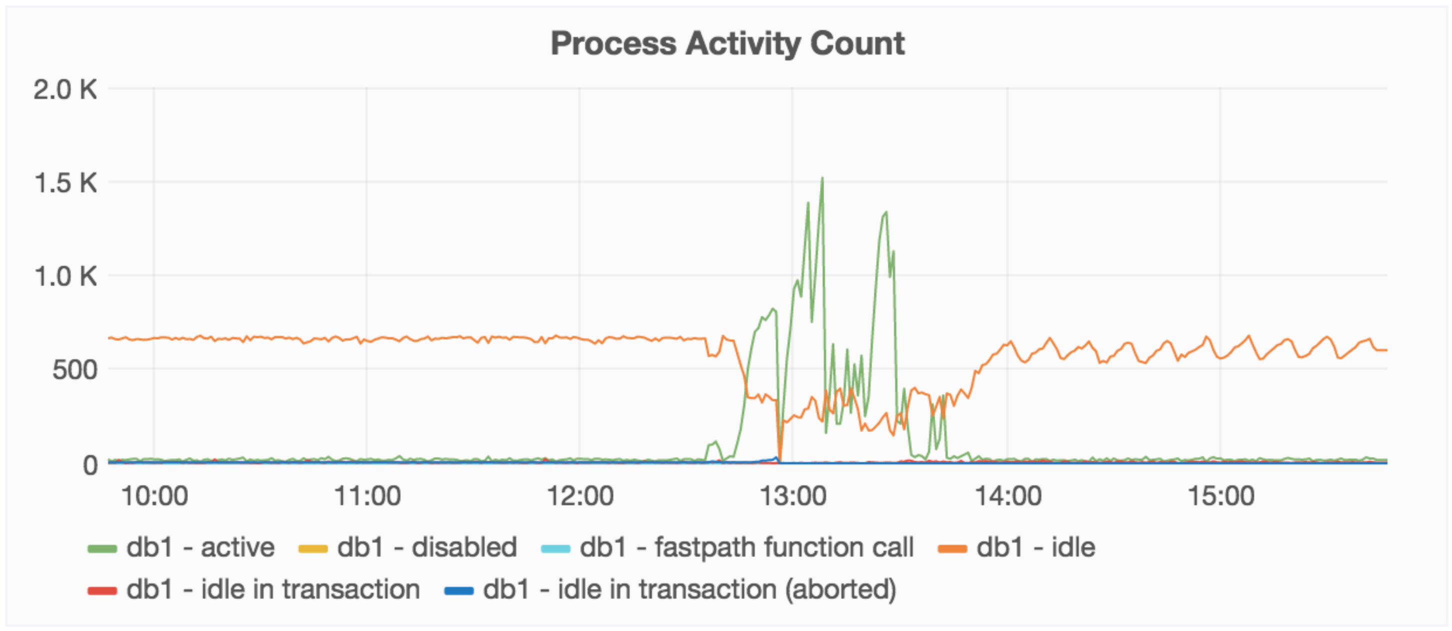
How storage dropped when the outage was resolved

