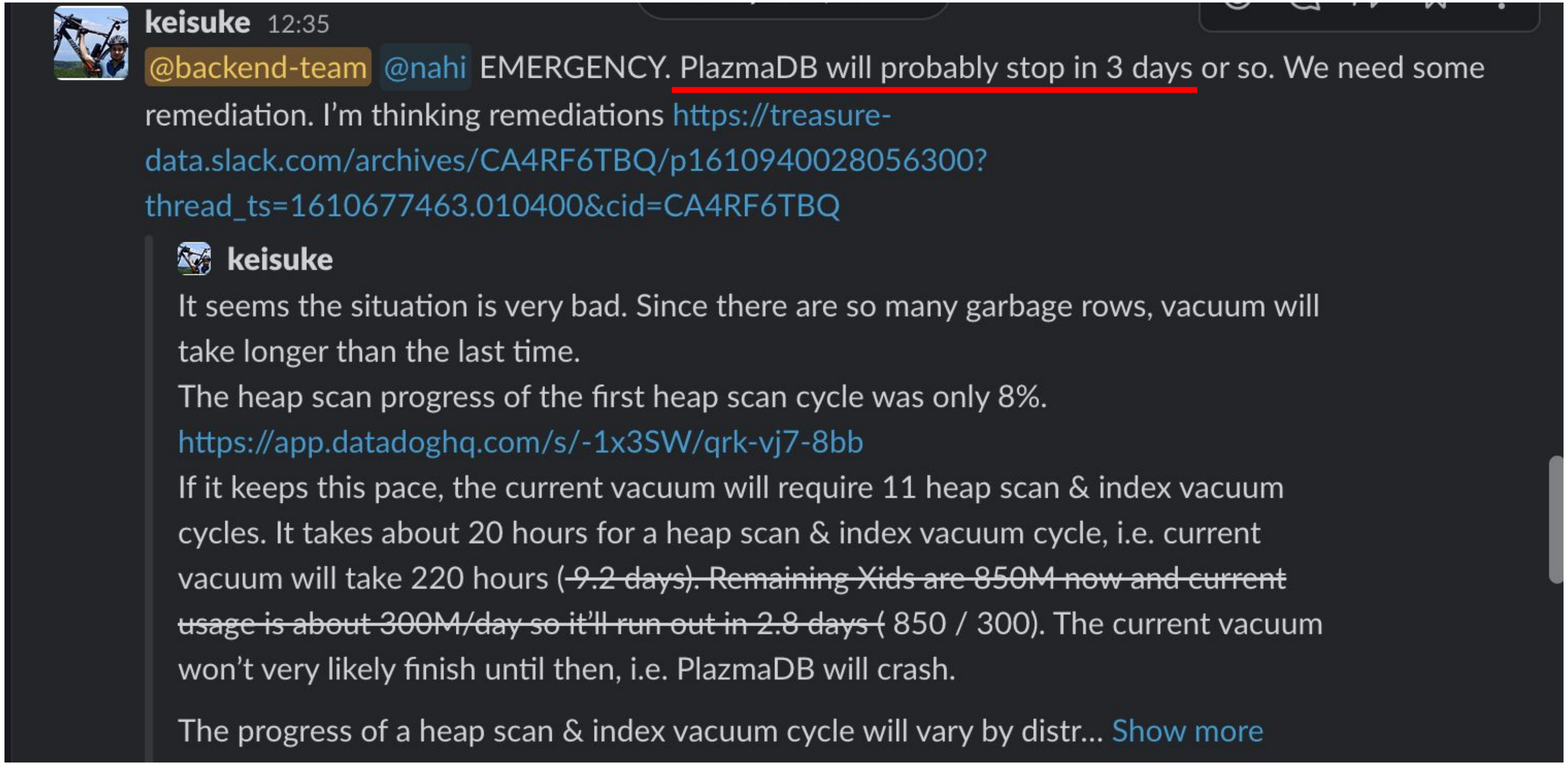
TREASURE DATA

Journey to Improve Stability and Scalability of Plazma

Keisuke Suzuki
Software Engineer (Backend Storage team)

2022/11/29 TD Tech Talk

EMERGENCY



keisuke 12:35

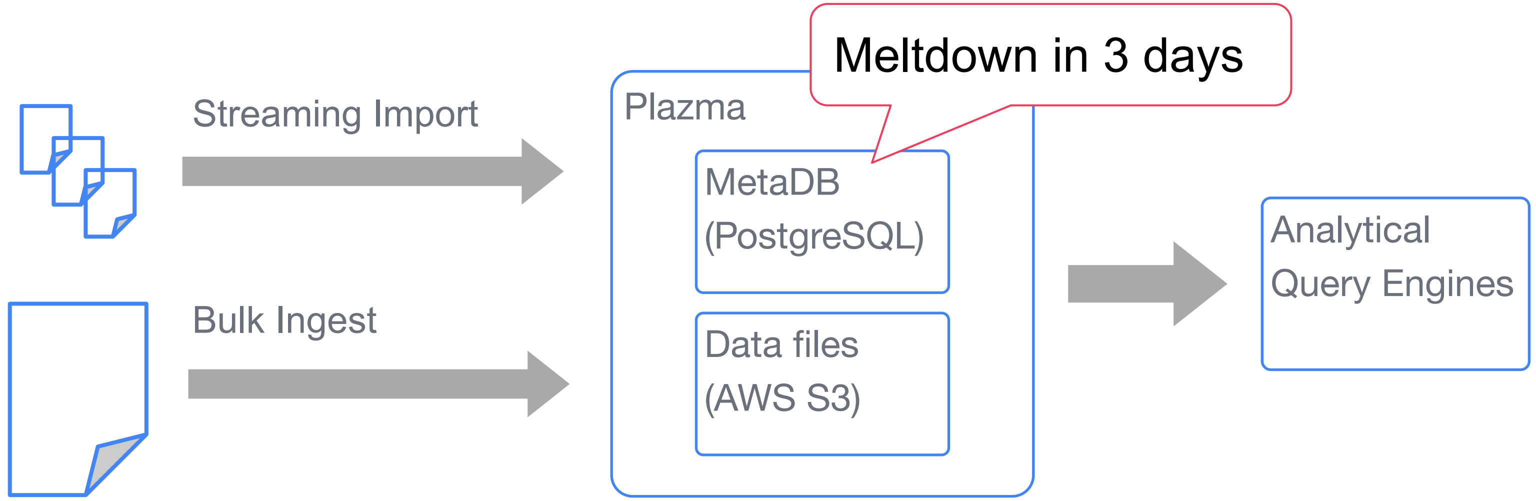
@backend-team @nahi EMERGENCY. PlazmaDB will probably stop in 3 days or so. We need some remediation. I'm thinking remediations https://treasure-data.slack.com/archives/CA4RF6TBQ/p1610940028056300?thread_ts=1610677463.010400&cid=CA4RF6TBQ

keisuke

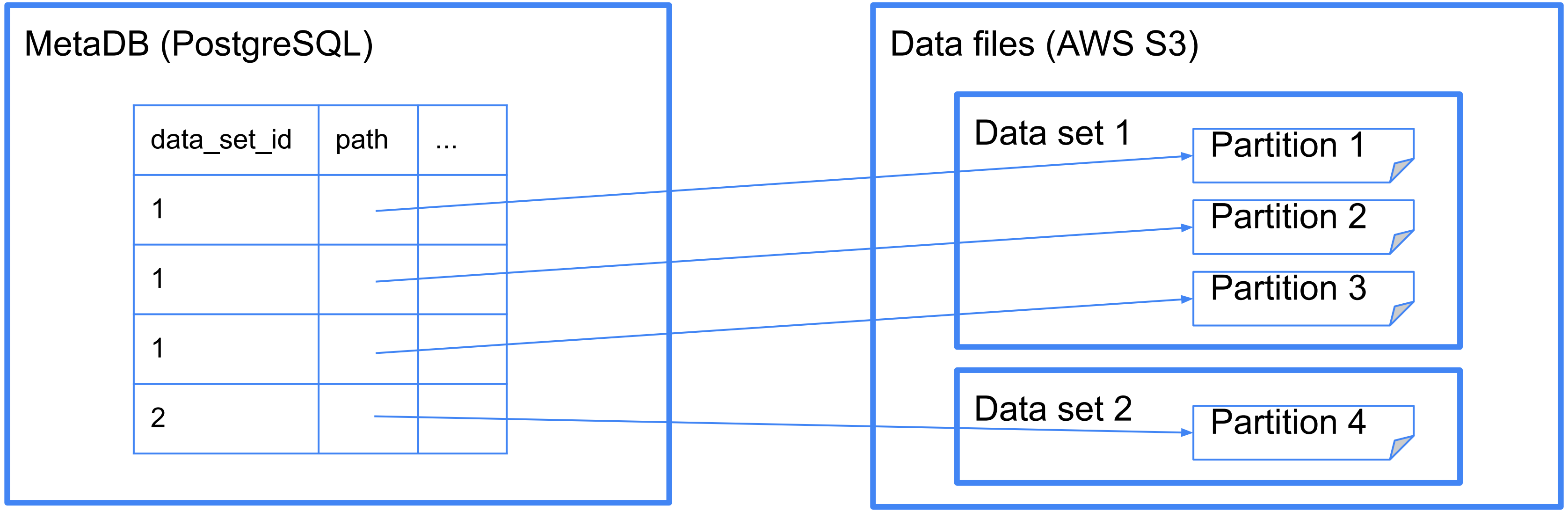
It seems the situation is very bad. Since there are so many garbage rows, vacuum will take longer than the last time.
The heap scan progress of the first heap scan cycle was only 8%.
https://app.datadoghq.com/s/-1x3SW/qrk-vj7-8bb
If it keeps this pace, the current vacuum will require 11 heap scan & index vacuum cycles. It takes about 20 hours for a heap scan & index vacuum cycle, i.e. current vacuum will take 220 hours (~~9.2 days). Remaining Xids are 850M now and current usage is about 300M/day so it'll run out in 2.8 days (~~ 850 / 300). The current vacuum won't very likely finish until then, i.e. PlazmaDB will crash.

The progress of a heap scan & index vacuum cycle will vary by distr... Show more

What happened?

