arm
TREASURE DATA

Handling Performance Issues of Production Systems

Oct. 8th, 2020

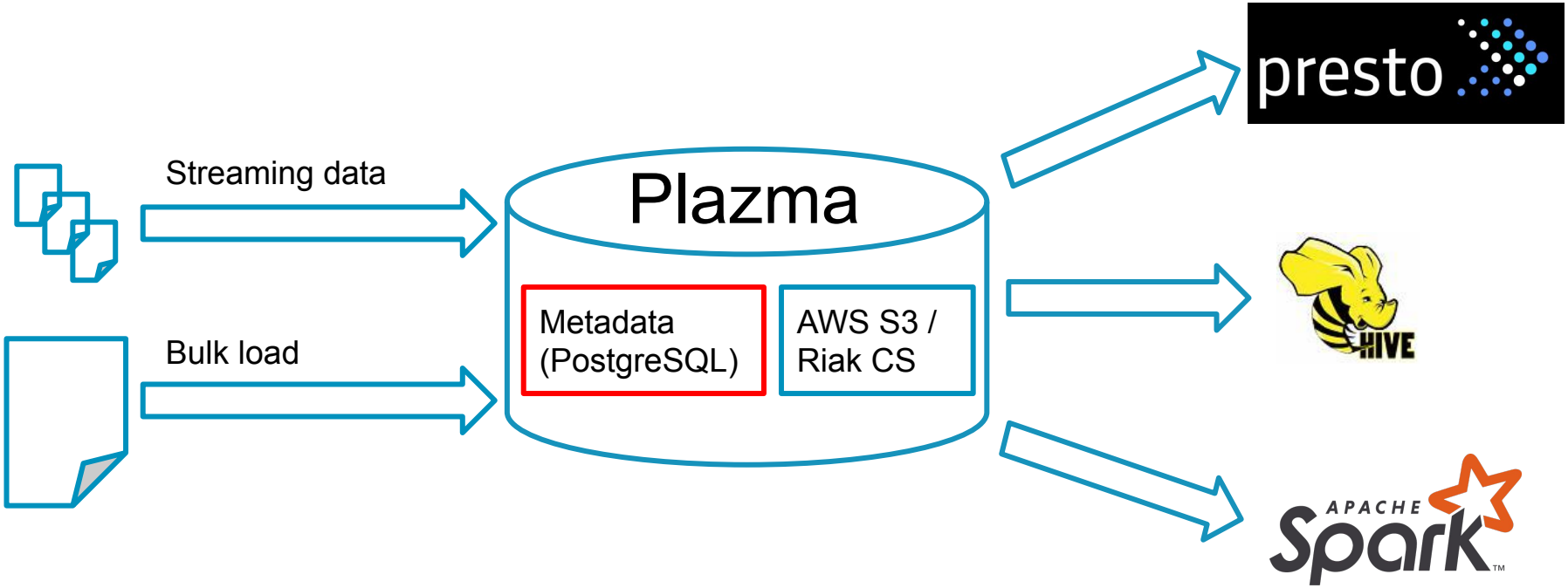
Keisuke Suzuki

Who am I?

Keisuke Suzuki

- Backend Engineer @ Treasure Data KK
  - Plazma: Distributed storage
- Personal interests
  - DB / Distributed systems / Performance optimization

Plazma


Streaming data
Bulk load
Plazma
Metadata
(PostgreSQL)
AWS S3 /
Riak CS
presto
HIVE
APACHE
Spark

Daily Workload and Storage Size of Metadata

Write workload

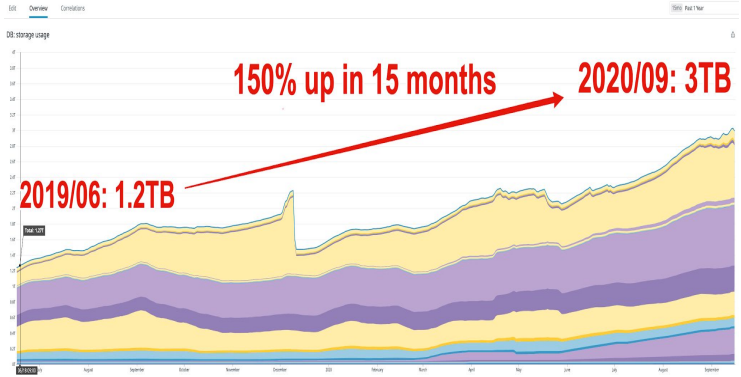
- Transaction
  - Avg : 1.2k TPS
  - Max : 4k TPS
- IOPS
  - Avg : 1.7k IOPS
  - Max : 50k IOPS

Read workload

- Queries
  - Avg : 20 queries/s
    38k records/s
  - Max : 100 queries/s
    450k records/s
- IOPS
  - Avg : 7.7k IOPS
  - Max : 35k IOPS

Storage size

- Current Size
  - Total of tables : 3TB
  - Biggest table : 800GB



arm
TREASURE DATA

Handling Performance Issues

Common Performance Issue Examples

- Load average is too high due to traffic spikes
- Excessive disk IOs due to poor cache hit ratio
- Daily batch is single threaded but data size is too large