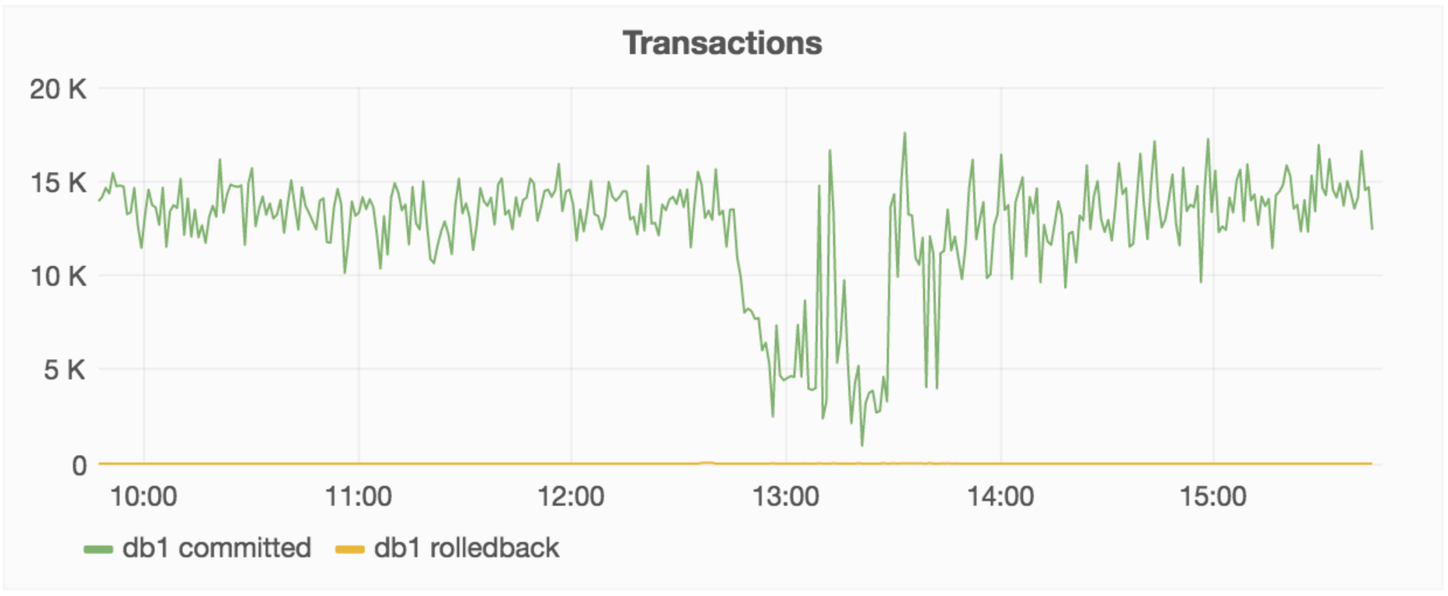


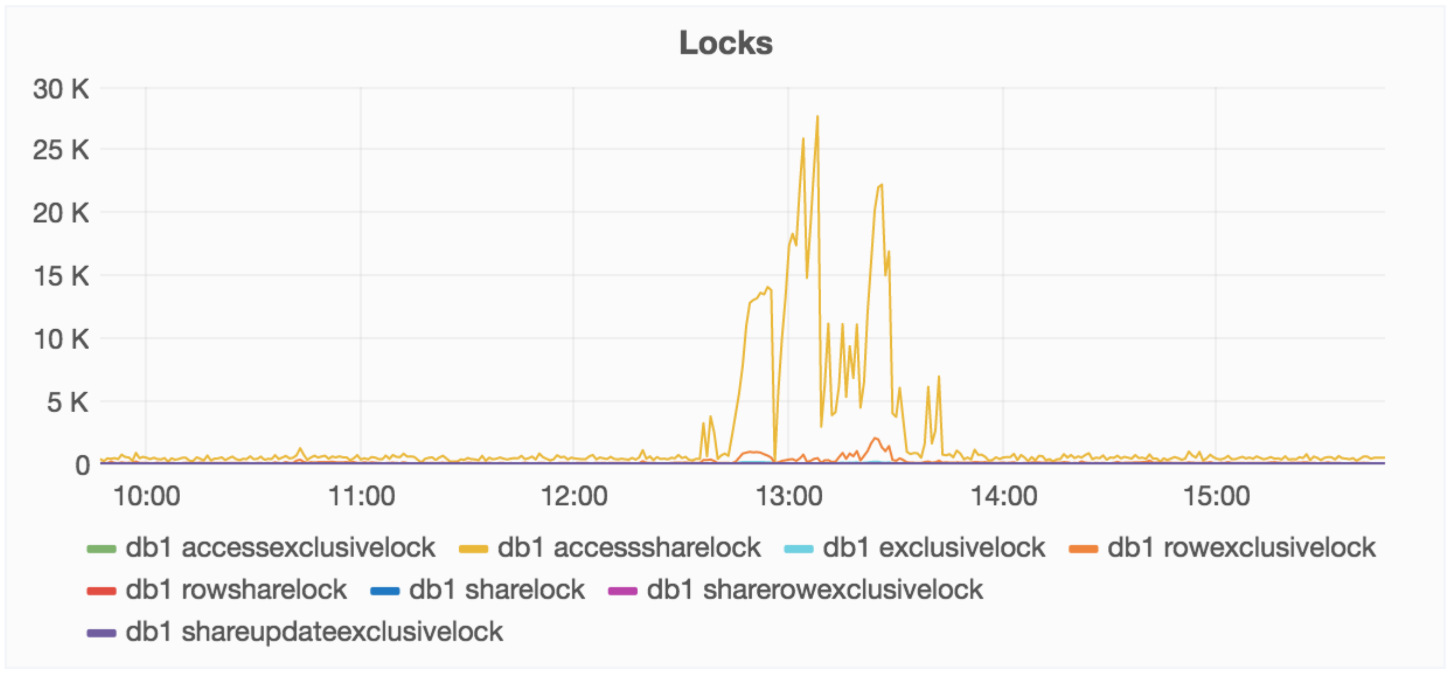
General view of the timeline

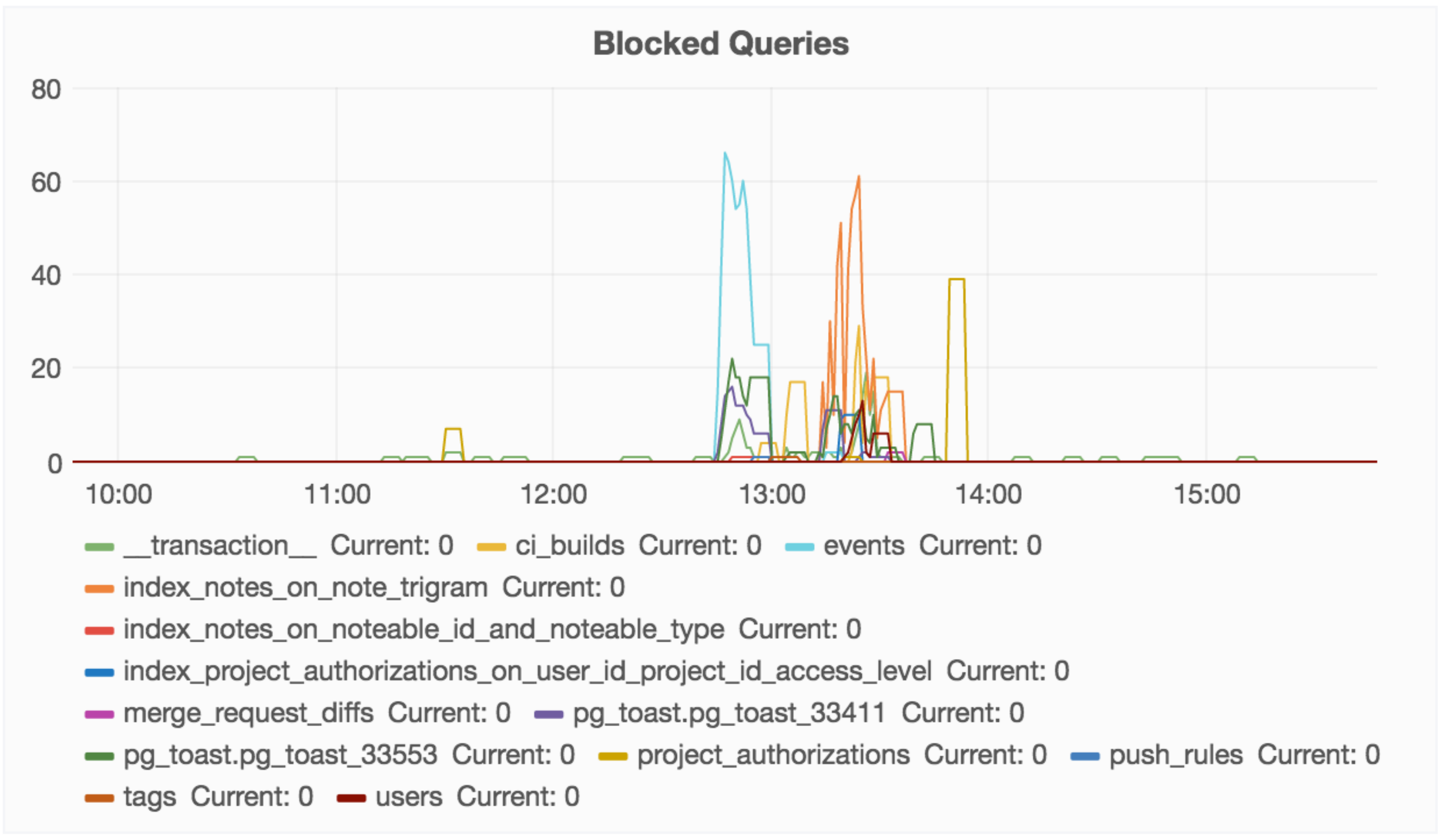


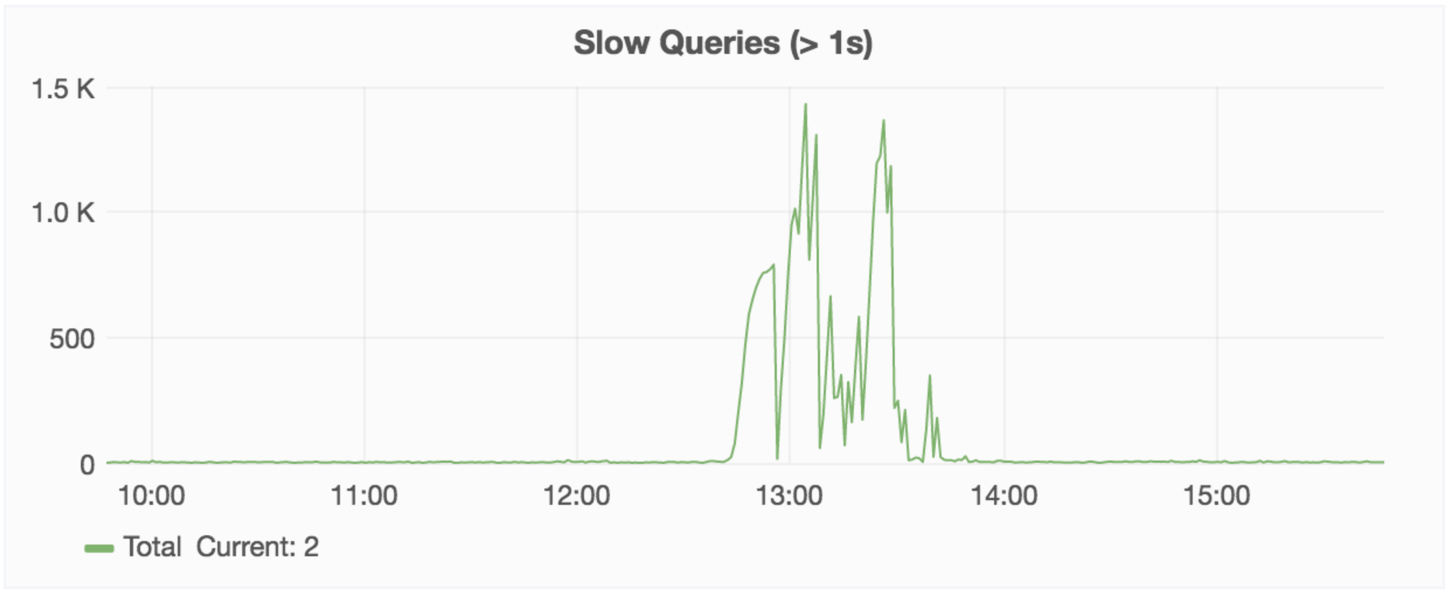


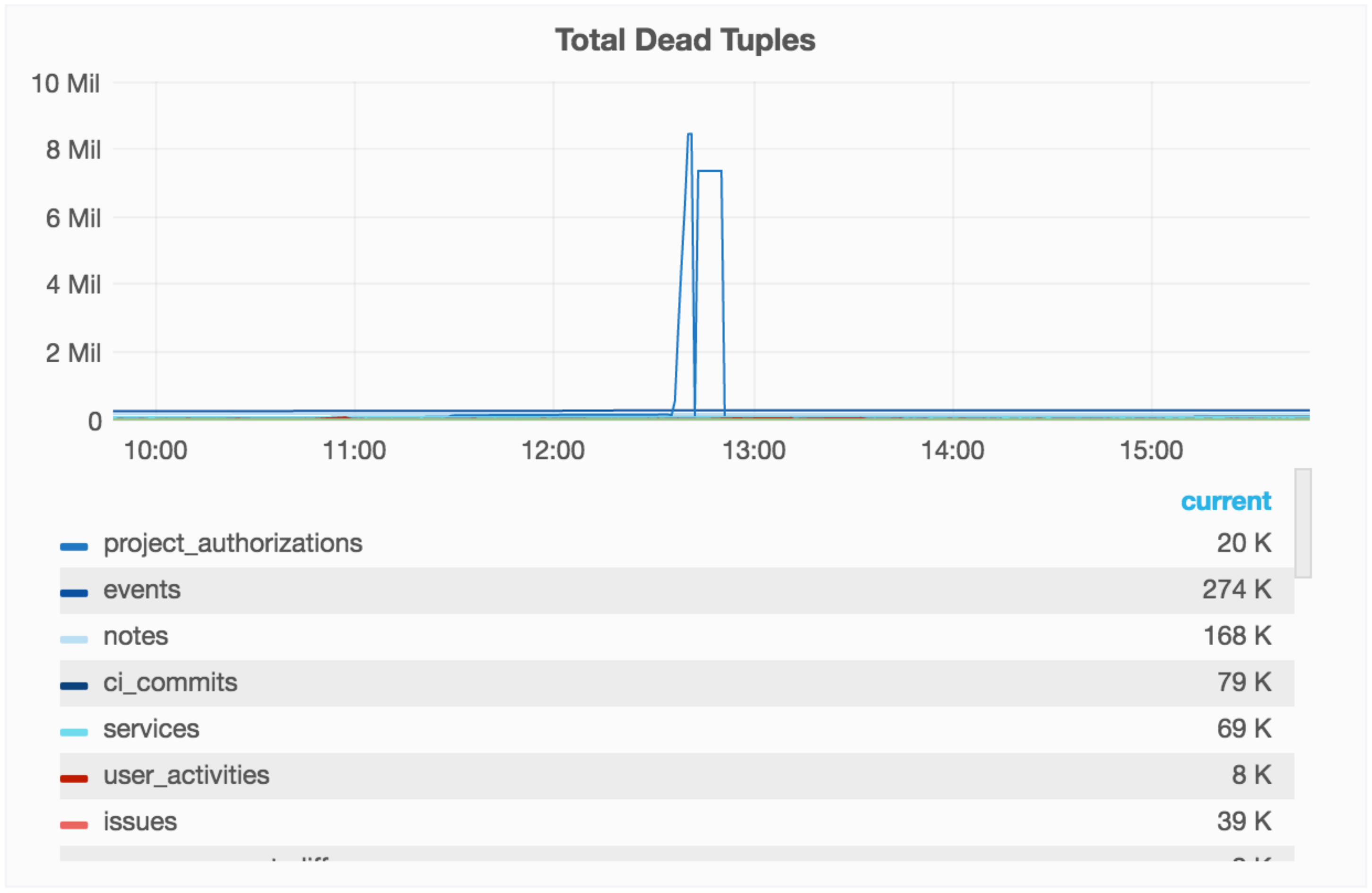



Transactions
20 K
15 K
10 K
5 K
0
10:00
11:00
12:00
13:00
14:00
15:00
db1 committed
db1 rolledback