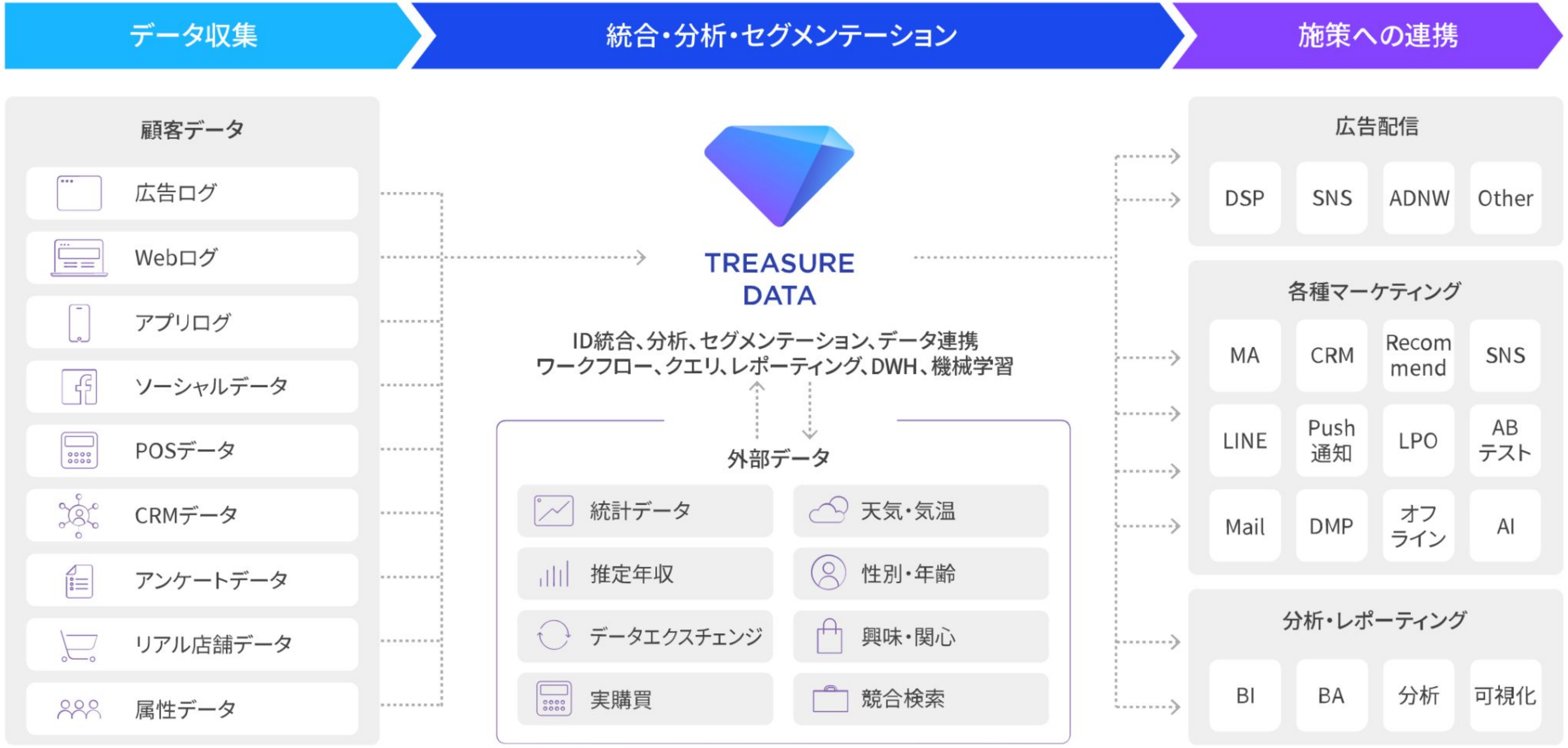


Data Set and Partition Metadata of Plazma

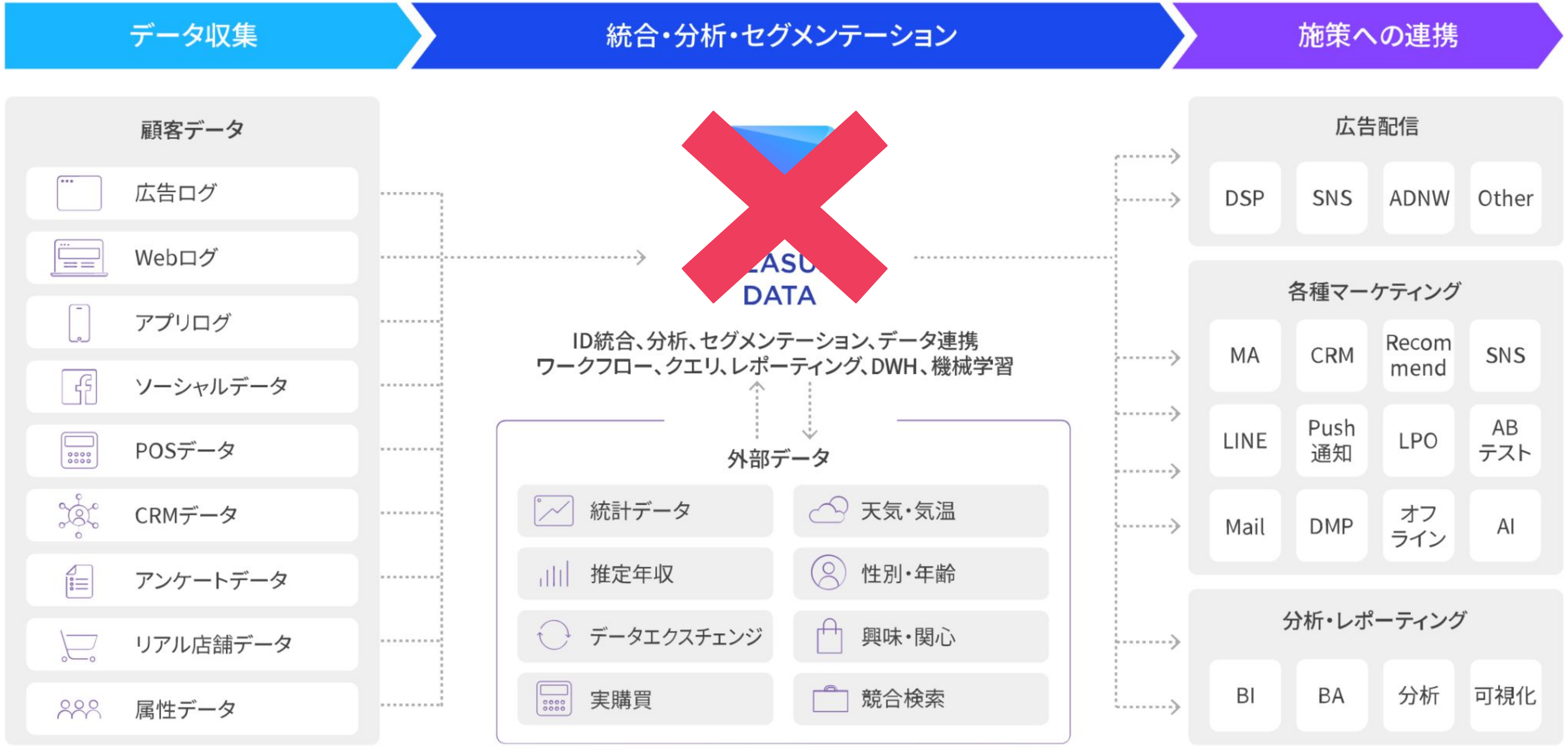


- Data set (Table) is collection of partition files
- MetaDB manages
  - Relationship between data sets and partitions
  - Location of partitions
  - Visibility of partitions etc…

The Worst Case Scenario…



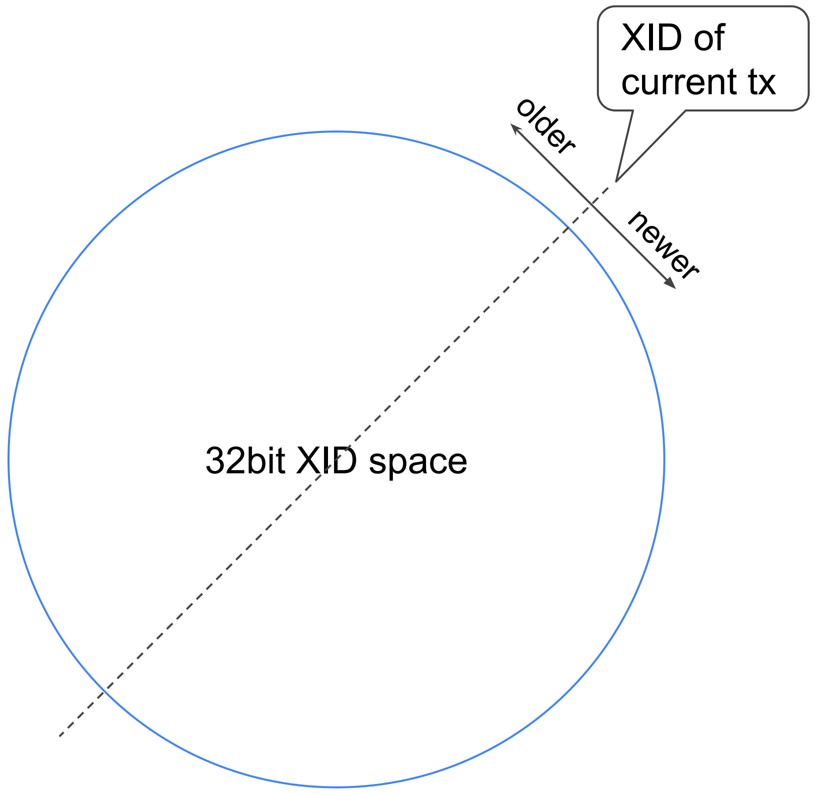
The Worst Case Scenario…



XID Wraparound of PostgreSQL

TransactionID (XID)

- 32bit ID for transaction ordering
- Wraparound when it reaches max



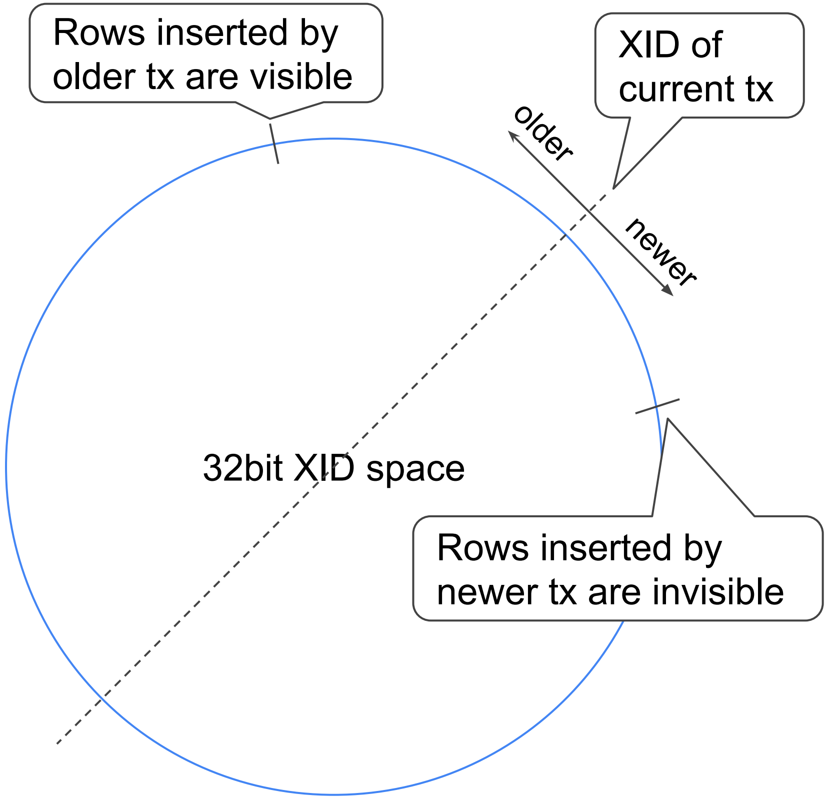
XID Wraparound of PostgreSQL

TransactionID (XID)

- 32bit ID for transaction ordering
- Wraparound when it reaches max

Used for row visibility

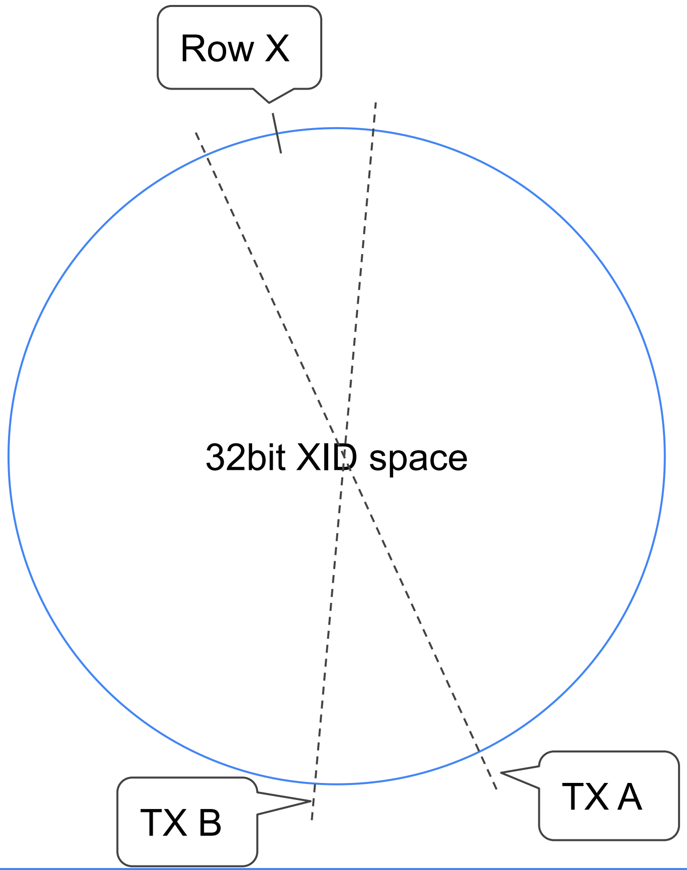
- Row insertion XID < Current XID
  -> Row is visible in TX
- Row insertion XID > Current XID
  -> Row is invisible in TX


Rows inserted by older tx are visible
XID of current tx
older
newer
32bit XID space
Rows inserted by newer tx are invisible

Data Loss by XID Wraparound

Visibility issue of rows with very old XID

- TX B > TX A (i.e. TX B is newer)
- TX A can see Row X
  - Row X is in older side
- TX B should also see Row X but cannot actually
  - Row X is in newer side because of wraparound
    -> Data Loss!!