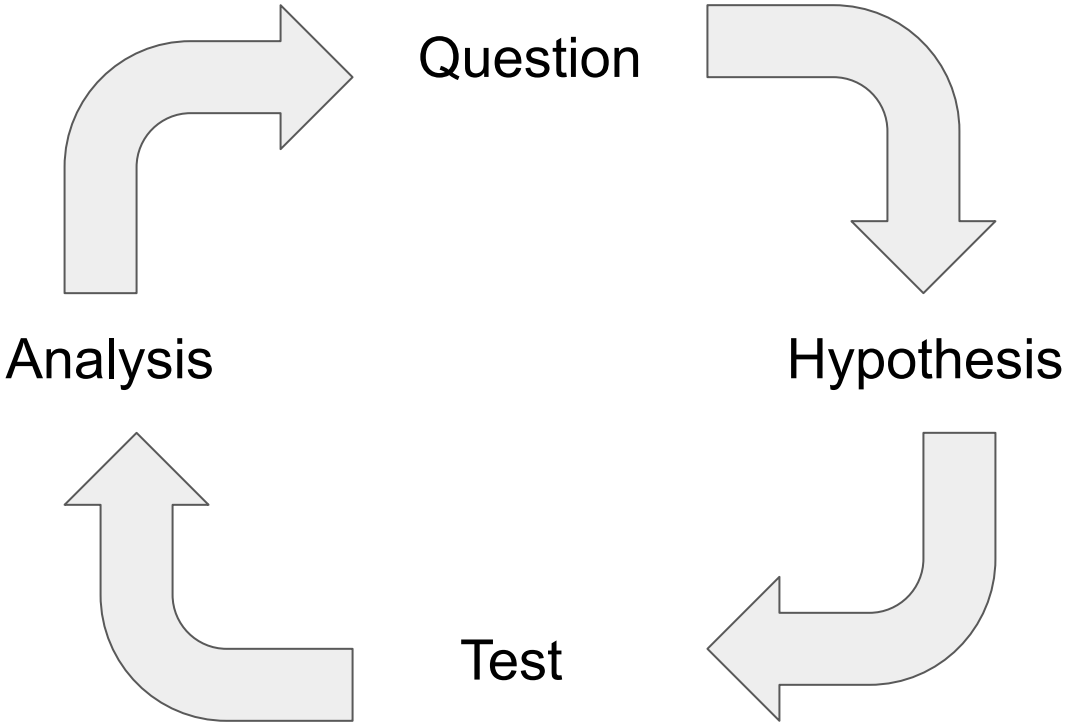
Typical Performance Issue Handling Process

1. Problem statement
   - Clarify what is happening now. e.g.
     - What makes you think there is a performance issue?
     - What environment does it happen?
     - Who are affected?
     - When did it start?
     - How frequently does it happen?

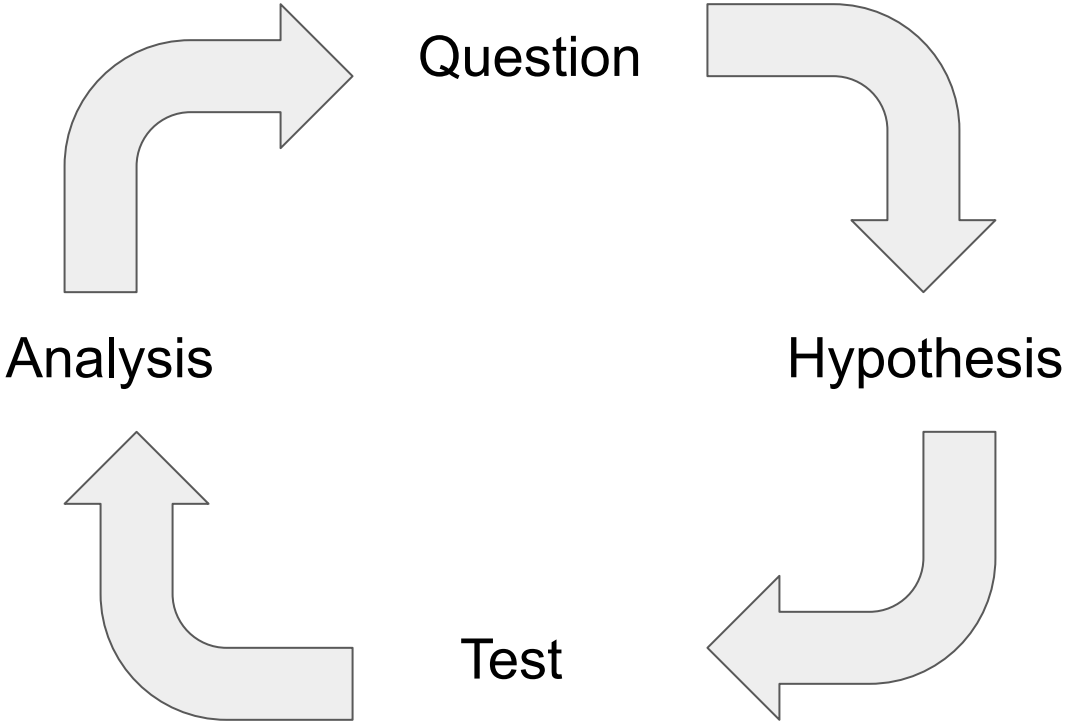
2. Check known issues
   - Matching conditions, metrics and logs with patterns of previous issues

3. Diagnose unknowns in detail
   - Narrow down suspicious points by scientific approach

Diagnosis Cycle

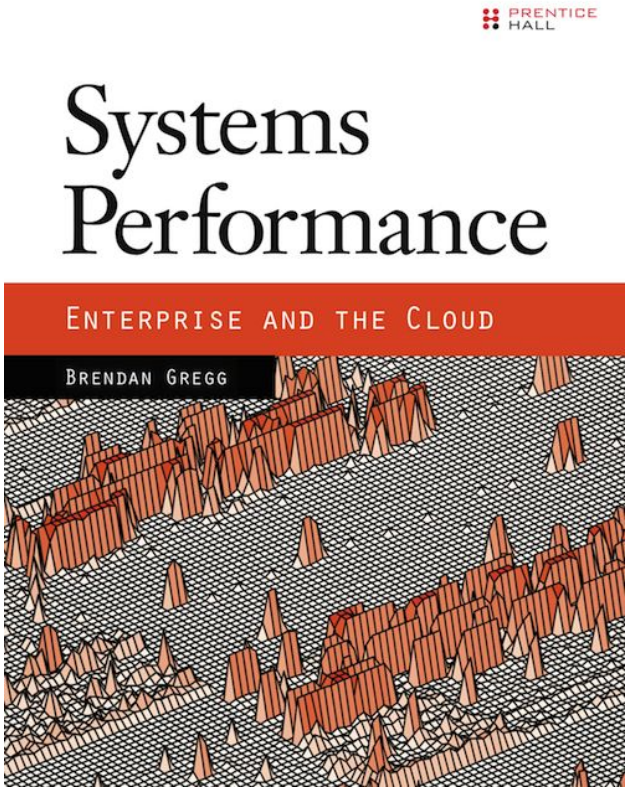


Diagnosis Cycle



How we can minimize iteration to find a root cause?