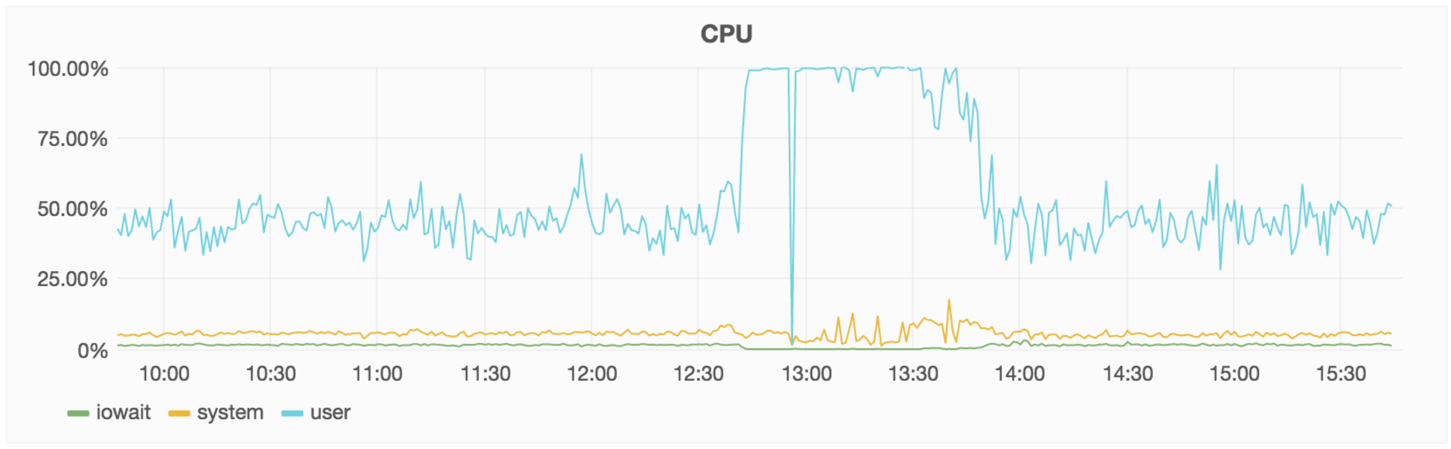

Locks
30 K
25 K
20 K
15 K
10 K
5 K
0
10:00
11:00
12:00
13:00
14:00
15:00
db1 accessexclusivelock
db1 accesssharelock
db1 exclusivelock
db1 rowexclusivelock
db1 rowsharelock
db1 sharelock
db1 sharerowexclusivelock
db1 shareupdateexclusivelock

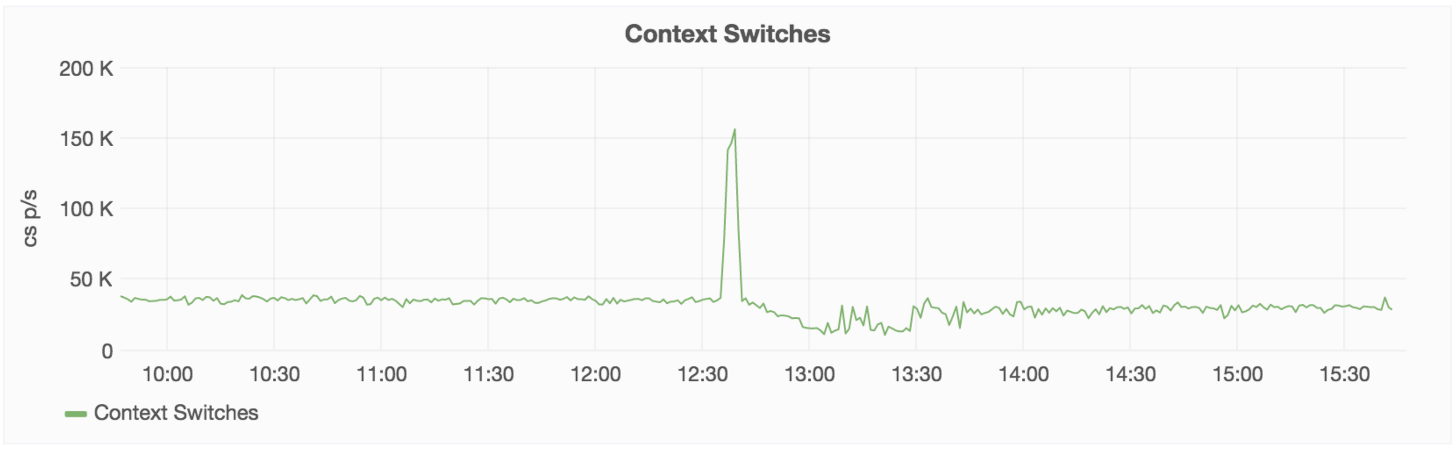

Blocked Queries
80
60
40
20
0
10:00
11:00
12:00
13:00
14:00
15:00
__transaction__ Current: 0
ci_builds Current: 0
events Current: 0
index_notes_on_note_trigram Current: 0
index_notes_on_noteable_id_and_noteable_type Current: 0
index_project_authorizations_on_user_id_project_id_access_level Current: 0
merge_request_diffs Current: 0
pg_toast.pg_toast_33411 Current: 0
pg_toast.pg_toast_33553 Current: 0
project_authorizations Current: 0
push_rules Current: 0
tags Current: 0
users Current: 0

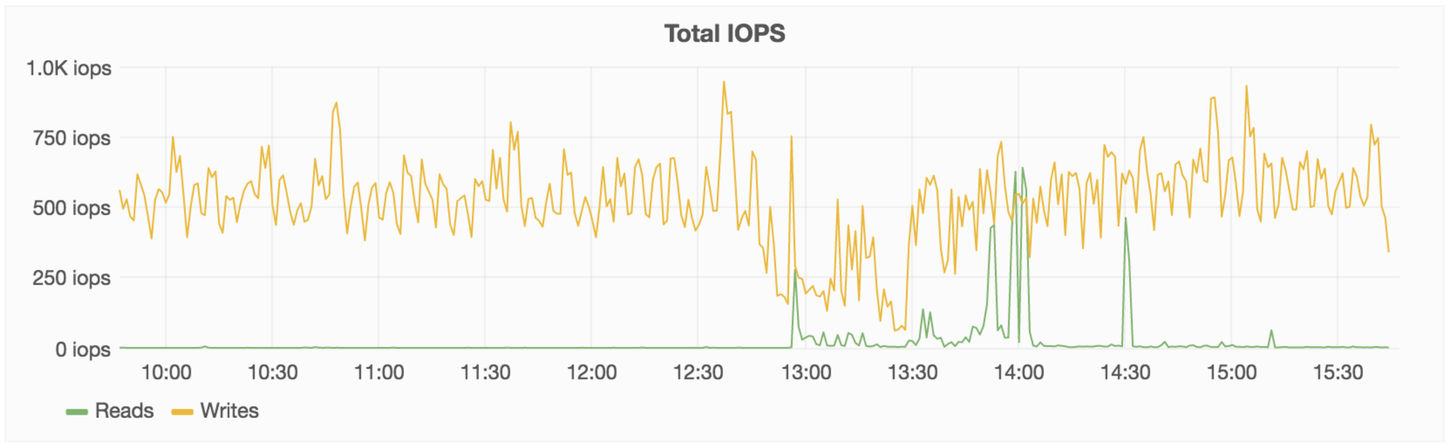

Slow Queries (> 1s)
1.5 K
1.0 K
500
0
10:00
11:00
12:00
13:00
14:00
15:00
Total Current: 2