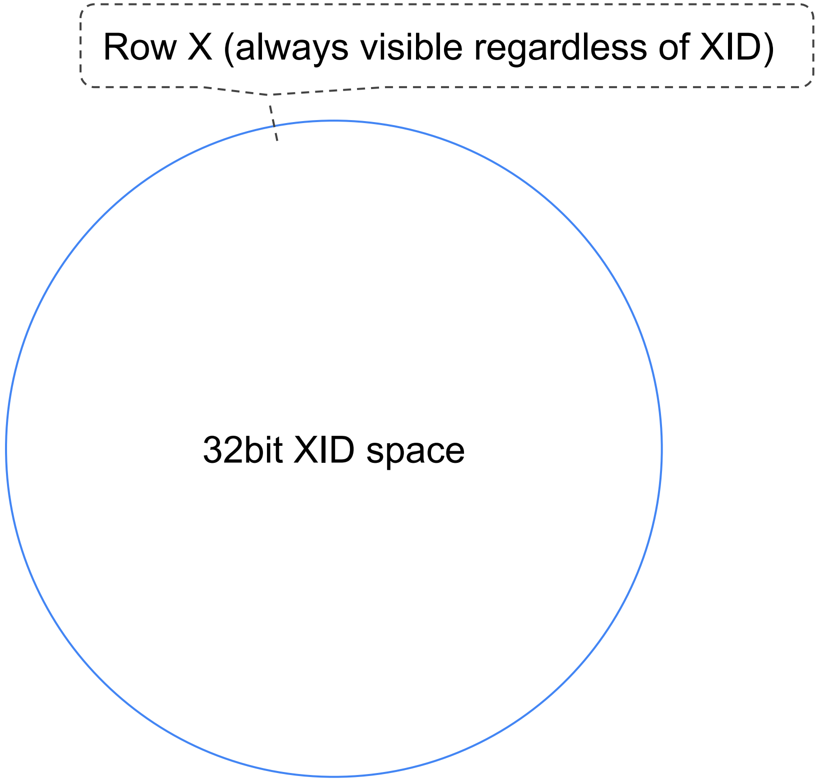


Vacuum to Freeze Old Rows

- Vacuum : table GC / optimization
  - Autovacuum : routine worker to run vacuum automatically

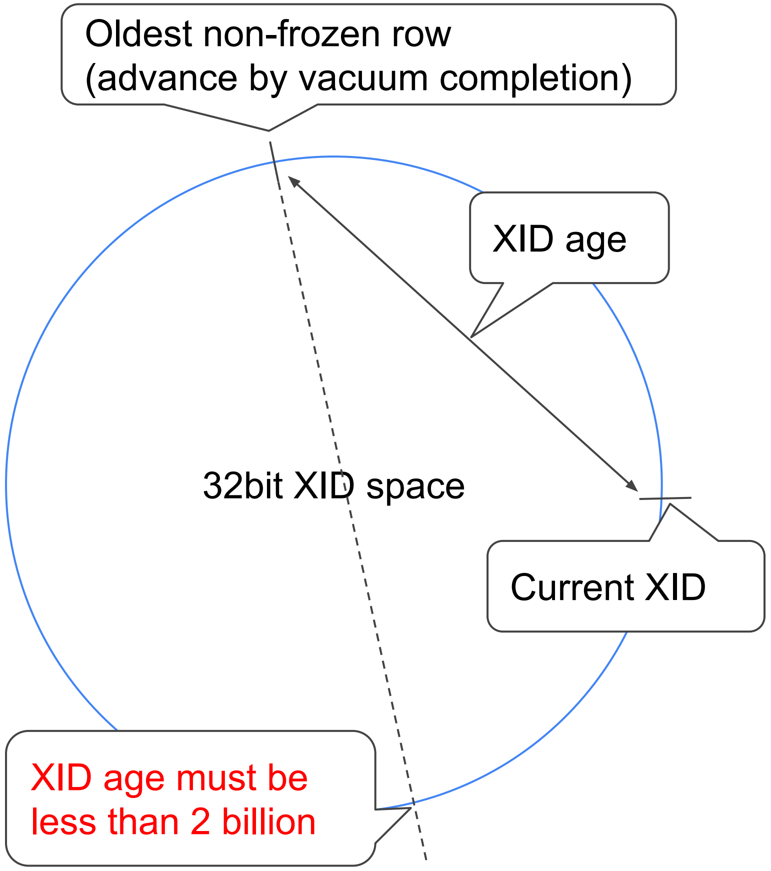
- Vacuum marks old rows as frozen
  - Frozen rows are visible for all txs

(More details on PostgreSQL document)



Freeze must complete before wraparound

- PostgreSQL shut down if it's close to data loss
  - When XID age hits 1.999 billion (= 1 million XIDs are left)
- Ensure
  Vacuum speed > XID consumption
  - Vacuum progress and XID age should be monitored for large DB
- Vacuum takes long time when:
  - Table is large
  - Many dead rows (tuples) in table


Oldest non-frozen row
(advance by vacuum completion)
XID age
32bit XID space
Current XID
XID age must be less than 2 billion

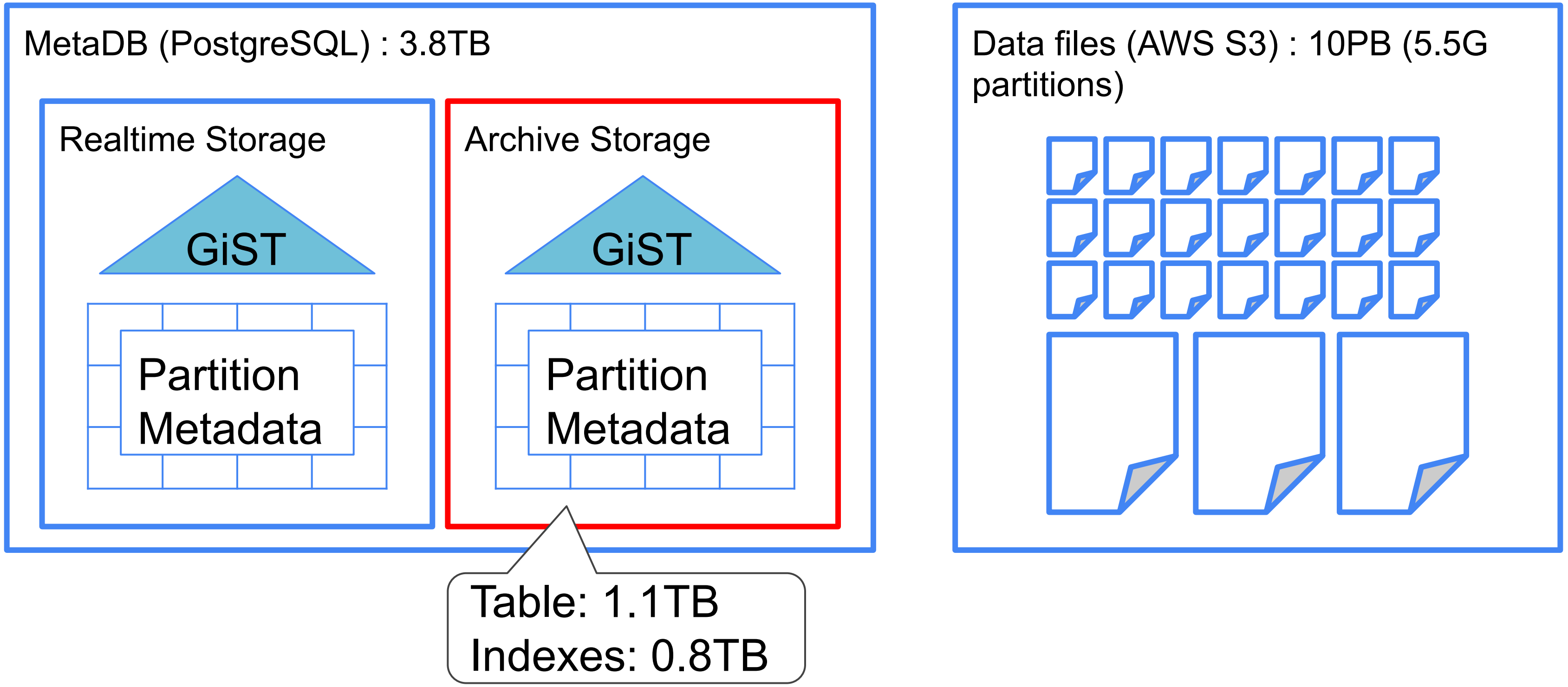
XID Consumption Rate of Plazma MetaDB

- Avg XID consumption rate = 2.9k XIDs / sec = 250M XIDs / day

- Time to see data loss
  - We put XID age soft limit on 600M (autovacuum_freeze_max_age)
    - 1400M XIDs are available (= 2000M - 600M)

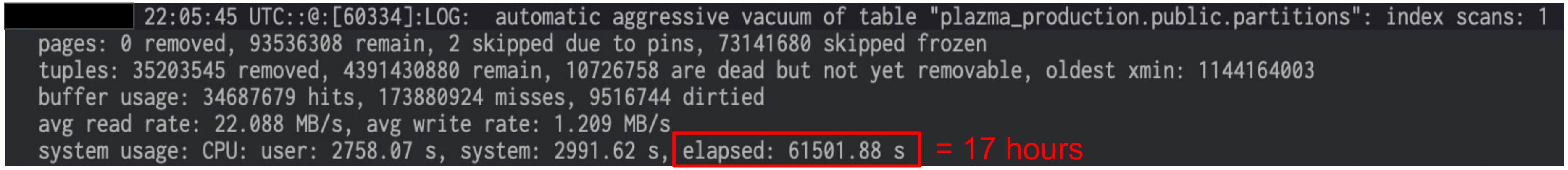
  - Time to hit XID age 2 billion
    = Time to run out available XIDs
    = 5.6 days (= 1400M / 250M/day)

Data Volume of Plazma



Vacuum Execution Time on Plazma MetaDB

- Vacuum on the largest table usually took for 12-24 hours
  -> This is small enough than time to see data loss (5.6 days)



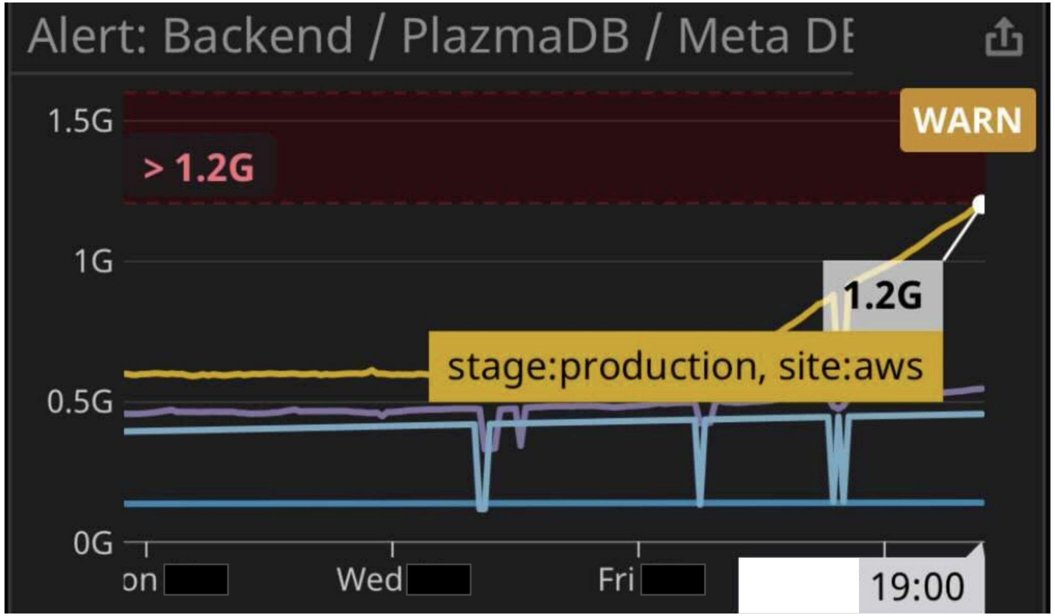
```
[redacted] 22:05:45 UTC::@:[60334]:LOG:  automatic aggressive vacuum of table "plazma_production.public.partitions": index scans: 1
  pages: 0 removed, 93536308 remain, 2 skipped due to pins, 73141680 skipped frozen
  tuples: 35203545 removed, 4391430880 remain, 10726758 are dead but not yet removable, oldest xmin: 1144164003
  buffer usage: 34687679 hits, 173880924 misses, 9516744 dirtied
  avg read rate: 22.088 MB/s, avg write rate: 1.209 MB/s
  system usage: CPU: user: 2758.07 s, system: 2991.62 s, elapsed: 61501.88 s
```

= 17 hours

So what went wrong?