Reference Book



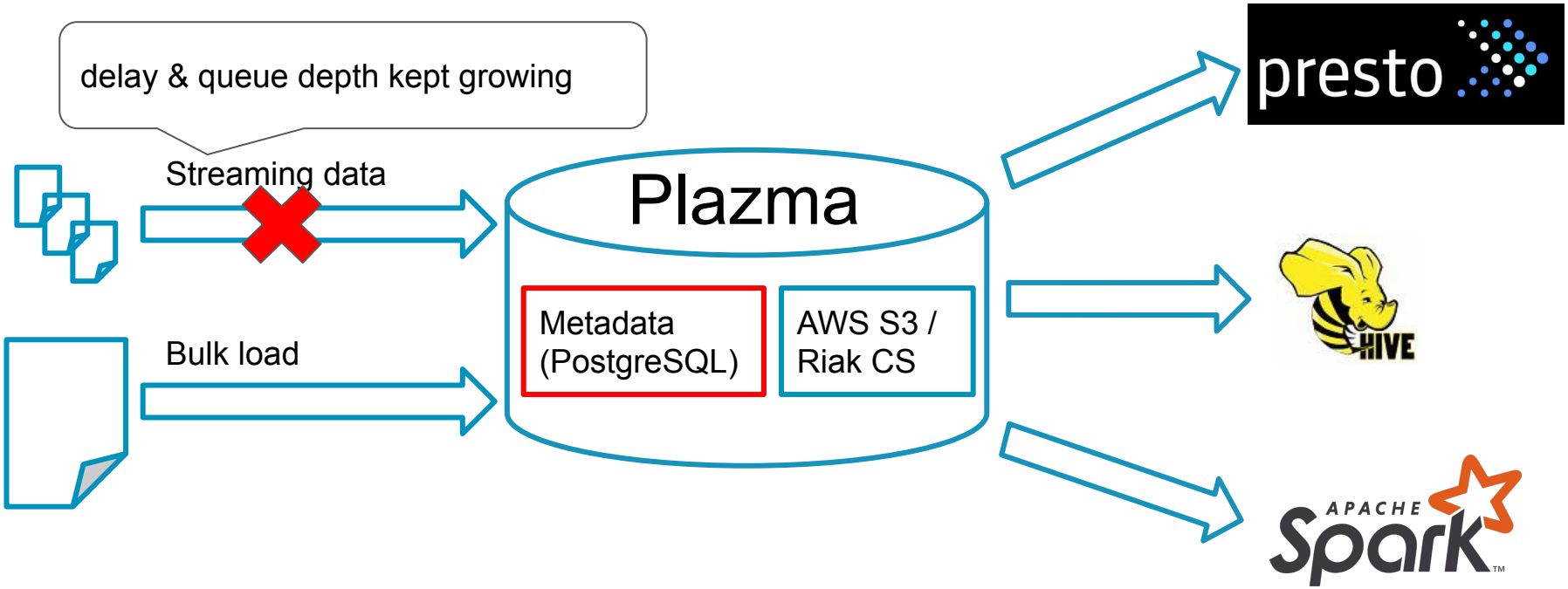
Systems Performance: Enterprise and the Cloud