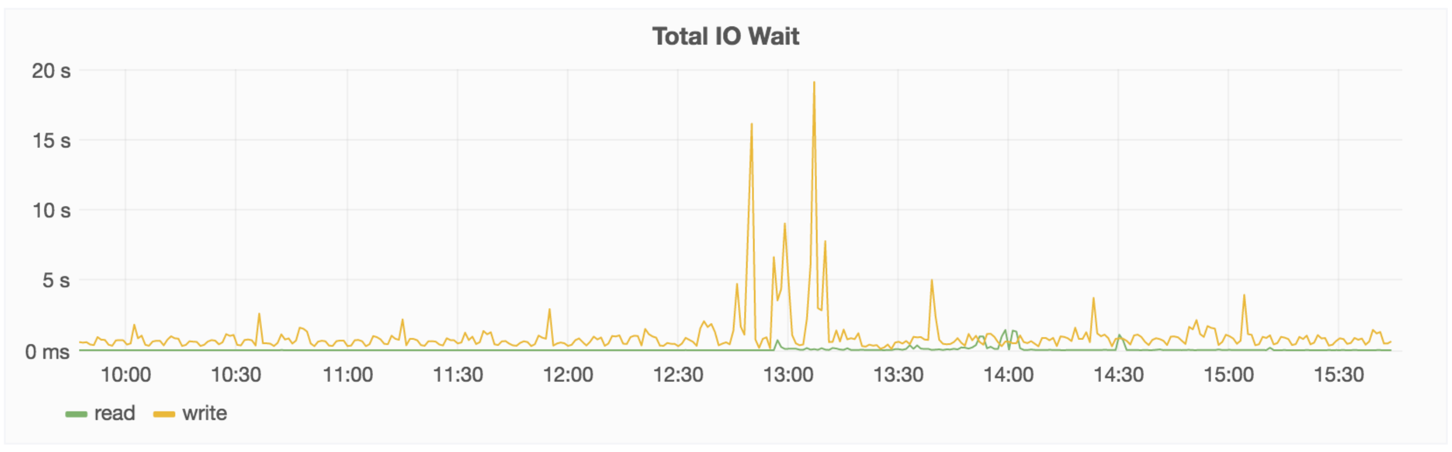


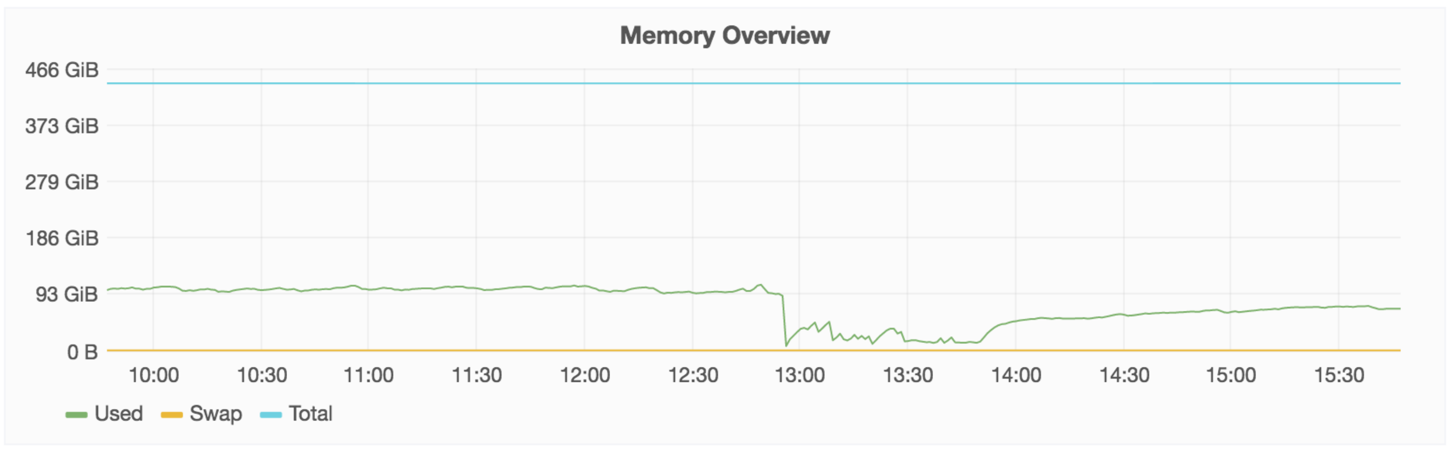
DB1 host metrics



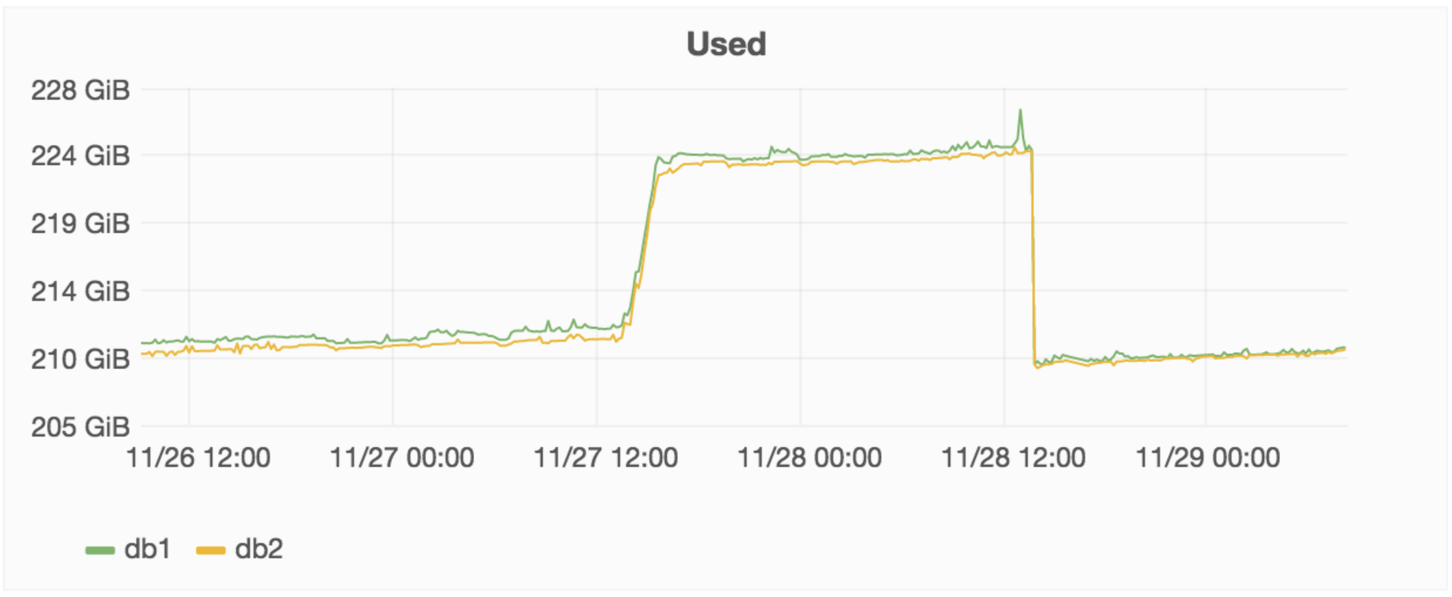




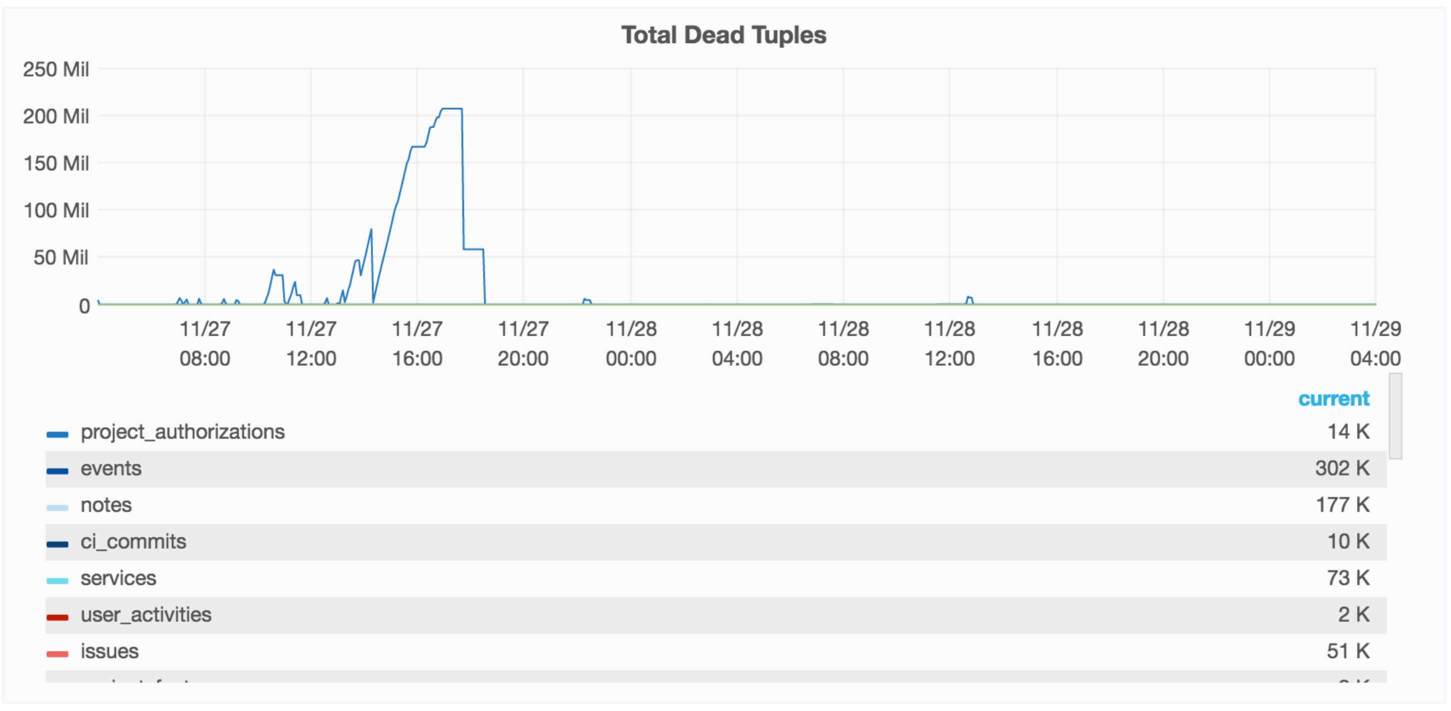




Wide view of DB1 storage usage



Inline with what was going on with dead tuples, which makes the 8M dead tuples look like a glitch



	current
project_authorizations	14 K
events	302 K
notes	177 K
ci_commits	10 K
services	73 K
user_activities	2 K
issues	51 K

What went wrong

- Between the point where the high load started (12:43), to the point when I was alerted by @jnijhof (12:50) to the point when we got a pingdom page (13:08) there were 25 minutes of extremely high latency and downtime. Our paging sucked here.
- There where no helper scripts in the database servers, since we changed database servers recently we lost all those scripts that we kept throwing in `/root` in those hosts.
- We had no way to clearly understand what the table bloat is, we need to consider a way of understanding this.
- We were originally not aware of `VACUUM FULL` performing an aggressive storage reclaim fixing table fragmentation.

- We don't have pg_repack available in the new database to use it in the case it is necessary for a similar situation.

What can be better

- [x] Add alerts for database high load
- [ ] Add page for when GitLab.com stops replying completely
- [x] Add alert for high number of slow queries in the database
- [x] Add alert for high rate of dead tuples per minute (10k for more than 5 minutes is a lot and will bring up issues like this again)
- [x] Add low level host metrics to the database dashboard
- [x] Add availability graphs to the fleet overview dashboard for clarity
- [x] Add database helper scripts to the chef recipe so we can always find the scripts and we don't need to build them while in the middle of an outage.
- [ ] Add pg_repack extension in the new databases to have it as an extra tool for this kind of issues.
- [ ] Update the PG under heavy load runbook with all the findings from this outage to keep it up to date

cc/ @jnijhof @ahanselka @stanhu @northrup @ahmadsherif @maratkalibek @eReGeBe @yorickpeterse

Linked issues 0

Yorick Peterse @yorickpeterse · 3 years ago Maintainer

For those reading: the project authorisations problem is solved by https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/7733 which will be released in 8.14.2.