Sudden Growth of XID age

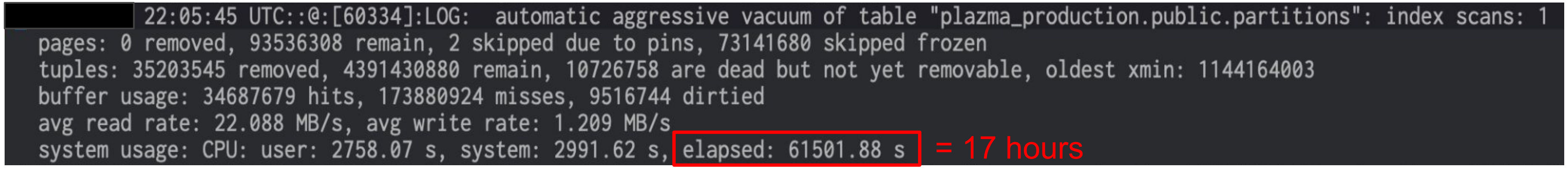
1 day before my EMERGENCY post, XID reached 1.2 billion
Which is much higher than soft limit 600M



XID age

Vacuum Took Longer

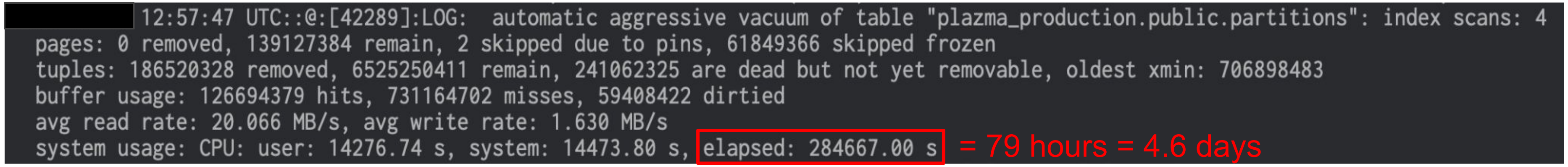
- This time, vacuum completed before running out XIDs
- But execution time of vacuum was much longer than usual
  - Usually



```
[redacted] 22:05:45 UTC::@:[60334]:LOG:  automatic aggressive vacuum of table "plazma_production.public.partitions": index scans: 1
  pages: 0 removed, 93536308 remain, 2 skipped due to pins, 73141680 skipped frozen
  tuples: 35203545 removed, 4391430880 remain, 10726758 are dead but not yet removable, oldest xmin: 1144164003
  buffer usage: 34687679 hits, 173880924 misses, 9516744 dirtied
  avg read rate: 22.088 MB/s, avg write rate: 1.209 MB/s
  system usage: CPU: user: 2758.07 s, system: 2991.62 s, elapsed: 61501.88 s
```

= 17 hours

  - This time



```
[redacted] 12:57:47 UTC::@:[42289]:LOG:  automatic aggressive vacuum of table "plazma_production.public.partitions": index scans: 4
  pages: 0 removed, 139127384 remain, 2 skipped due to pins, 61849366 skipped frozen
  tuples: 186520328 removed, 6525250411 remain, 241062325 are dead but not yet removable, oldest xmin: 706898483
  buffer usage: 126694379 hits, 731164702 misses, 59408422 dirtied
  avg read rate: 20.066 MB/s, avg write rate: 1.630 MB/s
  system usage: CPU: user: 14276.74 s, system: 14473.80 s, elapsed: 284667.00 s
```

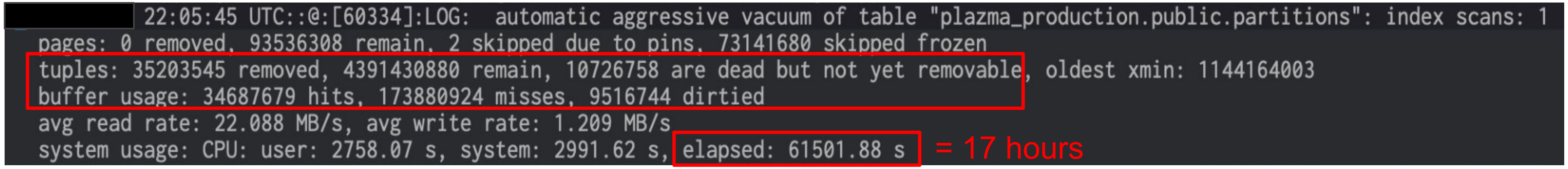
= 79 hours = 4.6 days

Factors of Long Vacuum

- Following factors were mainly contributed on the largest table
  - Number of dead rows
    - Increase when partition file is removed from Plazma
      -> Depends on customer workload
  - Bloat of GiST index
    - PostgreSQL before 12 had bloat issue on GiST index (We used 11)

Why did it took so long?

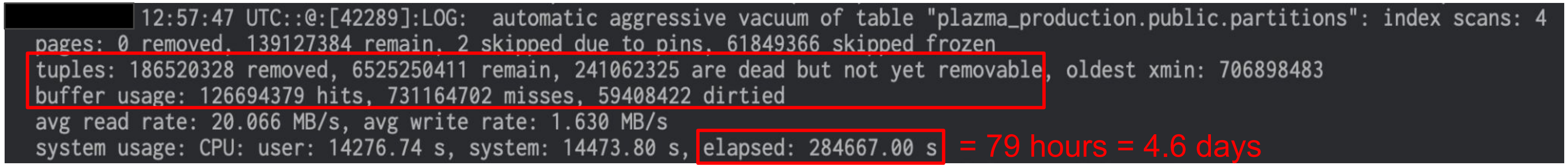
- Usually



```
[redacted] 22:05:45 UTC::@:[60334]:LOG:  automatic aggressive vacuum of table "plazma_production.public.partitions": index scans: 1
  pages: 0 removed, 93536308 remain, 2 skipped due to pins, 73141680 skipped frozen
  tuples: 35203545 removed, 4391430880 remain, 10726758 are dead but not yet removable, oldest xmin: 1144164003
  buffer usage: 34687679 hits, 173880924 misses, 9516744 dirtied
  avg read rate: 22.088 MB/s, avg write rate: 1.209 MB/s
  system usage: CPU: user: 2758.07 s, system: 2991.62 s, elapsed: 61501.88 s
```

= 17 hours

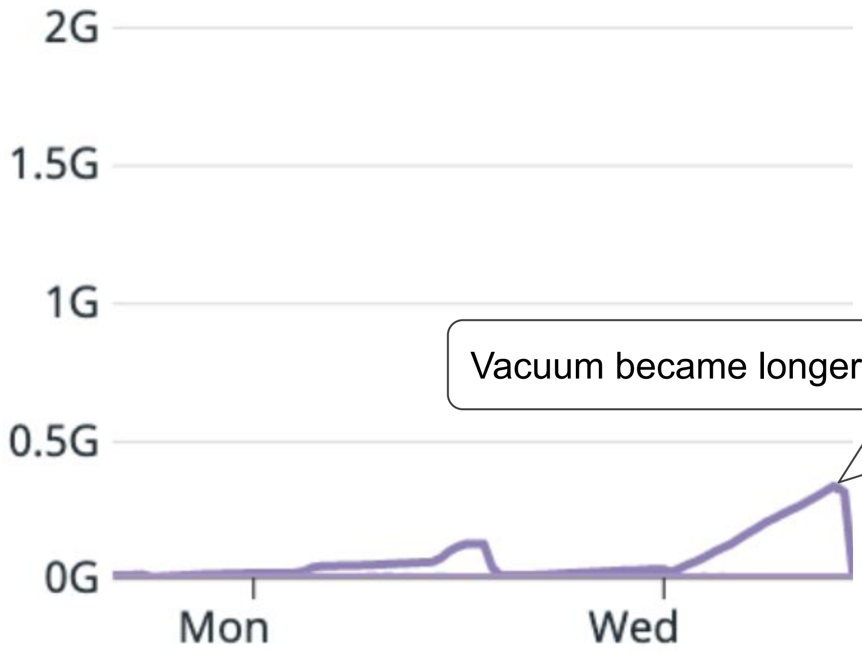
- This time



```
[redacted] 12:57:47 UTC::@:[42289]:LOG:  automatic aggressive vacuum of table "plazma_production.public.partitions": index scans: 4
  pages: 0 removed, 139127384 remain, 2 skipped due to pins, 61849366 skipped frozen
  tuples: 186520328 removed, 6525250411 remain, 241062325 are dead but not yet removable, oldest xmin: 706898483
  buffer usage: 126694379 hits, 731164702 misses, 59408422 dirtied
  avg read rate: 20.066 MB/s, avg write rate: 1.630 MB/s
  system usage: CPU: user: 14276.74 s, system: 14473.80 s, elapsed: 284667.00 s
```

= 79 hours = 4.6 days

Tons of Dead Rows

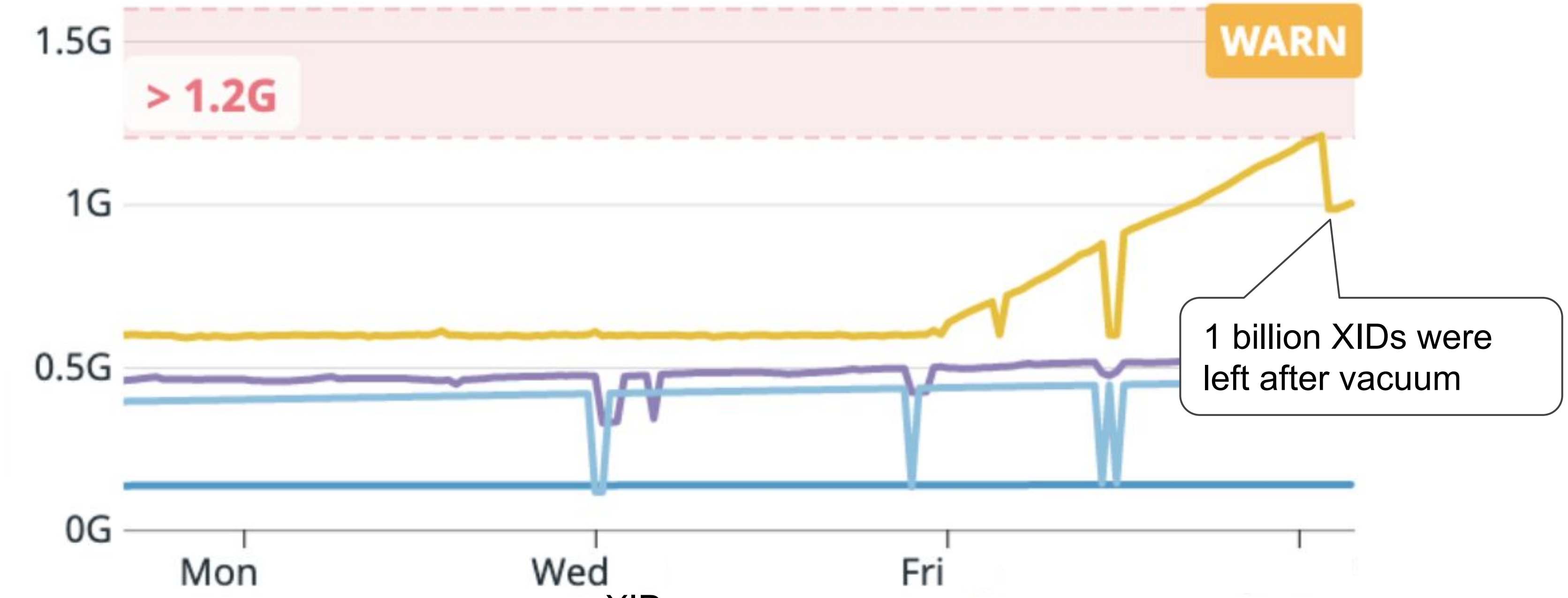


Vacuum became longer to cleanup them

Approximated number of dead rows

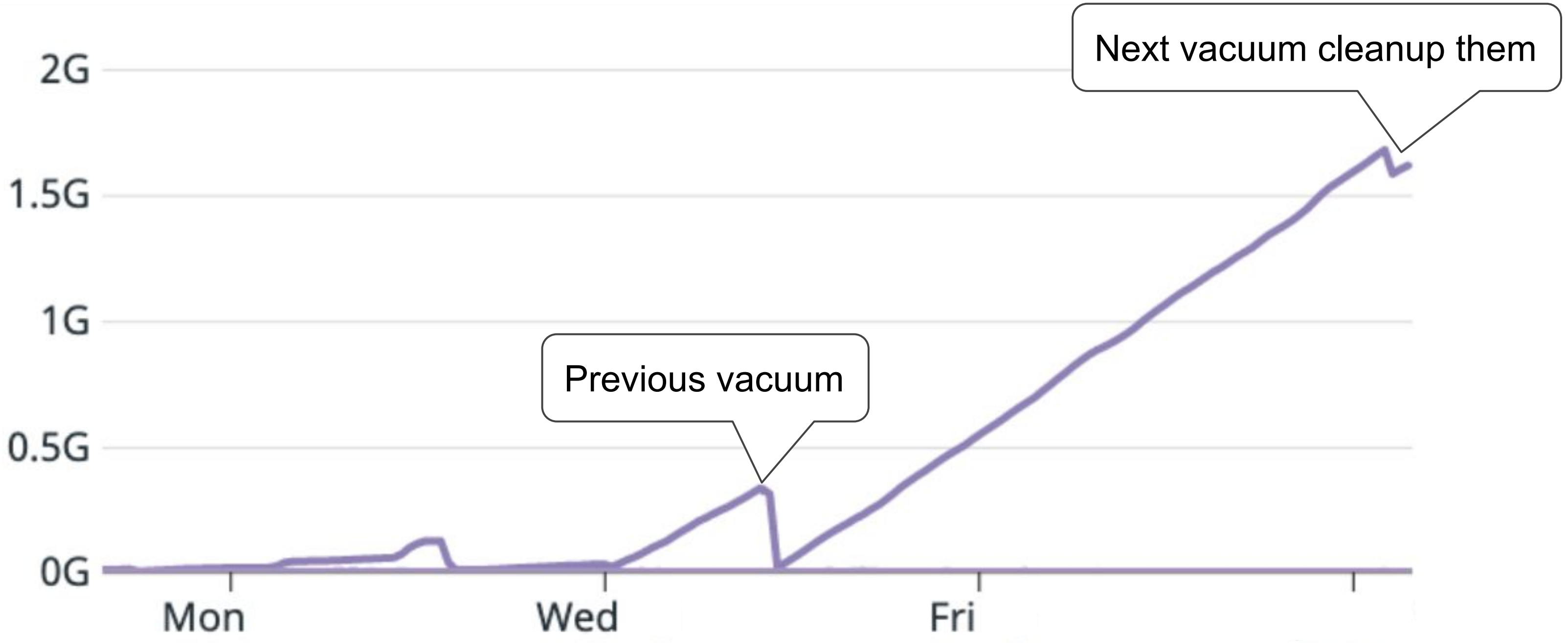
High XID age even after Vacuum

XID age was over 1 billion even after vacuum completion
-> More dead rows were added while vacuum ran