- Concepts of performance
- Methodology
- Benchmarking
- Case Study

arm
TREASURE DATA

Case Study of TD

Case : Sudden DB Performance Degradation

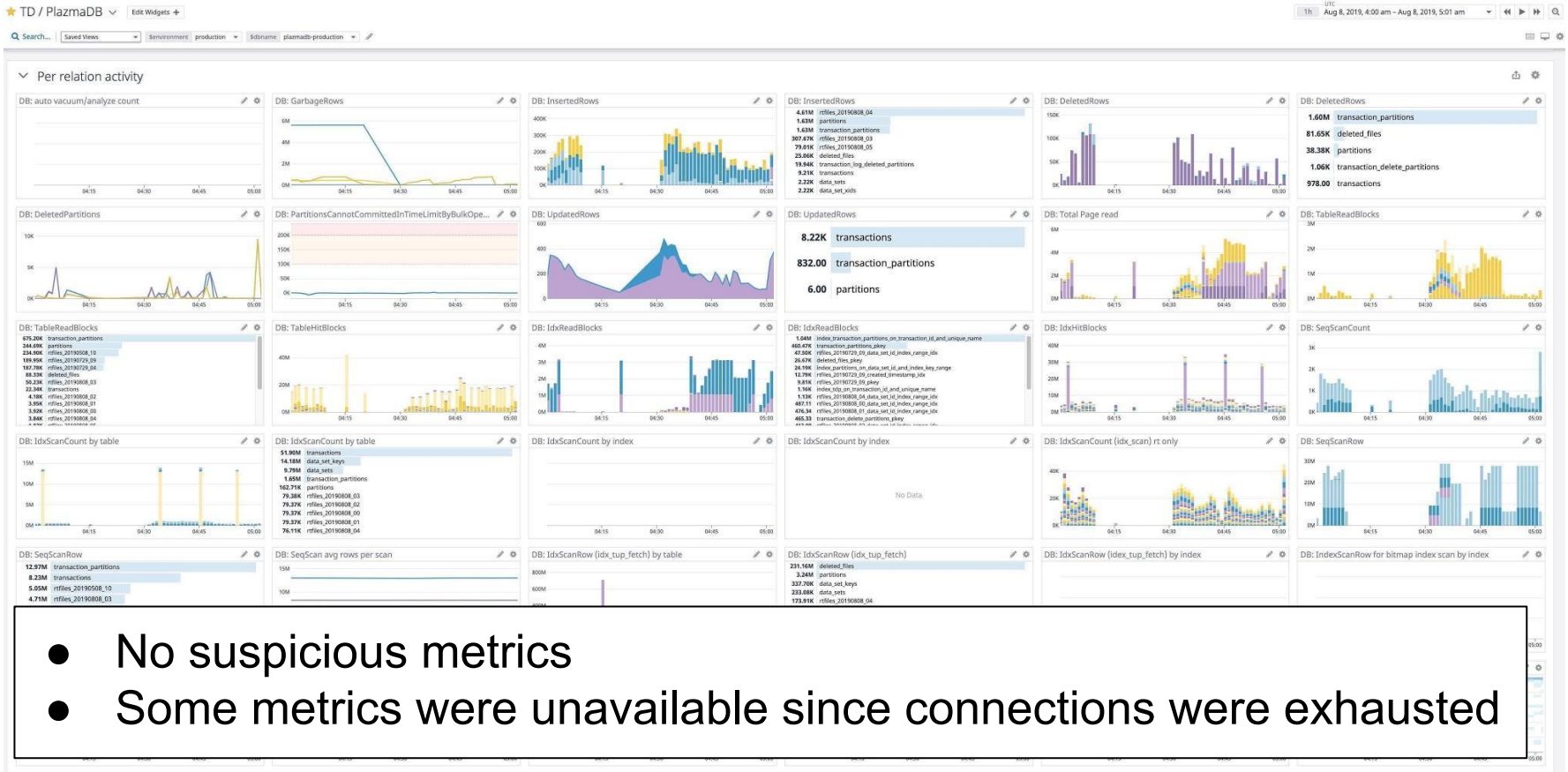


CPU / Disk IO / Network IO Metrics

Timeline
04:10 Start degradation
04:28 Start reboot with failover
04:30 Complete reboot

- CPU Utilization: Drop
- Disk IO: Drop
- Database Connections: Up

PostgreSQL Metrics



- No suspicious metrics
- Some metrics were unavailable since connections were exhausted

First Analysis and Hypothesis

- Facts
  - No recent release & configuration change
  - No abnormal traffic was observed on the metrics
  - The symptom disappeared after failover reboot

- Hypothesis
  - RDS DB instance had a disk issue -> FALSE
    - Asked AWS support

Second Occurrence

- New Facts
  - The symptom happened again 1 week after the first occurence
    - The symptom is not related to some specific hardware?
  - Failover reboot solved the problem again
- What could be a cause?

USE Method

1. Create list of system resources
   - List should be exhaustive as much as possible
     - Hardware resource : e.g.) CPU cores, DRAM, Disks
     - Software resource : e.g.) Queues, locks, thread pool

2. Check Utilization, Saturation and Errors for all resources
   - Errors : number of errors reported
   - Utilization : percent that a resource was busy over a time interval
   - Saturation : wait queue depth

Major Resources of RDBMS

- Hardware resources
  - CPU cores
  - Memory
  - Disks
  - Network

- Software resources
  - Connection limit
  - Locks

Major Resources of RDBMS

- Hardware resources
  - CPU cores
  - Memory
  - Disks
  - Network

  } Low Utilization / No errors

- Software resources
  - Connection limit -> High Utilization
  - Locks -> ?
    - We hadn’t enabled lock wait logs so enabled and waited reproduction

Utilization / Saturation of Locks

```
2019-08-10 21:01:26 UTC:...:[27473]:LOG:  process 27473 still waiting for
ExclusiveLock on extension of relation 2357607 of database 16414 after 1000.270 ms

2019-08-10 21:01:26 UTC:...:[27473]:DETAIL:  Process holding the lock: 27504. Wait
queue: 27473, 23705, 27327, 27433, 27299, 27353, 27591, 27337, 27548, 27431, ...

2019-08-10 21:01:26 UTC:...:[27473]:STATEMENT:  INSERT INTO rtfiles_20190810_21
(data_set_id, created_timestamp, first_index_key, last_index_key, file_size,
record_count, path) VALUES ($1, $2, $3, $4, $5, $6, $7)
```

Not acquired by applications but by PostgreSQL internal processing

EXCLUSIVE Table Lock

Table 13.2. Conflicting Lock Modes

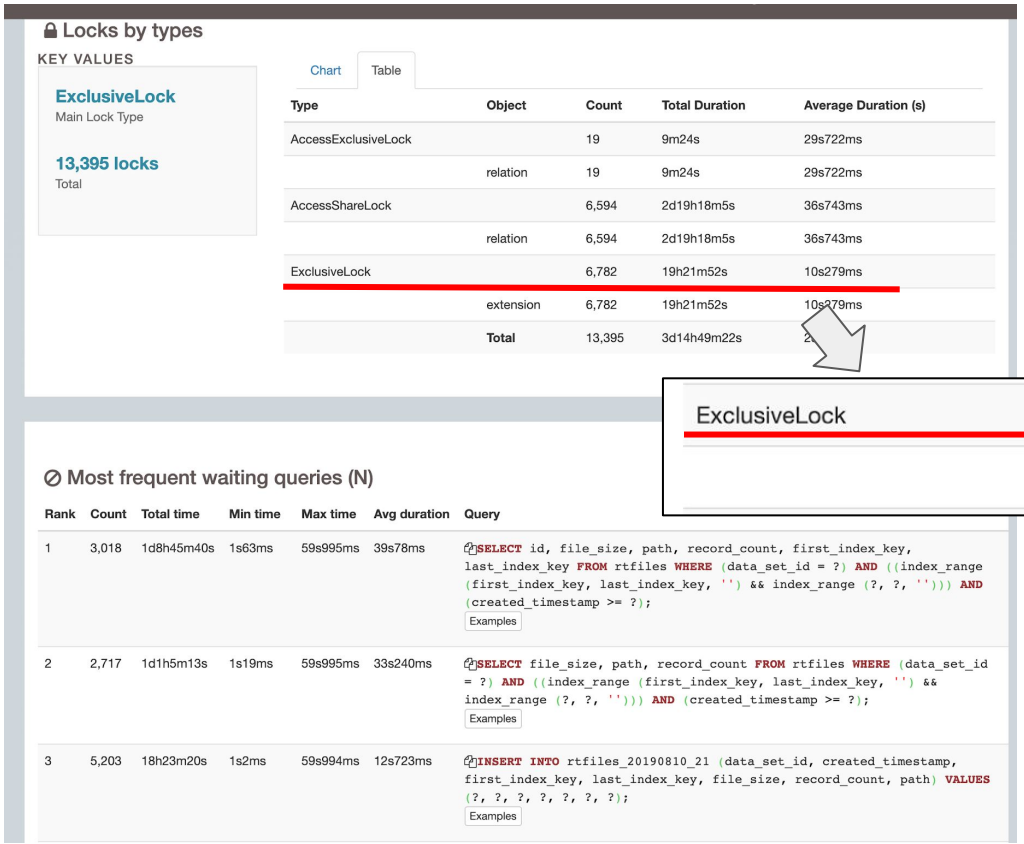
Requested Lock Mode	Current Lock Mode							
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