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

Thanks @yorickpeterse

I would like to move the conversation into understanding what happened to get all that table bloat and how it flew undetected for so long

Pablo Carranza [GitLab] @pcarranza-gitlab Marked the task **Add availability graphs to the fleet overview dashboard for clarity** as completed 3 years ago

Yorick Peterse @yorickpeterse · 3 years ago Maintainer

@pcarranza Prior to https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/7733 the code would use a serializable transaction. The idea was to prevent the data from the same user being updated concurrently as this could potentially result in an inconsistent state. This worked as follows:

```
loop until success
  start serializable transaction
    delete existing entries for user X
    insert new entries for user X
    if commit failed
      retry
```

While the removal worked as inspected the queries would block each other on the insert. This meant that if data had to be refreshed for user A, B, and C all of this would happen in serial. Because this code would try to run until success (much like your typical compare-and-swap loop) this could lead to a lot of queries being executed (and failing). I suspect that due to the `DELETE` succeeding this would create a lot of dead tuples, though I expect that not to happen until a transaction commits (but I'm not familiar enough with these internals to be certain). This would explain the VACUUM overhead, the space usage, the dead tuples, etc.

The new setup is basically:

```
lock entries for user X
start regular transaction
  delete entries for user X
  insert new entries for user X
  commit
unlock entries for user X
```

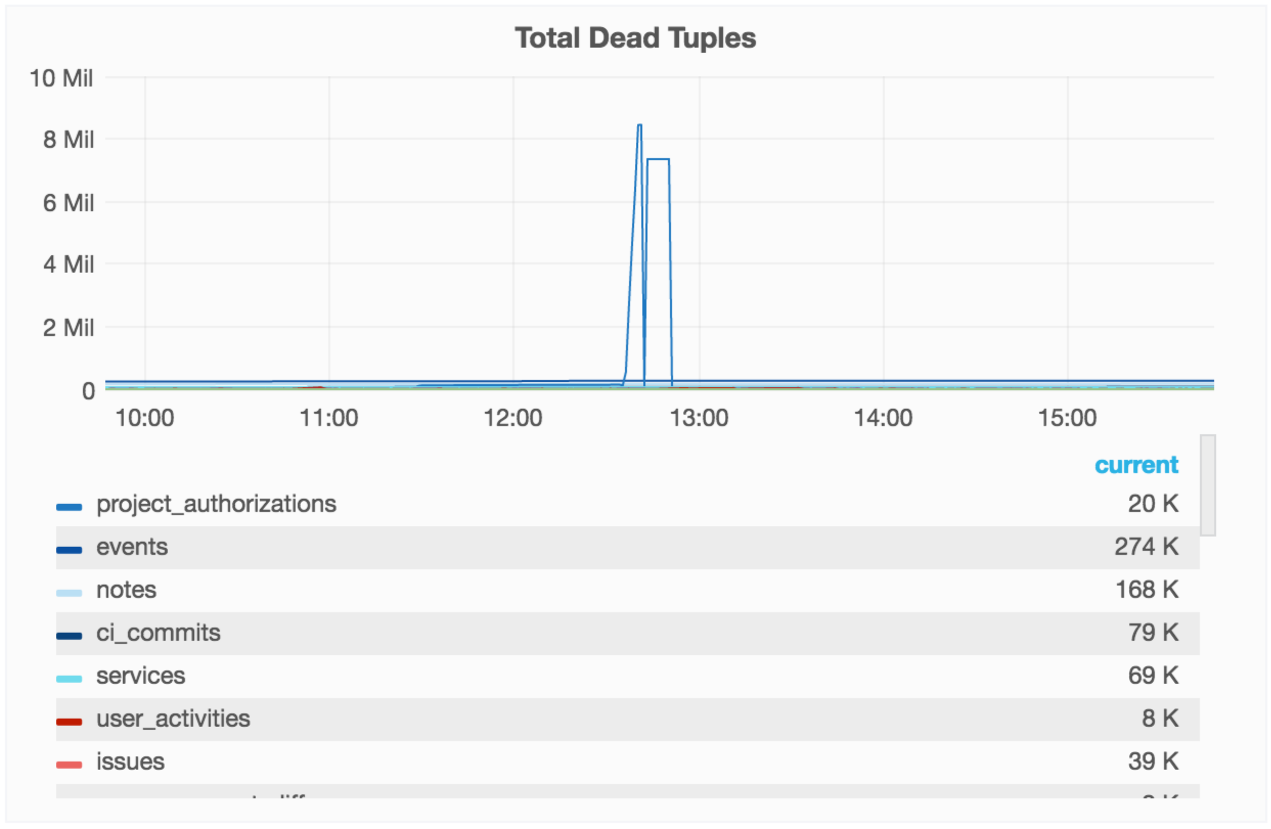
This ensures there's ever only 3 queries being executed (DELETE, SELECT, and an INSERT), and there's no busy loop needed anymore.

Edited by Yorick Peterse 3 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

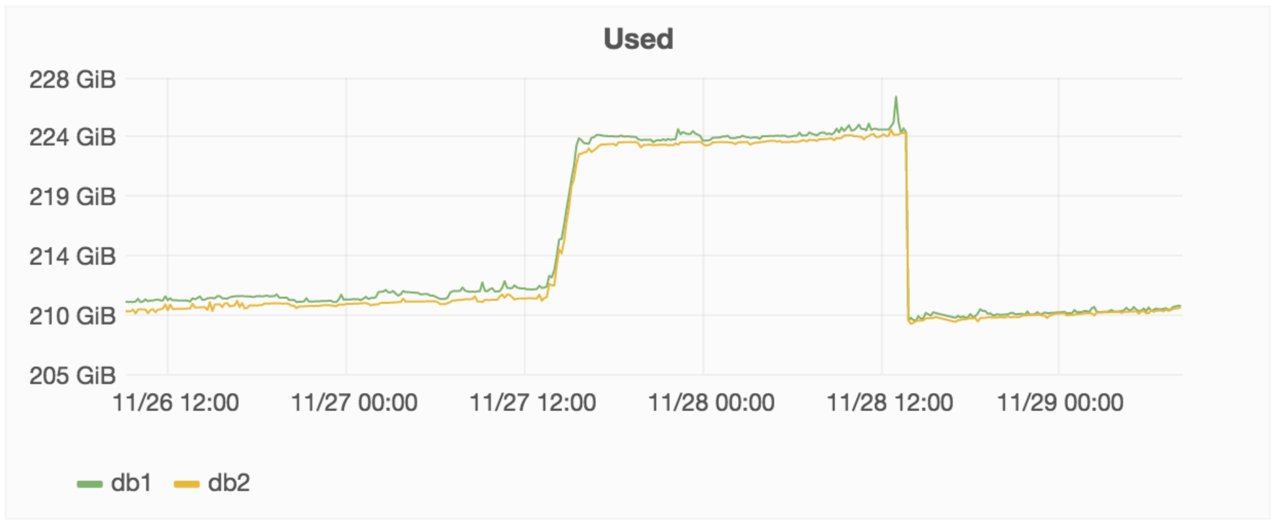
@yorickpeterse I was writing an answer and started wondering.

This graph is showing the spike in dead tuples:

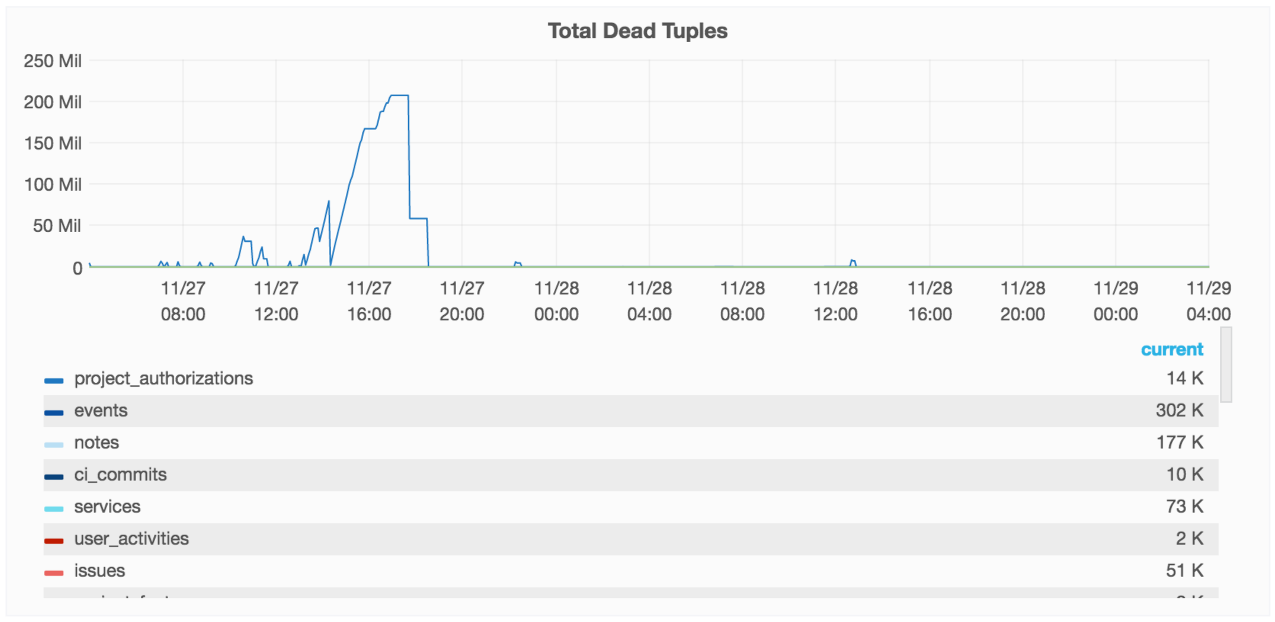


	current
project_authorizations	20 K
events	274 K
notes	168 K
ci_commits	79 K
services	69 K
user_activities	8 K
issues	39 K

But it doesn't explain why we had so much storage used:



Taking a second look at dead tuples in the same timespan look what I found:



	current
project_authorizations	14 K
events	302 K
notes	177 K
ci_commits	10 K
services	73 K
user_activities	2 K
issues	51 K

So an 8M dead tuple peak is not a peak, it's barely an annoyance.