XID age

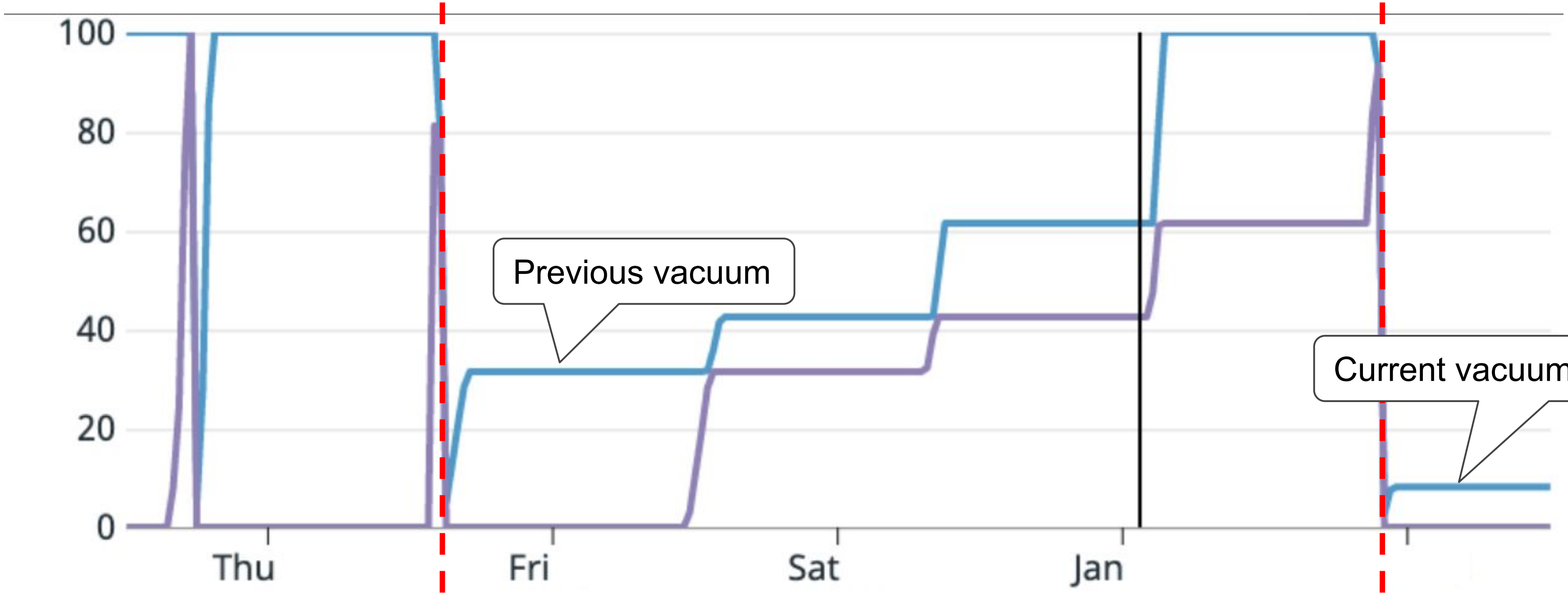
Much More Dead Rows…



Approximated number of dead rows

Vacuum Got Slower

- Next vacuum was even slower than previous one



Progress of heap block scan / vacuum (%)

Countdown for Meltdown

- We realized vacuum wouldn’t complete before running out XIDs
  - I escalated EMERGENCY this timing
  - Estimated time to complete was 9.6 days
    (from vacuum progress of first some hours)
  - Expected time to run out XIDs was 2.8 days

Plazma MetaDB would meltdown in 2.8 days if no action was taken

What would you do?

Actions after Detection

- Escalation to team
  - All team members started working on remediation
    - Kept working between JST and PST
  - Created document to gather information and record decisions

- Consider remediation ideas
  - Can we delay XID run out?
  - How can we get XID age back to normal before running out?

- Preparation for worst case scenario
  - We need to stop most of business and focus on recovery
    - Started informing the risk for internal stakeholders
  - We put decision point on XID age = 1.8 billion

Can We Delay XID Run Out?

- Ideas to reduce XIDs used by applications of Plazma
  - Stop internal unimportant workloads
    - Stopped GC and optimizer workers of Plazma temporarily
  - Reduce # of partitions ingested into Plazma
    - Roughly 1 TX is used to ingest 1 partition metadata
    - Reduced # of partitions by aggregating more data into a partition
      - Extend buffering time
      - Modify granularity of partition
  - Stop workload that consumes many XIDs
    - Not implemented soon and kept as option because significant impact is expected for customers

Remediation Ideas so far

- Delay XID run out
  - ✅ Stop internal unimportant workloads
  - ✅ Reduce # of partitions to ingest into Plazma
  - [pending] Stop workload that consumes many XIDs
- Next question: How can we recover XID age before running out?

Try to accelerate vacuum by config change / scale up

- Update configuration to run vacuum more aggressively
  - Conclusion : no improvement expected by config change
    - Working memory size of vacuum was already max (maintenance_work_mem = 1GB)

    - Autovacuum ran without throttling (autovacuum_vacuum_cost_delay = 0)

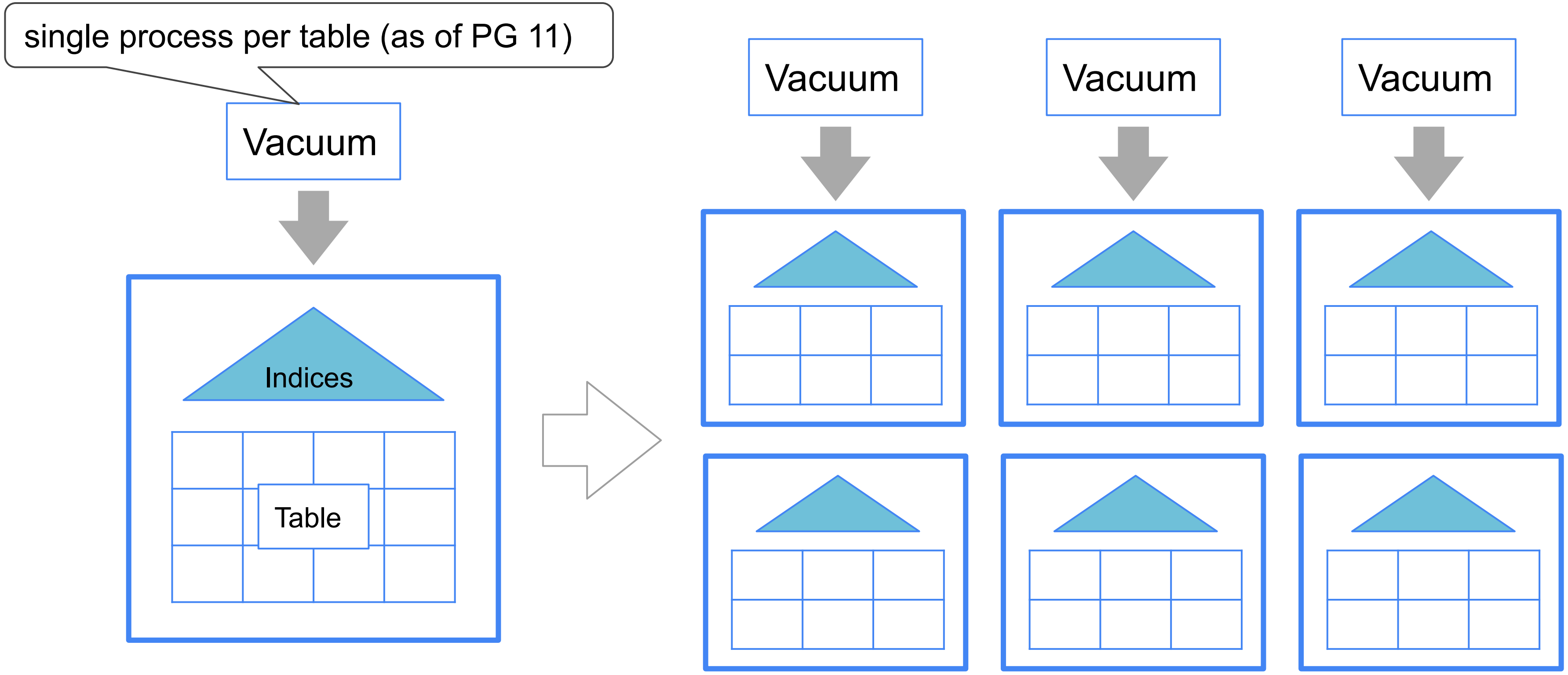
    - Vacuum on a table cannot be parallelized (in PG 11)

- Scale up DB server if any server resource was bottleneck
  - Conclusion : no improvement expected by scale up
    - 3 autovacuum workers were running without saturation of CPU / IO
      - Resources could not be fully utilized by lack of parallelism

Remediation Ideas

- Delay XID run out
  - ✅ Stop internal unimportant workloads
  - ✅ Reduce # of partitions to ingest into Plazma
  - [pending] Stop workload that consumes many XIDs
- Recover XID age before running out
  - ❌ Update configuration to run vacuum more aggressively
  - ❌ Scale up DB server if any server resource was bottleneck
- Any others?

Parallelize Vacuum by Table Partitioning



Challenges of Table Partitioning

- We had already discussed table partitioning idea but not implemented yet
    - Performance issue when many partitions were used
        - Query planning time increases in proportion to # of partitions
        - It was significantly improved by PG 12

- Also, migration of 1.1TB data to new partitioned table was challenge

It was very promising, but too challenging in limited time

Vacuum with Skip Index Cleanup

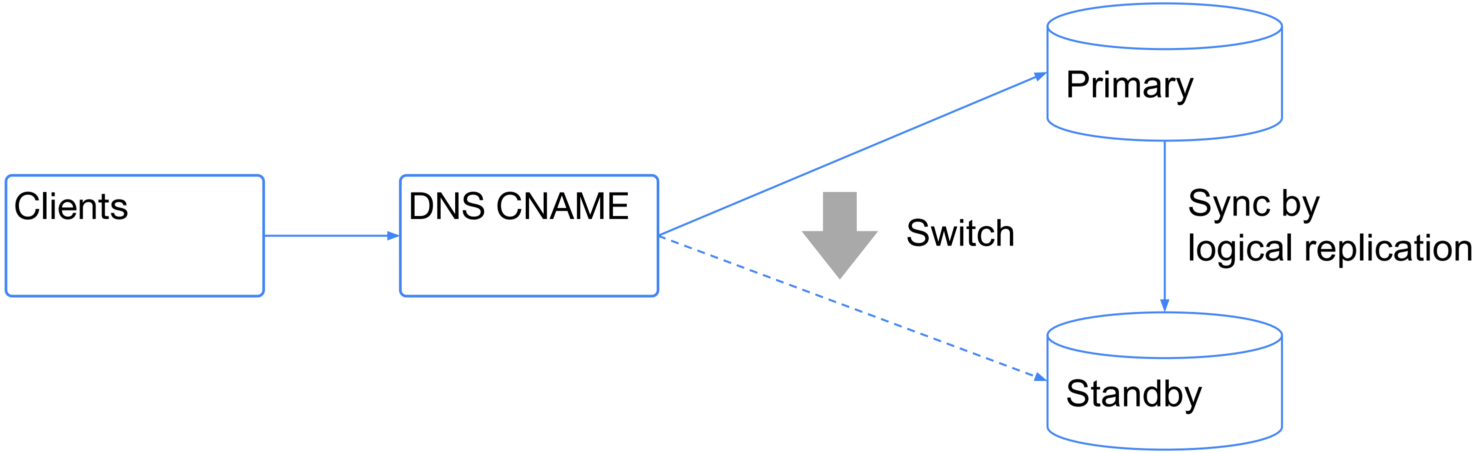
- Option to skip index cleanup step of vacuum (introduced by PG 12)
  - > Use of this option reduces the ability to reclaim space and can lead to index bloat, but it is helpful when the main goal is to freeze old tuples. (from release note)

- Postgres major version upgrade was required to use the option (we used 11)
  - Fortunately, we've already completed evaluation of 12
    - We carefully evaluate compatibility and performance of new version since upgrade isn't revertible
  - In other production regions, we've already used 12

We started test of vacuum with skip index cleanup option
-> Restored DB from snapshot, upgraded to 12 and ran vacuum

Migrate DB by Logical Replication

- Copy current DB by logical replication and migrate
  - XID is reset on the replica DB
    - Physical data layout isn't synchronized by logical replication
  - Dead rows can also be cleaned
    - Logical replication copies only alive rows
    - -> We can skip cleanup of tons of dead rows



Feasibility of Logical Replication Idea

- We've already tested logical replication for DB maintenance
  - Downtime of maintenance can be reduced compared to in-place operation
  - Actually, we planned major version upgrade to 12 by logical replication

- Challenge
  - Not sure how long it'll take to build and sync logical replica for 3.8TB DB

Started building and synchronizing logical replica DB

Remediation Ideas

- Delay XID run out
  - ✅ Stop internal unimportant workloads
  - ✅ Reduce # of partitions to ingest into Plazma
  - [pending] Stop workload that consumes many XIDs