https://www.postgresql.org/docs/11/explicit-locking.html

EXCLUSIVE lock conflicts except ACCESS SHARE lock

- INSERT / UPDATE / DELETE -> take ROW SHARE -> NG
- SELECT -> take ACCESS SHARE -> OK

Many EXCLUSIVE Locks Usage



Locks by types

KEY VALUES

ExclusiveLock
Main Lock Type

13,395 locks
Total

Chart | Table

Type	Object	Count	Total Duration	Average Duration (s)
AccessExclusiveLock		19	9m24s	29s722ms
	relation	19	9m24s	29s722ms
AccessShareLock		6,594	2d19h18m5s	36s743ms
	relation	6,594	2d19h18m5s	36s743ms
ExclusiveLock		6,782	19h21m52s	10s279ms
	extension	6,782	19h21m52s	10s279ms
	Total	13,395	3d14h49m22s	2

Most frequent waiting queries (N)

Rank	Count	Total time	Min time	Max time	Avg duration	Query
1	3,018	1d8h45m40s	1s63ms	59s995ms	39s78ms	SELECT id, file_size, path, record_count, first_index_key, last_index_key FROM rtfiles WHERE (data_set_id = ?) AND ((index_range (first_index_key, last_index_key, '') && index_range (?, ?, ''))) AND (created_timestamp >= ?); Examples
2	2,717	1d1h5m13s	1s19ms	59s995ms	33s240ms	SELECT file_size, path, record_count FROM rtfiles WHERE (data_set_id = ?) AND ((index_range (first_index_key, last_index_key, '') && index_range (?, ?, ''))) AND (created_timestamp >= ?); Examples
3	5,203	18h23m20s	1s2ms	59s994ms	12s723ms	INSERT INTO rtfiles_20190810_21 (data_set_id, created_timestamp, first_index_key, last_index_key, file_size, record_count, path) VALUES (?, ?, ?, ?, ?, ?, ?); Examples

ExclusiveLock		6,782	19h21m52s	10s279ms
	extension	6,782	19h21m52s	10s279ms

Similar Issue Reported in the Community...

Currently bigger shared_buffers settings don't combine well with relations being extended frequently. Especially if many/most pages have a high usagecount and/or are dirty and the system is IO constrained.

shared_buffers = 144GB in our env

As a quick recap, relation extension basically works like:
1) We lock the relation for extension
2) ReadBuffer*(P_NEW) is being called, to extend the relation
3) smgrnblocks() is used to find the new target block
4) We search for a victim buffer (via BufferAlloc()) to put the new block into
5) If dirty the victim buffer is cleaned
6) The relation is extended using smgrextend()
7) The page is initialized

The problems come from 4) and 5) potentially each taking a fair while. If the working set mostly fits into shared_buffers 4) can requiring iterating over all shared buffers several times to find a victim buffer. If the IO subsystem is buys and/or we've hit the kernel's dirty limits 5) can take a couple seconds.