There you go... we need to alert whenever we reach 100M dead rows, or we need to have a way of alerting whenever we see a huge increase in dead rows and storage usage for a given time. I've no idea how to do this yet.

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

Good thing that we fixed the Tuple Texas Chainsaw Massacre feature, @yorickpeterse

Edited by Pablo Carranza [GitLab] 3 years ago

Chris ☯ @MrChrisW · 3 years ago

Support received a request (handled by @athar) from a GitLab.com user on ˋÄ̧ʹÂ̂ǻǻ̀Ä regarding an issue "not seeing all created projects on the dashboard page". As @athar did not have staging/production access he was unable to debug the issue (https://gitlab.com/gitlab-com/infrastructure/issues/801 + improvements to SE onboarding). This was escalated and debugged on ˋÄ̧ʹÂ̂ǻǻ̀Ä̧. Steps taken:

1. Impersonate the user and review dashboard - missing a number of created projects
2. Load the rails console and review `current_user.authorized_projects` - same result
3. Check the users `authorized_projects_populated` attribute = `true`
4. Updated `authorized_projects_populated to false` and refreshed the users dashboard - issue resolved

We attempted to first debug this issue on staging.gitlab.com (before screwing with production), however it was borked - https://gitlab.com/gitlab-com/infrastructure/issues/802

We should work to identify these **occasional** issues when first reported (via support). This will involve having cross-team communication from support to dev or infra if we receive a report from a GitLab.com user describing unusual behaviour.

Edited by Chris 3 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

@MrChrisW I fail to see how this relates to the outage specifically. But I agree with the comms feedback.

Chris ☯ @MrChrisW · 3 years ago

@pcarranza I'm not saying it relates to this outage specifically, I'm making the point that we saw the symptoms related to https://gitlab.com/gitlab-org/gitlab-ce/merge_requests/7733 early (-1day) before it started causing problems in production.

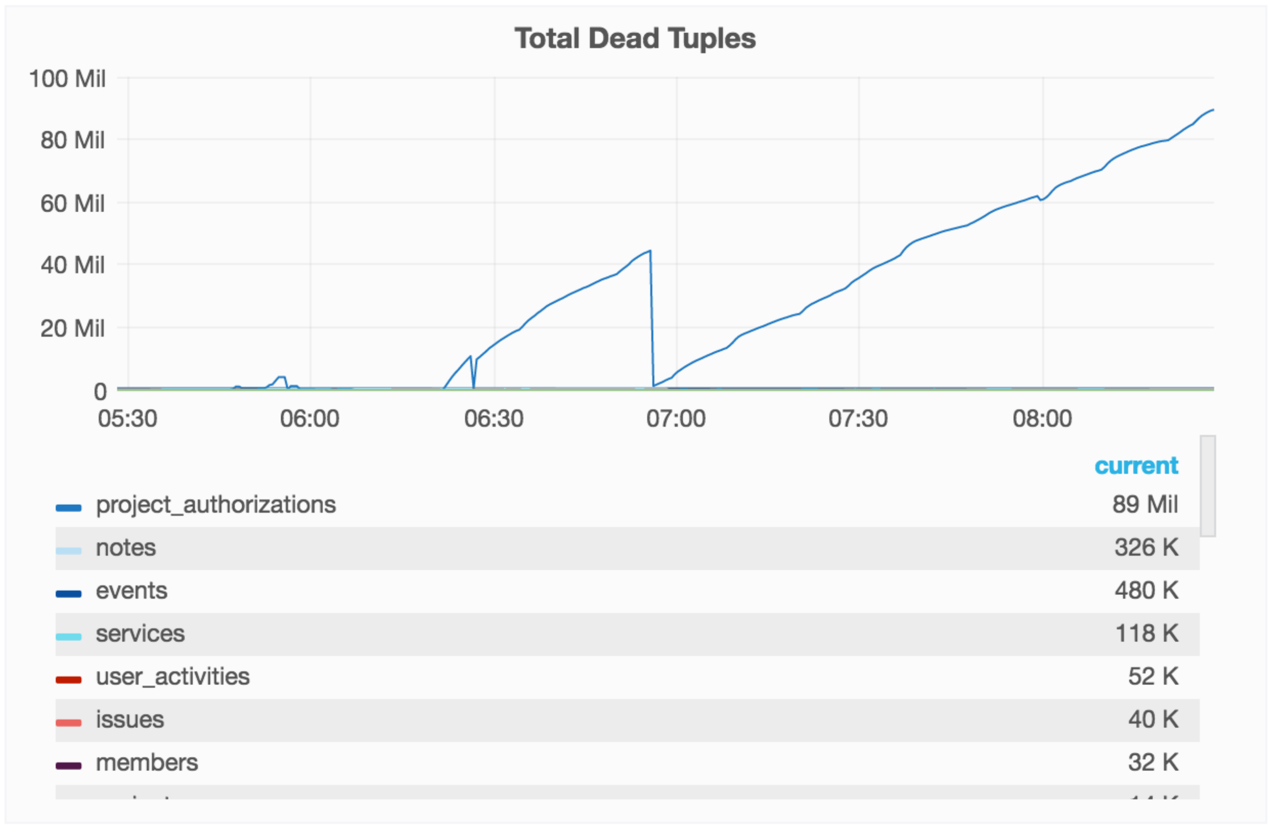
Edited by Chris 3 years ago

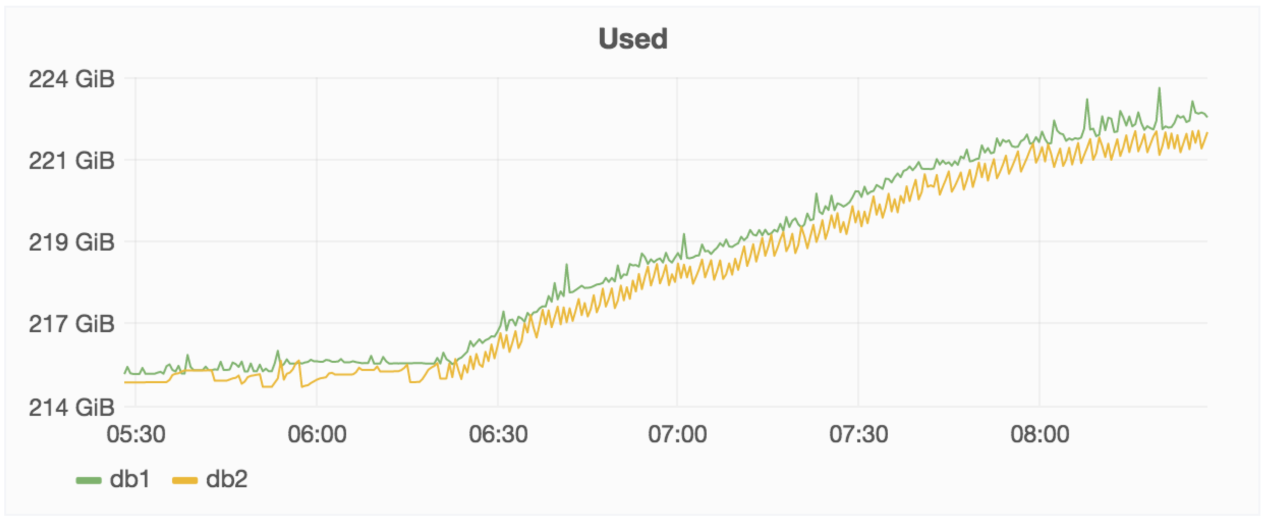
Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