- Recover XID age before running out
  - ❌ Update configuration to run vacuum more aggressively
  - ❌ Scale up DB server if any server resource was bottleneck
  - ❌ Partitioning the table to parallelize vacuum
  - [WIP] Upgrade PostgreSQL to 12 and run vacuum with skip index cleanup
  - [WIP] Migrate DB by logical replication to reset XID

We bet on vacuum with skip index cleanup and logical replication ideas

Timeline : Day 1 & 2

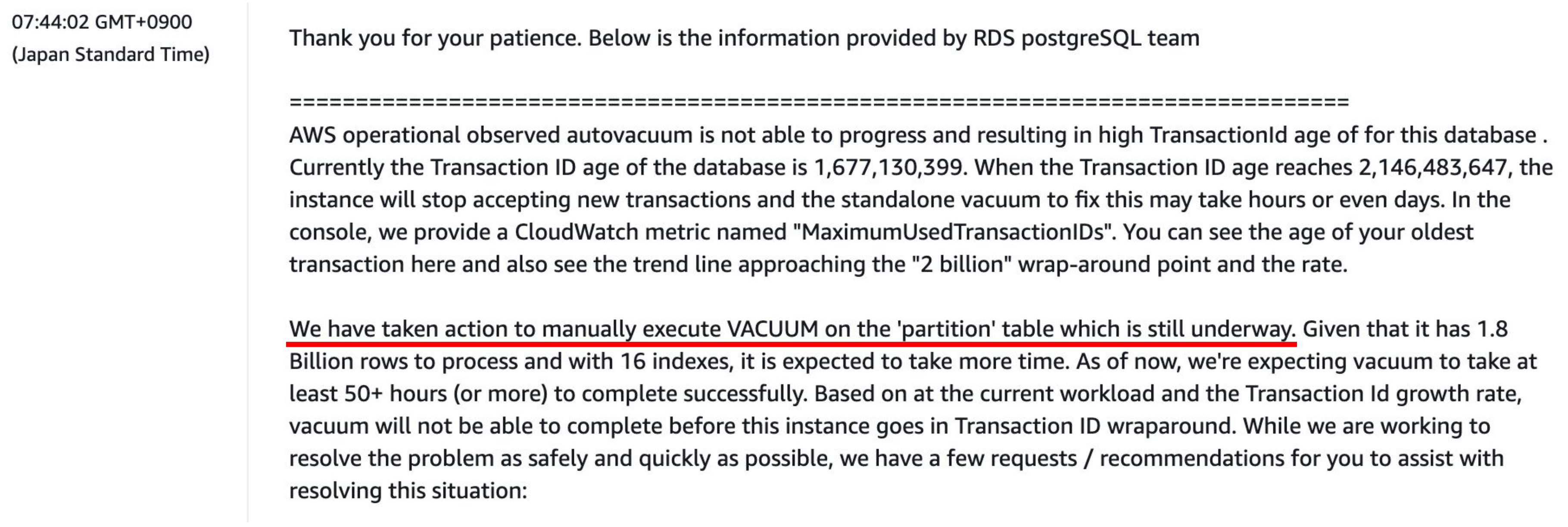
- Day 1 (JST)
  - 12:30 - Escalation (XID age = 1.15 billion)
  - 18:00 - XID age = 1.22 billion
- Day 2
  - 2:00 - Completed test of vacuum with skip index cleanup
    - Took 6.25 hours on snapshot DB
  - 12:00 - Decided to go with vacuum with skip index cleanup
    - Sync of logical replica hadn't completed -> kept replica for backup plan
    - Started rehearsal of major version upgrade maintenance on staging
  - 14:00 - DB maintenance was announced
  - 18:00 - XID age = 1.52 billion
    - XID age increased 300M in a day -> reach 1.8 billion on 17:30 Day 3

Timeline : Day 2 & 3

- Day 2 (continue from previous slide)
  - 23:00 - Completed initial data copy of logical replication
    - Still require catch up of diff after copy started

- Day 3
  - 3:00 - Noticed autovacuum worker had been killed (???)

AWS RDS Team Took Action



07:44:02 GMT+0900
(Japan Standard Time)

Thank you for your patience. Below is the information provided by RDS postgreSQL team

==

AWS operational observed autovacuum is not able to progress and resulting in high TransactionId age of for this database . Currently the Transaction ID age of the database is 1,677,130,399. When the Transaction ID age reaches 2,146,483,647, the instance will stop accepting new transactions and the standalone vacuum to fix this may take hours or even days. In the console, we provide a CloudWatch metric named "MaximumUsedTransactionIDs". You can see the age of your oldest transaction here and also see the trend line approaching the "2 billion" wrap-around point and the rate.

We have taken action to manually execute VACUUM on the 'partition' table which is still underway. Given that it has 1.8 Billion rows to process and with 16 indexes, it is expected to take more time. As of now, we're expecting vacuum to take at least 50+ hours (or more) to complete successfully. Based on at the current workload and the Transaction Id growth rate, vacuum will not be able to complete before this instance goes in Transaction ID wraparound. While we are working to resolve the problem as safely and quickly as possible, we have a few requests / recommendations for you to assist with resolving this situation:

Timeline : Day 2 & 3

- Day 2 (continue from previous slide)
  - 23:00 - Completed initial data copy of logical replication
    - Still require catch up of diff after copy started

- Day 3
  - 3:00 - Noticed autovacuum worker had been killed

  - 9:30 - Decided to wait until XID age = 1.9 billion (changed from 1.8 billion)
    - Expected time of vacuum completion : 18:00 - 19:00
    - Expected time XID age = 1.8 billion : 17:30

  - 11:00 - Started DB maintenance to upgrade to PG 12
    - Logical replication was dropped as required to run pg_upgrade

Complete Upgrade to PG 12

- In-place major version upgrade 11 -> 12
  - Upgrade by modify-db-instance of RDS (pg_upgrade is used internally)

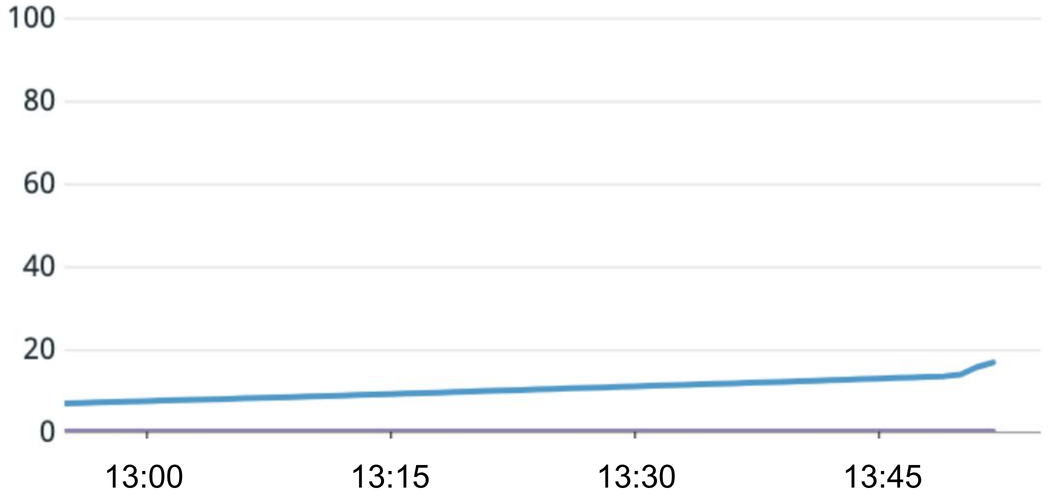
  - We've experienced the operation for several times
    - Detail: presentation of PGConf.ASIA 2018
    - But we usually start preparation at least 4 weeks before maintenance
      -> we had only 2 days this time

  - Downtime : 24 minutes

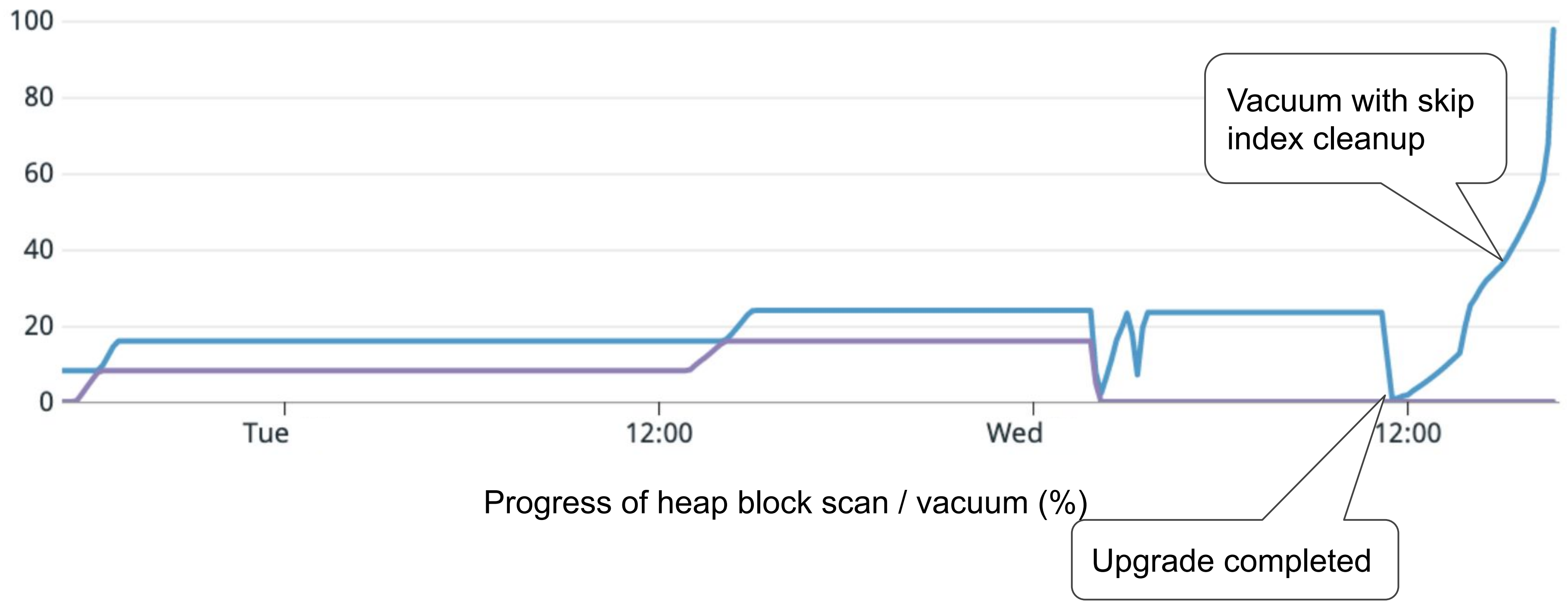
Started Vacuum with Skip Index Cleanup

- Day 3
  - 12:00 - Upgrade completed & started vacuum with skip index cleanup
    - Also, setup new logical replication for backup plan

  - 14:00 - vacuum progress ~ 20%

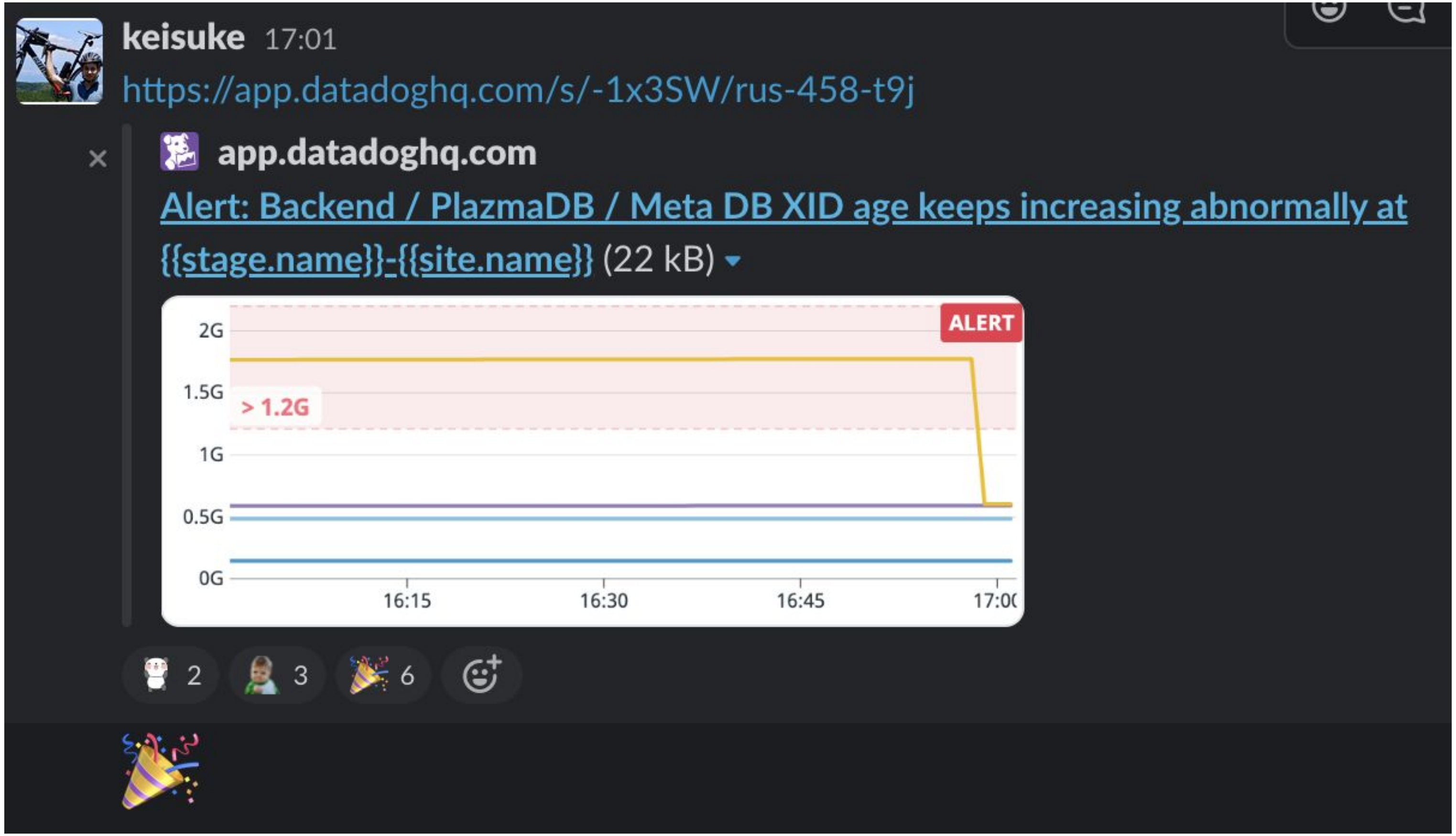


Skip Index Cleanup Did Great Job!