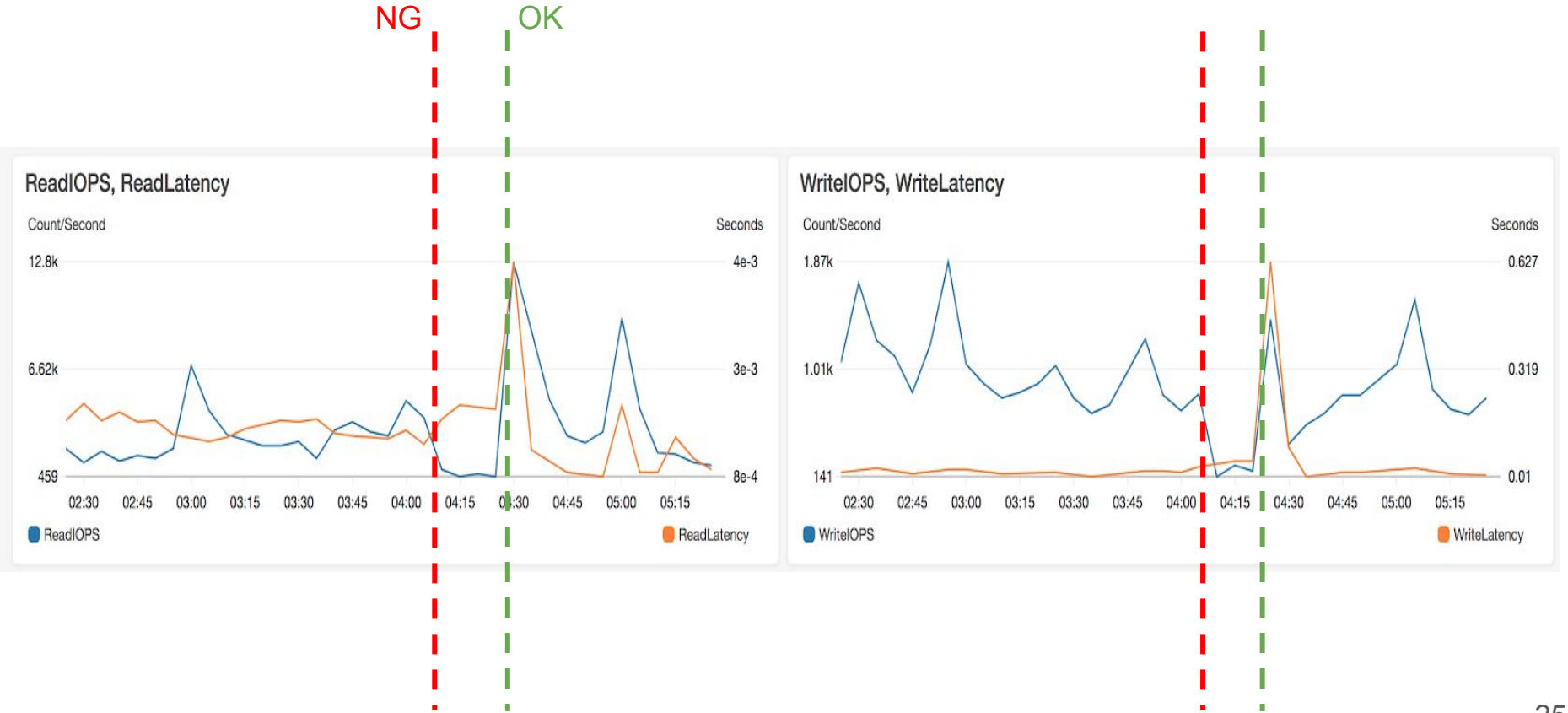
https://www.postgresql-archive.org/Relation-extension-scalability-tc5843810.html

New Facts

- Relation extension is slow on PostgreSQL 9.5 or older
  - A new page is added when a relation has no room to add a new record
  - Shared buffer entry eviction happen on relation extension with EXCLUSIVE lock
    - Victim buffer selection (clock sweep) & dirty page flush are slow

- But why it doesn't surface usually?
  - PostgreSQL page size = 8kB, Avg record size = 300B
    -> extension happens every 8kB / 300B = 27 insert queries
  - Avg INSERT query rate = 1.2k/sec
    -> extension happens 1.2k/s / 27 = 45/s
  - This amount of extension usually doesn't cause a issue