Makes sense, thanks @MrChrisW

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

We have another tuple chainsaw massacre going on right now



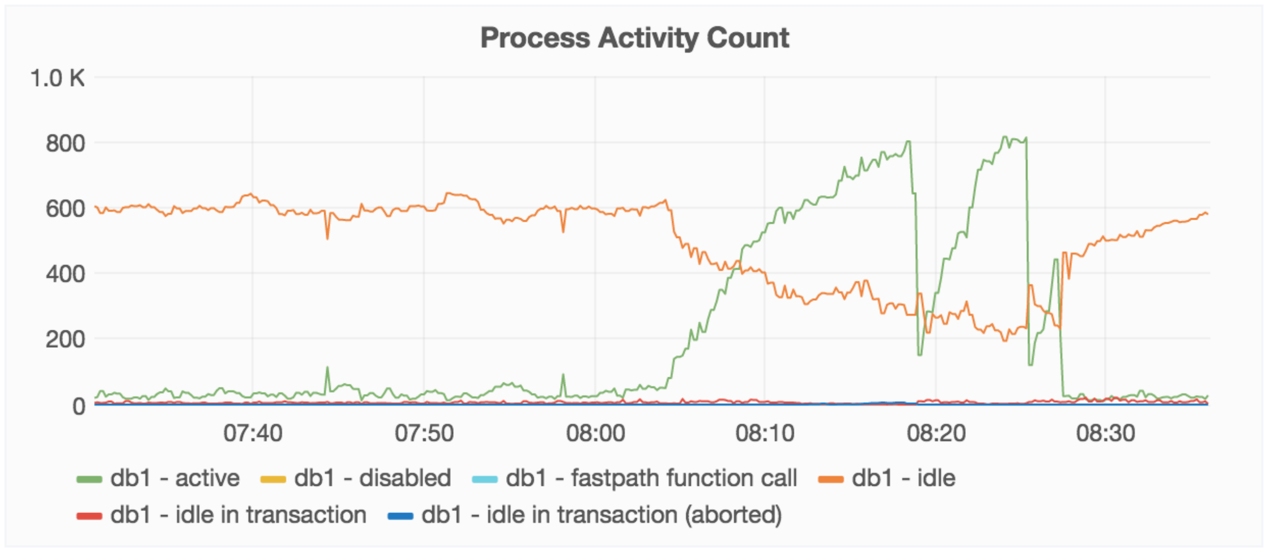


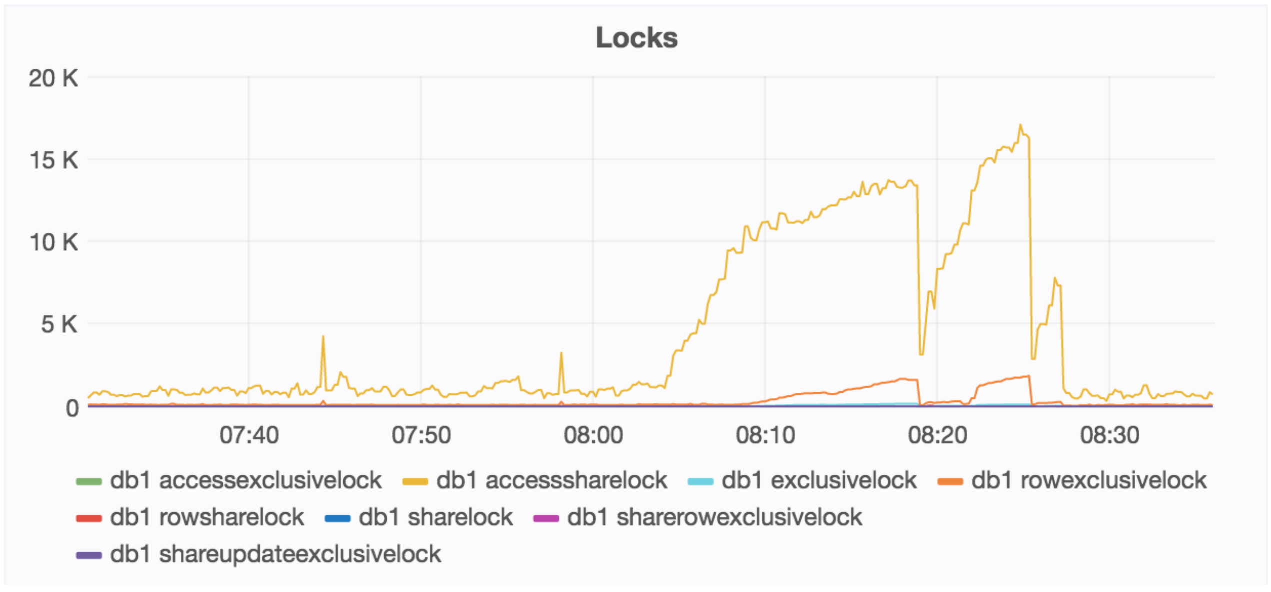
Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

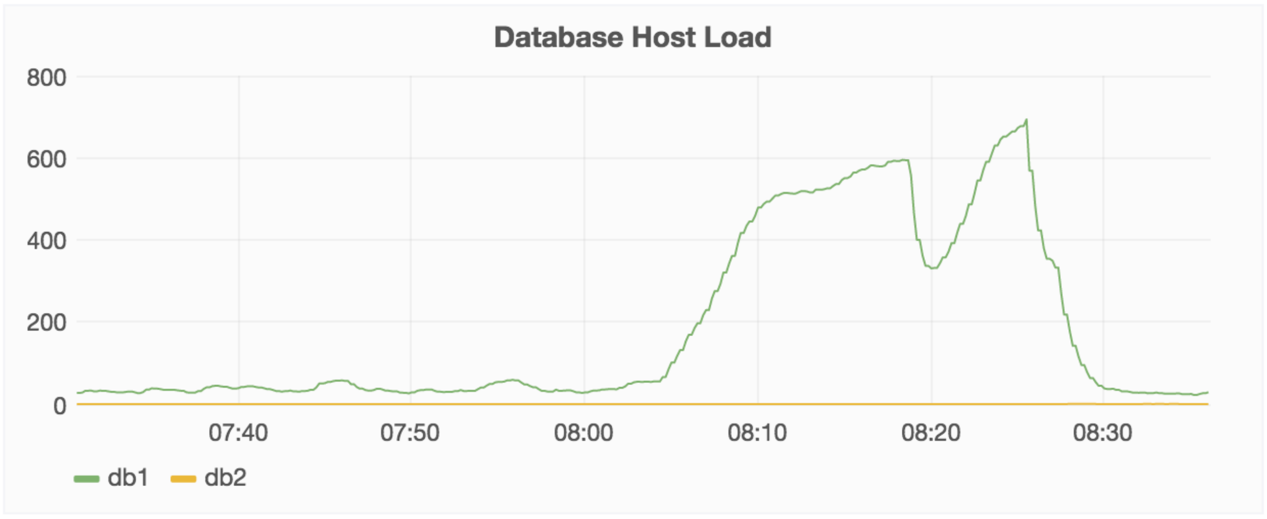
I'll clean up the pile of dead bodies when the dust settles.

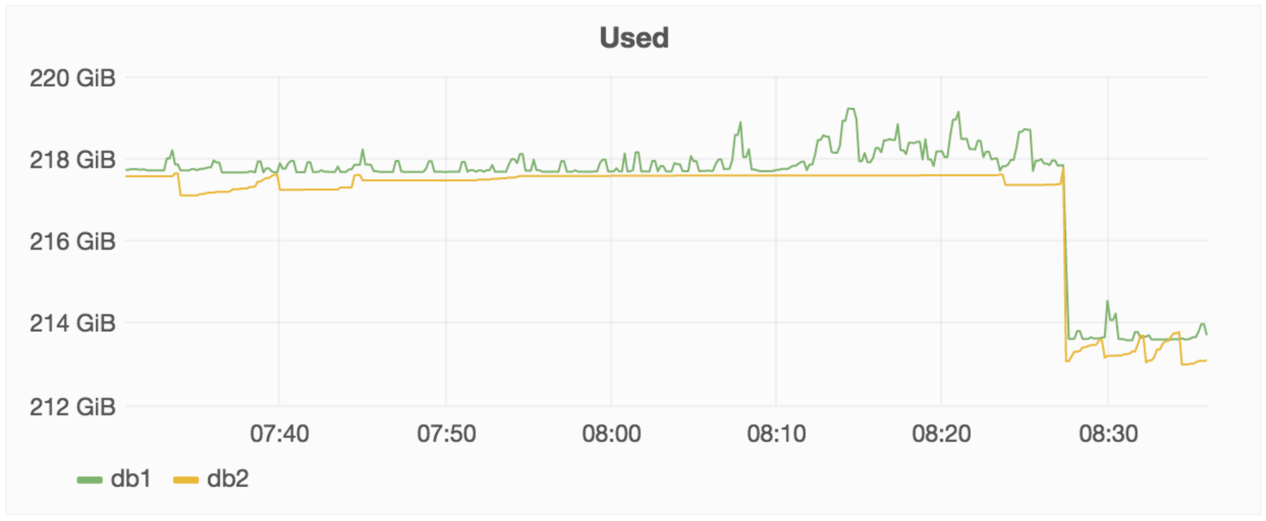
Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

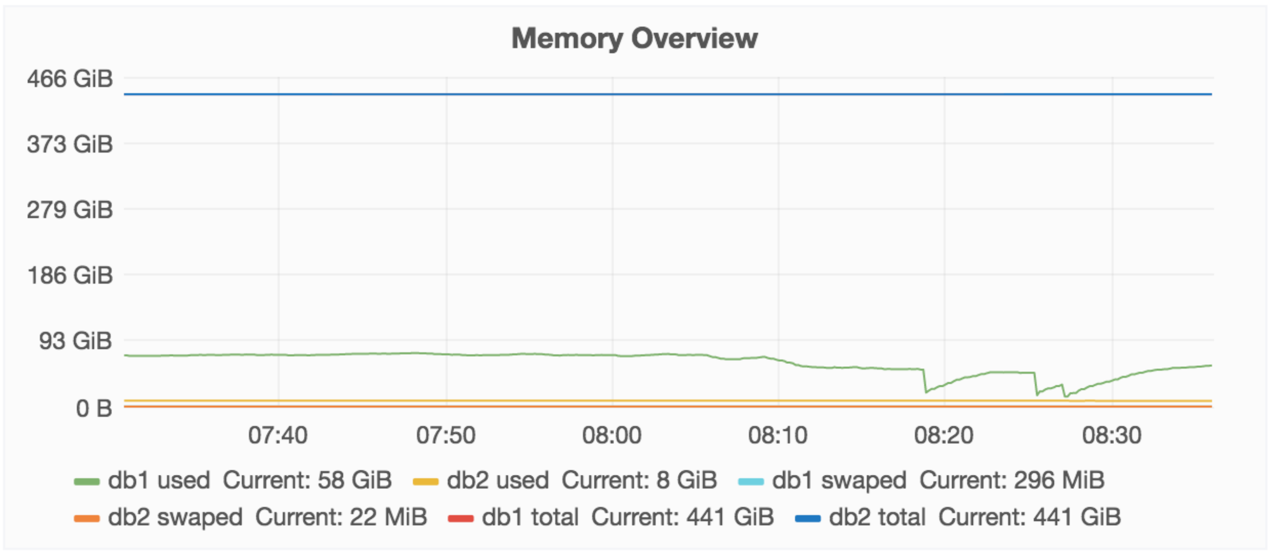
Another event today morning with the same behavior and resolution