Progress of heap block scan / vacuum (%)

We beated it!



keisuke 17:01

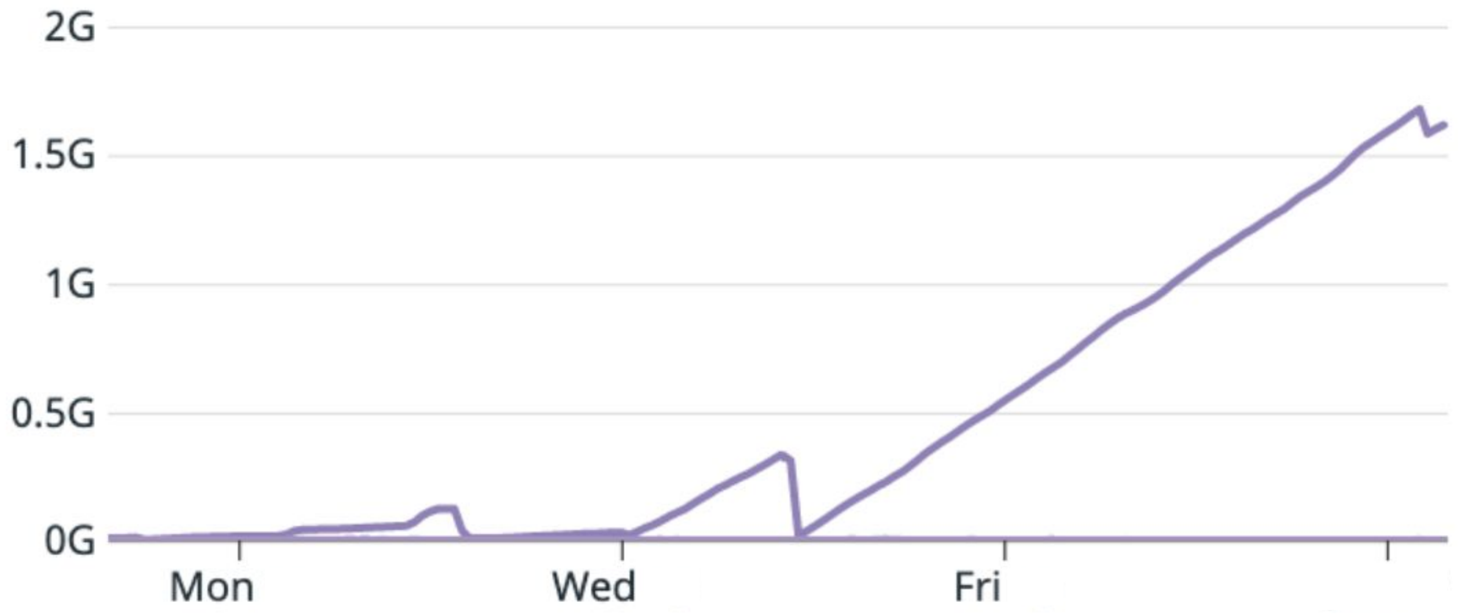
https://app.datadoghq.com/s/-1x3SW/rus-458-t9j

app.datadoghq.com

Alert: Backend / PlazmaDB / Meta DB XID age keeps increasing abnormally at {{stage.name}}-{{site.name}} (22 kB)

Why did it happen?

Vacuum took long time due to Tons of Dead Rows



Approximated number of dead rows

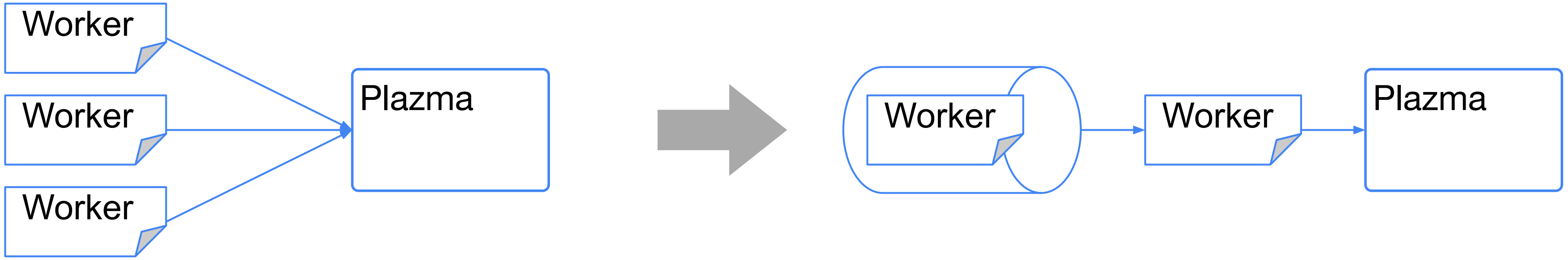
Source of Dead Rows

- 1 weeks ago, we noticed GC worker hadn’t been working
  - Worker to cleanup partition files of deleted data sets
    - When table is deleted in TD, it is just soft deletion
    - Data set associated to table is deleted after retention period
    - # of deleted partition files = # of new dead rows of PostgreSQL
  - It kept failing by timeout to delete large data sets
    -> Resumed with extending timeout
- Worker hadn’t worked for last 1 week
  - Accumulated data sets were deleted
  - It had 10ms sleep per data set but still too aggressive
    - If deleted data set has many partitions, many rows will be dead

Follow up : Improve Background Workers

- Eliminate hotspot caused by background workers
  - Introduced throttling for # of deleted partitions
    - Spread out # of dead rows even if a data set has many partitions
    - Safer backlog handling in case of failure of batch

  - Scheduling for background workers to avoid overloading
    - Plazma has various workers and they were scheduled independently -> caused hotspot when they happened to run concurrently



Follow up : Cleanup of Dead Rows

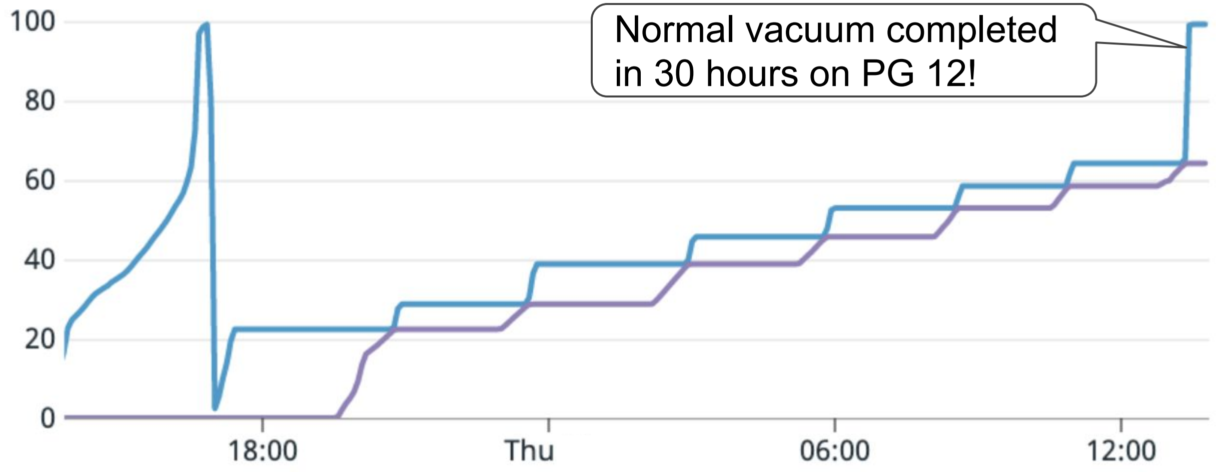
- Dead rows aren’t cleaned up by vacuum with skip index cleanup
    - Need normal vacuum

- On PG 11, estimation of normal vacuum was 9.6 days
    - If normal vacuum didn’t complete before XID run out, we need to keep using vacuum with skip index cleanup

Follow up : Cleanup of Dead Rows

- Dead rows aren't cleaned up by vacuum with skip index cleanup
  - Need normal vacuum

- On PG 11, estimation of normal vacuum was 9.6 days
  - If normal vacuum didn't complete before XID run out, we need to keep using vacuum with skip index cleanup

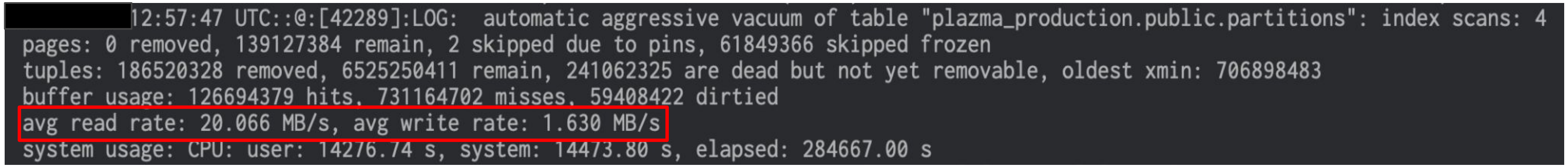


Performance Improvement of GiST Index Vacuum

> Improve the performance of vacuum scans of GiST indexes
(From [PG 12 release note]())

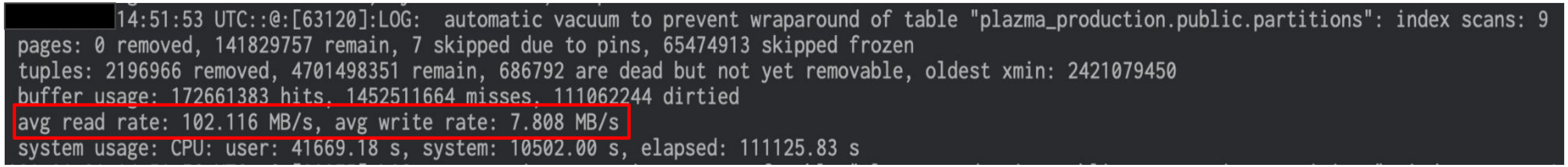
Thank you PostgreSQL community!