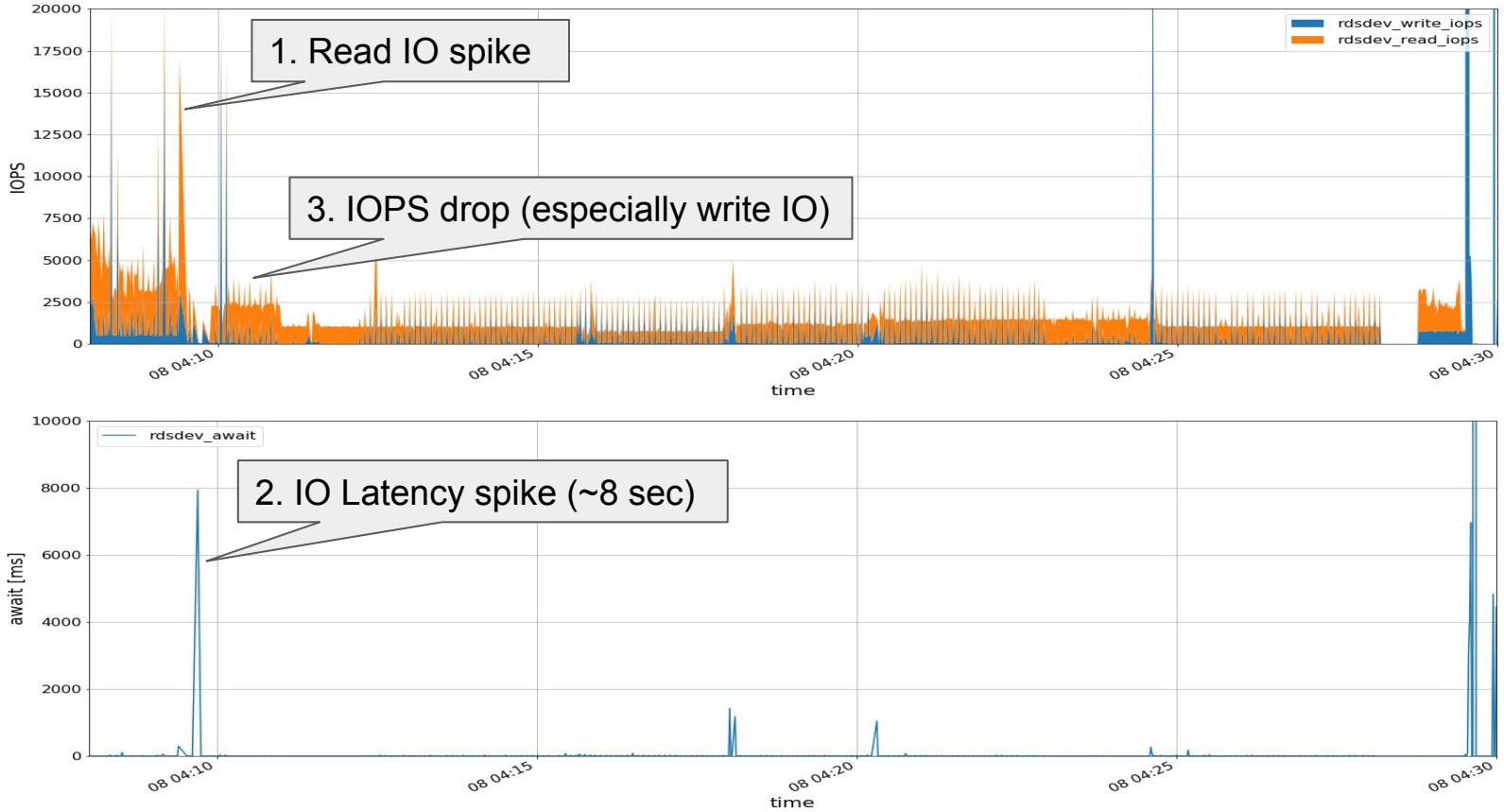
No Disk IO spike on CloudWatch Metrics



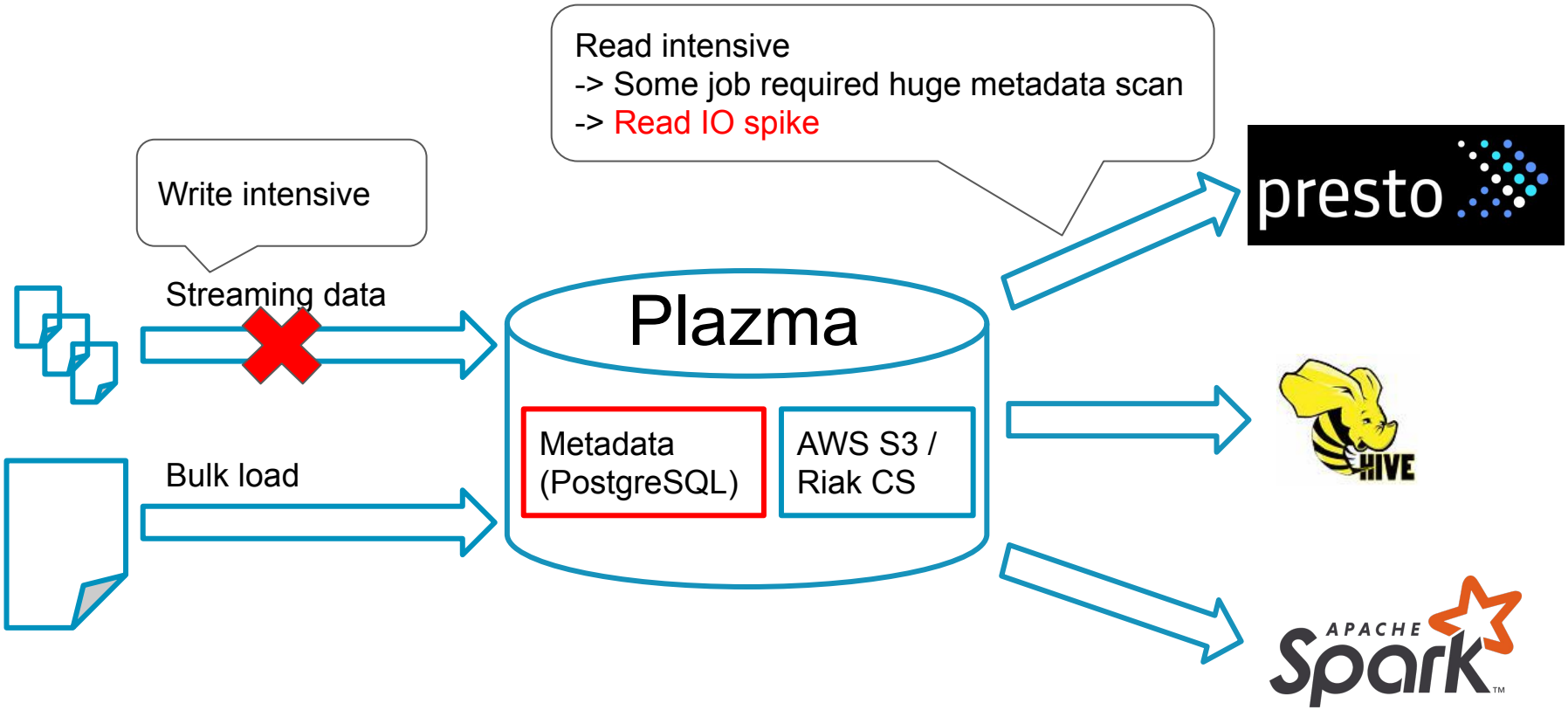
Resolution of Metrics

- CloudWatch : 60 seconds
  - Used for daily monitoring
- RDS Enhanced Monitoring : 1 - 60 seconds
  - Enabled but resolution was 30 seconds (default)
  - Changed resolution to 1 second