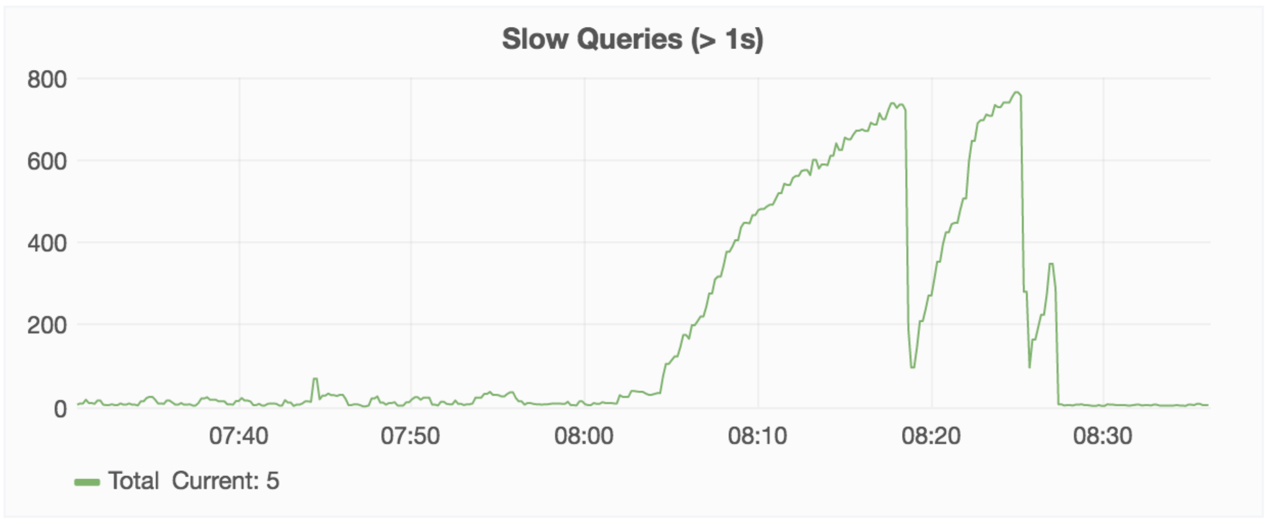


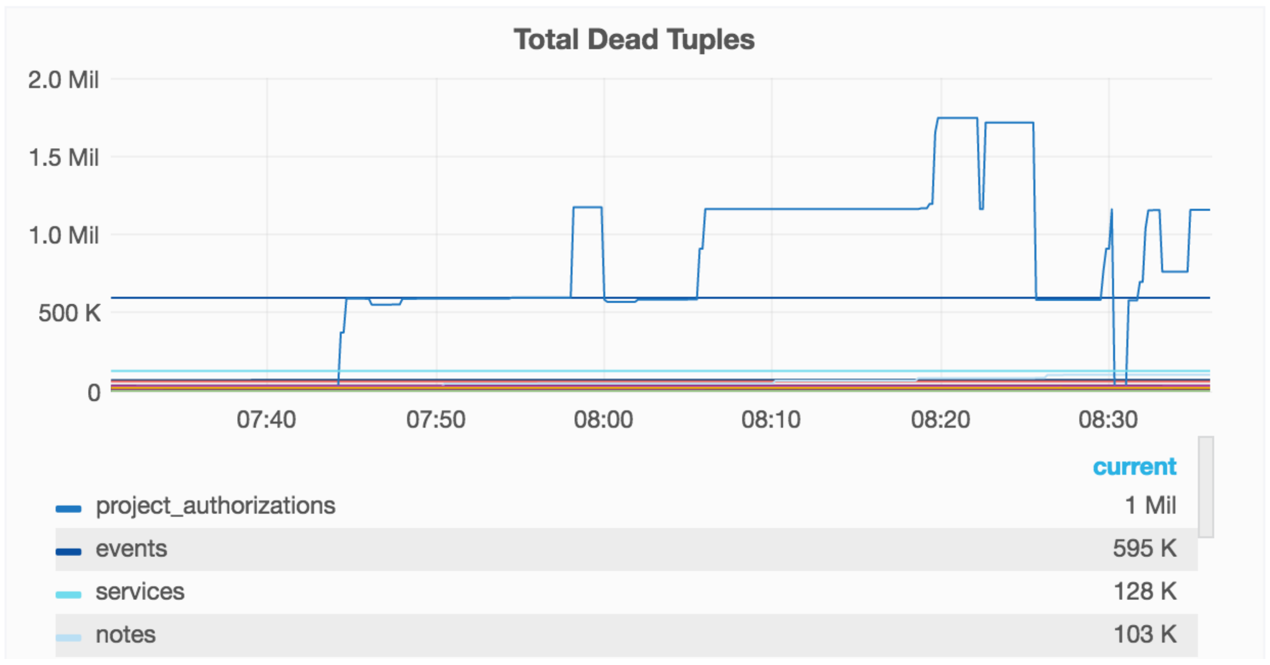

Locks
20 K
15 K
10 K
5 K
0
07:40
07:50
08:00
08:10
08:20
08:30
db1 accessexclusivelock
db1 accesssharelock
db1 exclusivelock
db1 rowexclusivelock
db1 rowsharelock
db1 sharelock
db1 sharerowexclusivelock
db1 shareupdateexclusivelock

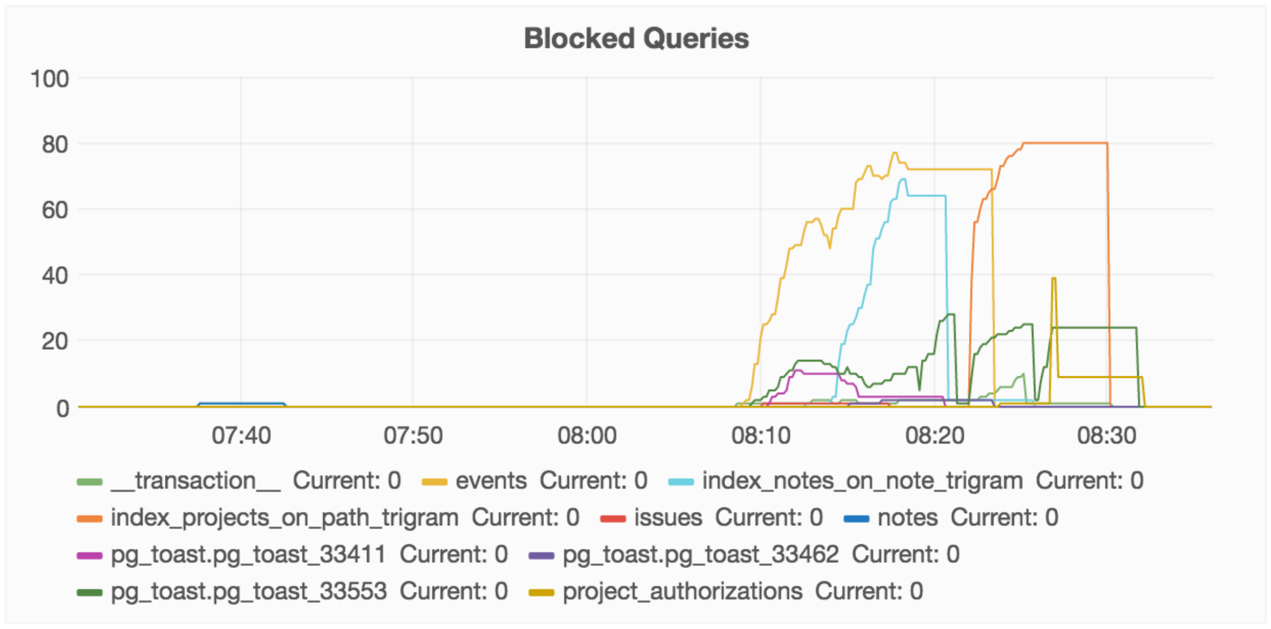

Database Host Load
800
600
400
200
0
07:40
07:50
08:00
08:10
08:20
08:30
db1
db2


Used
220 GiB
218 GiB
216 GiB
214 GiB
212 GiB
07:40
07:50
08:00
08:10
08:20
08:30
db1
db2


Memory Overview
466 GiB
373 GiB
279 GiB
186 GiB
93 GiB
0 B
07:40
07:50
08:00
08:10
08:20
08:30
db1 used Current: 58 GiB
db2 used Current: 8 GiB
db1 swaped Current: 296 MiB
db2 swaped Current: 22 MiB
db1 total Current: 441 GiB
db2 total Current: 441 GiB


Slow Queries (> 1s)
800
600
400
200
0
07:40
07:50
08:00
08:10
08:20
08:30
Total Current: 5





Again, `vacuum verbose project_authorizations` and `vacuum full project_authorizations` to reclaim space.

Stan Hu @stanhu · 3 years ago Maintainer

What do we have to do to prevent this from happening again?

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

@stanhu Stop generating so much bloat in the project_authorizations table, or start using pg_repack on this table as it seems to be the main bloat generator available.

@yorickpeterse also mentioned reviewing the way we are updating this table as it seems that we are being too aggressive by basically blowing all the rows and generating them again.

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

From the alerting standpoint, I just added an alert for db high load with a lot of information, and we should probably alert whenever we see that there is a high dead tuple generation rate to get us ready. It will not prevent this from happening, but it will at least warn with some time to be ready.

Edited by Pablo Carranza [GitLab] 3 years ago

Yorick Peterse @yorickpeterse · 3 years ago Maintainer

@pcarranza Now that we essentially use a lock instead of serializable transactions it should be easier to perform a diff based update. I'll see if I can piece this together next week.

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

Thanks @yorickpeterse Issue you created is: https://gitlab.com/gitlab-org/gitlab-ce/issues/25257

Pablo Carranza [GitLab] @pcarranza-gitlab marked the task **Add alerts for database high load** as completed 3 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab marked the task **Add alert for high number of slow queries in the database** as completed 3 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab marked the task **Add alert for high rate of dead tuples per minute (10k for more than 5 minutes is a lot and will bring up issues like this again)** as completed 3 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab marked the task **Add low level host metrics to the database dashboard** as completed 3 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab marked the task **Add database helper scripts to the chef recipe so we can always find the scripts and we don't need to build them while in the middle of an outage.** as completed 3 years ago

Pablo Carranza [GitLab] @pcarranza-gitlab · 3 years ago

I'm closing this issue for now as I'm not clear the next step is to use pg_repack. I think we are going in a different direction here.

Pablo Carranza [GitLab] @pcarranza-gitlab closed 3 years ago

Achilleas Pipinellis 🦅 @axil removed milestone 2 years ago

Andrew Newdigate @andrewn moved to production#92 (closed) 1 year ago

Please register or sign in to reply