- Stats on PG 11



```
[redacted]12:57:47 UTC::@:[42289]:LOG:  automatic aggressive vacuum of table "plazma_production.public.partitions": index scans: 4
 pages: 0 removed, 139127384 remain, 2 skipped due to pins, 61849366 skipped frozen
 tuples: 186520328 removed, 6525250411 remain, 241062325 are dead but not yet removable, oldest xmin: 706898483
 buffer usage: 126694379 hits, 731164702 misses, 59408422 dirtied
 avg read rate: 20.066 MB/s, avg write rate: 1.630 MB/s
 system usage: CPU: user: 14276.74 s, system: 14473.80 s, elapsed: 284667.00 s
```

- Stats on PG 12

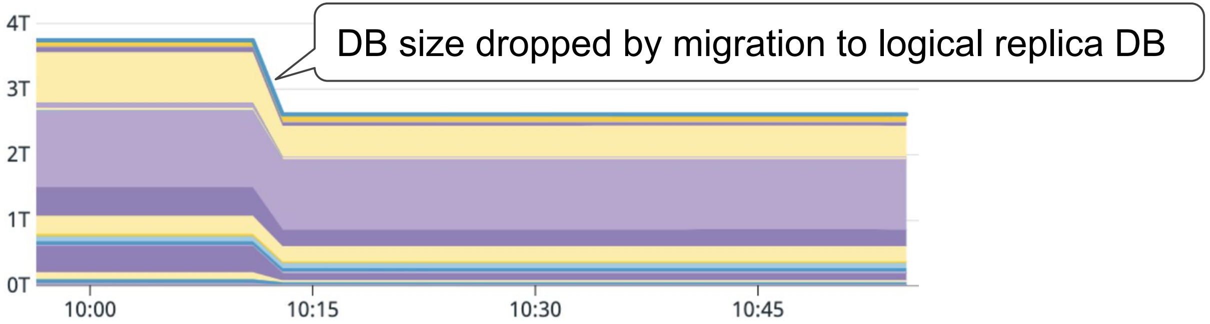


```
[redacted]14:51:53 UTC::@:[63120]:LOG:  automatic vacuum to prevent wraparound of table "plazma_production.public.partitions": index scans: 9
 pages: 0 removed, 141829757 remain, 7 skipped due to pins, 65474913 skipped frozen
 tuples: 2196966 removed, 4701498351 remain, 686792 are dead but not yet removable, oldest xmin: 2421079450
 buffer usage: 172661383 hits, 1452511664 misses, 111062244 dirtied
 avg read rate: 102.116 MB/s, avg write rate: 7.808 MB/s
 system usage: CPU: user: 41669.18 s, system: 10502.00 s, elapsed: 111125.83 s
```

Follow up : DB migration with Logical Replication

- Established maintenance operation with logical replication + DB migration
  - DB downtime of major version upgrade shorter than in-place upgrade e.g.
    - In-place (RDS modify-db-instance) : 23 min
    - Logical replication + DB migration : 106 sec

  - It can also be used like VACUUM FULL

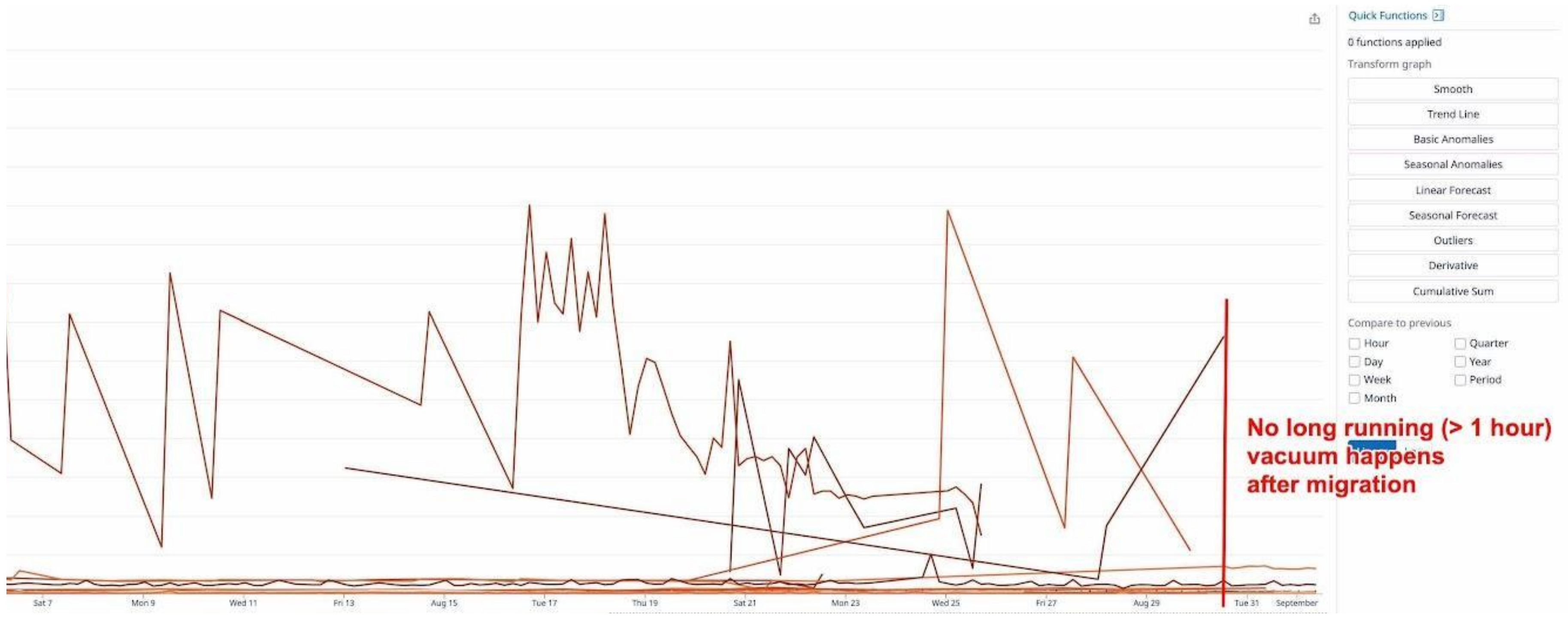


  - It took 52 hours to build and sync logical replica of MetaDB

Follow up : Table Partitioning

- Split the largest table (1.9TB incl. indices) into 256 partitioned table
  - Transparent data migration
    - Plazma Metadata abstraction layer read both old and new tables during migration
    - Data migration worker repeated delete chunk of rows from old table and insert into partitioned table
      - Worker also ran on background worker scheduler to avoid impact for customer workloads
    - Took 9 days to migrate all data

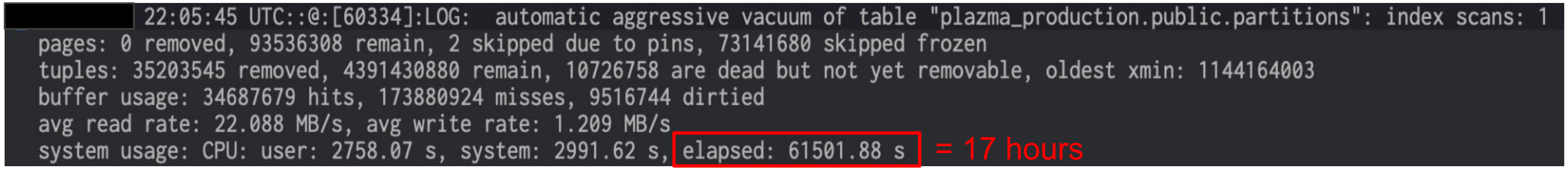
Vacuum Execution Time after Partitioning



No long running (> 1 hour) vacuum happens after migration

Vacuum Execution Time after Partitioning

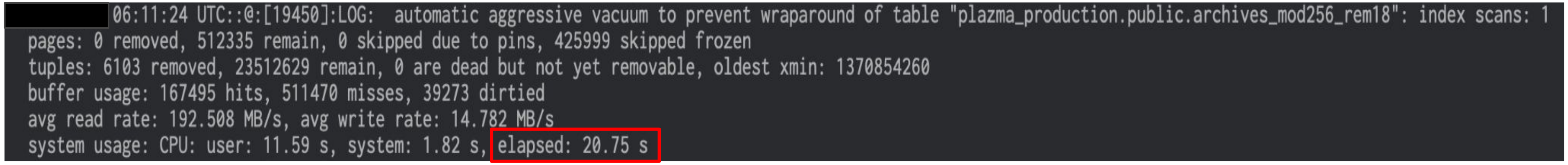
- PG 11, before partitioning



```
[redacted] 22:05:45 UTC::@:[60334]:LOG:  automatic aggressive vacuum of table "plazma_production.public.partitions": index scans: 1
  pages: 0 removed, 93536308 remain, 2 skipped due to pins, 73141680 skipped frozen
  tuples: 35203545 removed, 4391430880 remain, 10726758 are dead but not yet removable, oldest xmin: 1144164003
  buffer usage: 34687679 hits, 173880924 misses, 9516744 dirtied
  avg read rate: 22.088 MB/s, avg write rate: 1.209 MB/s
  system usage: CPU: user: 2758.07 s, system: 2991.62 s, elapsed: 61501.88 s
```

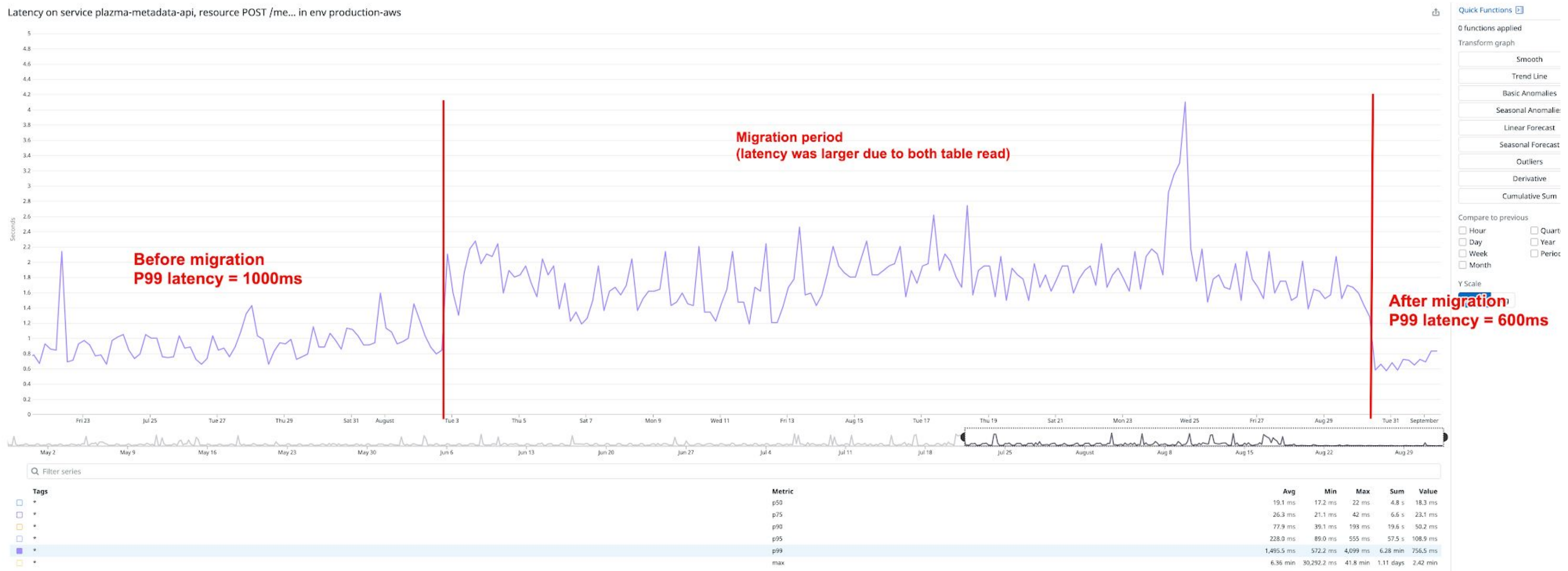
= 17 hours

- PG 12, after partitioning (vacuum of 1 partition table)



```
[redacted] 06:11:24 UTC::@:[19450]:LOG:  automatic aggressive vacuum to prevent wraparound of table "plazma_production.public.archives_mod256_rem18": index scans: 1
  pages: 0 removed, 512335 remain, 0 skipped due to pins, 425999 skipped frozen
  tuples: 6103 removed, 23512629 remain, 0 are dead but not yet removable, oldest xmin: 1370854260
  buffer usage: 167495 hits, 511470 misses, 39273 dirtied
  avg read rate: 192.508 MB/s, avg write rate: 14.782 MB/s
  system usage: CPU: user: 11.59 s, system: 1.82 s, elapsed: 20.75 s
```

Tail Latency Improvement


Latency on service plazma-metadata-api, resource POST /me... in env production-aws

	Tags	Metric	Avg	Min	Max	Sum	Value
☐	*	p50	19.1 ms	17.2 ms	22 ms	4.8 s	18.3 ms
☐	*	p75	26.3 ms	21.1 ms	42 ms	6.6 s	23.1 ms
☐	*	p90	77.9 ms	39.1 ms	193 ms	19.6 s	50.2 ms
☐	*	p95	228.0 ms	89.0 ms	555 ms	57.5 s	108.9 ms
■	*	p99	1,495.5 ms	572.2 ms	4,099 ms	6.28 min	756.5 ms
☐	*	max	6.36 min	30,292.2 ms	41.8 min	1.11 days	2.42 min

P99 MetaDB query response time improved 40%

Retro

- Bad points 👎
  - Caused crisis of system
    - Spent huge human effort and time for remediation
    - Should have earlier alert before it became so serious

- Good points 👍
  - Avoided catastrophic crash

  - We had ideas to improve system and had already worked on them
    - Upgrade to PG 12 and logical replication could be done in limited time

  - System had good observability
    - Plenty of metrics and logs for investigation and remediations
    - It would take time to recognize issue if observability wasn't enough

Summary

- We overcame crisis of Plazma MetaDB meltdown by XID wraparound
  - Accelerated row freeze by vacuum with skip index cleanup
  - Improved system stability and scalability by follow up actions
    - Throttling for background workers
    - Established DB migration operation with logical replication
    - Table partitioning
  - We enjoy evolution of PostgreSQL. Thank you PostgreSQL community.
  - Finally, vacuum time decreased 12-24 hours -> less than 60 seconds
- Workload of Plazma has been changing over time
  - We always pay attention to change of trend and keep improving system

Abstract

Treasure DataのストレージレイヤPlazmaは、2014年から運用されており、現在もTDのビジネスを支える基盤となっています。扱うデータ量は年々増大しており、安定性やスケーラビリティの担保のために常に改善を続けています。しかしながら、昨年ある重大な危機が発生しました。TDの根幹となる分析基盤を揺るがしうる危機を克服したストーリーを振り返ります。