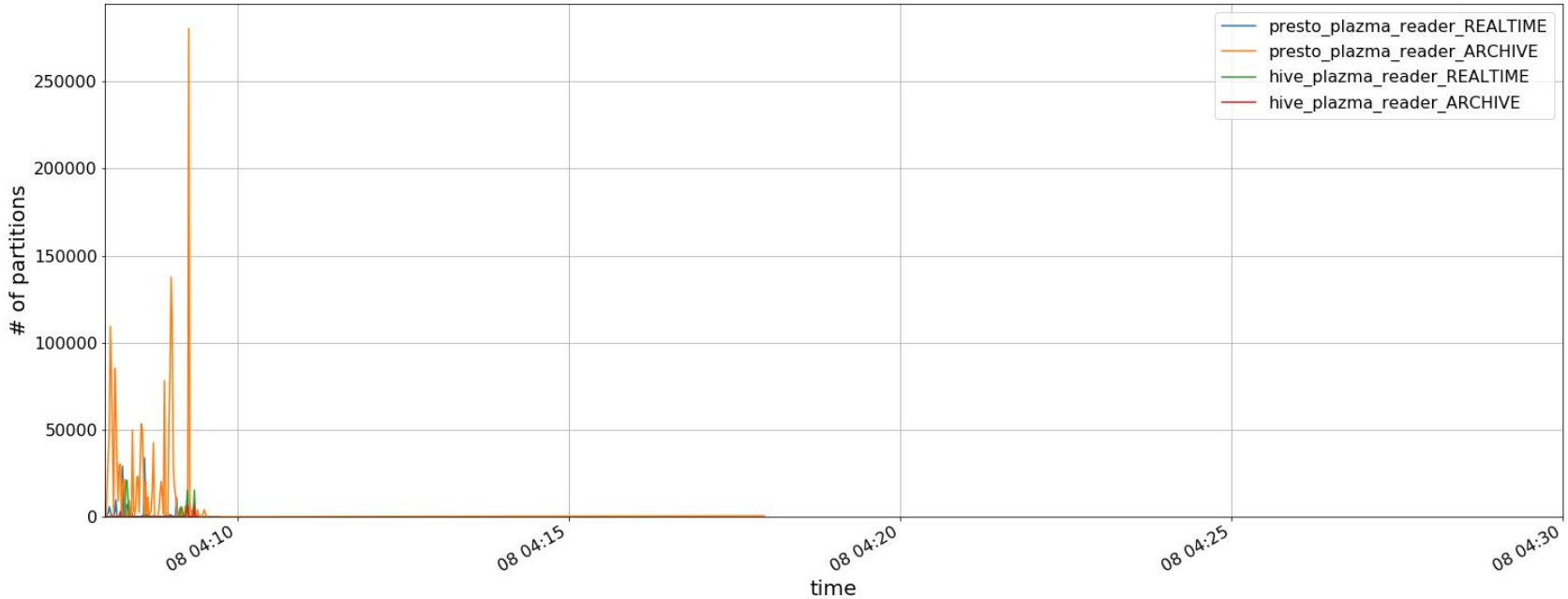
IOPS and Latency in 1 Second Resolution



Workload Characteristics



Metadata Read Workload in 1 Second Resolution



Short time spike is mostly OK but it was a cause this time

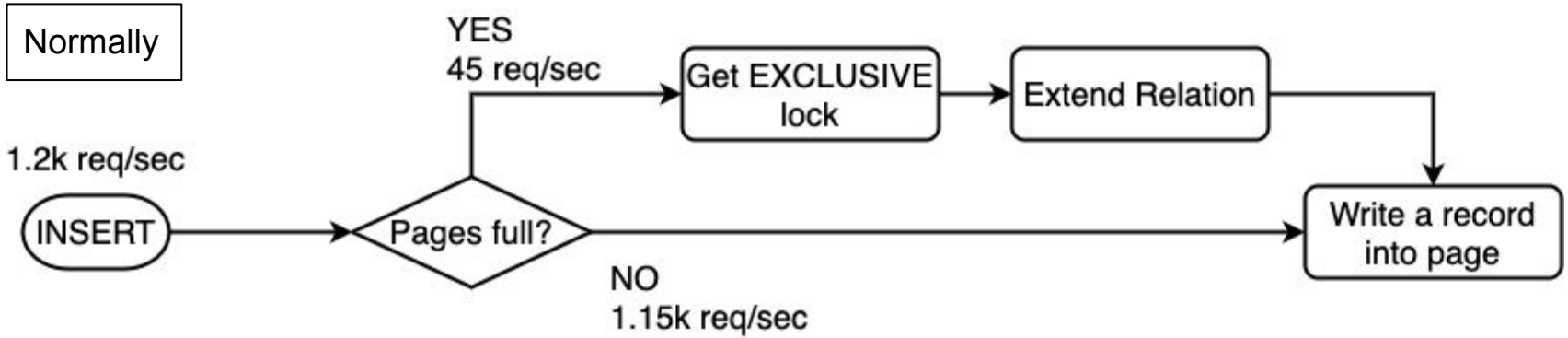
- metadata size (1.2TB) is much larger than shared buffer size (144GB)

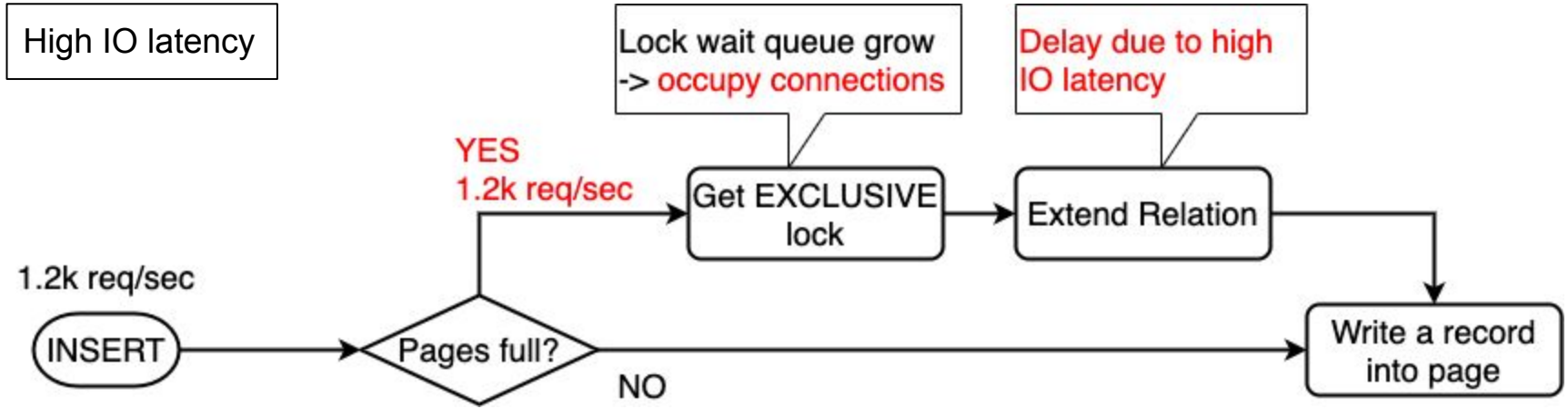
How did the issue happen?

1. Read workload spike from query engine applications
2. Run out Provisioned IOPS and IO latency became large temporarily
3. Dirty page flush became very slow due to high IO latency

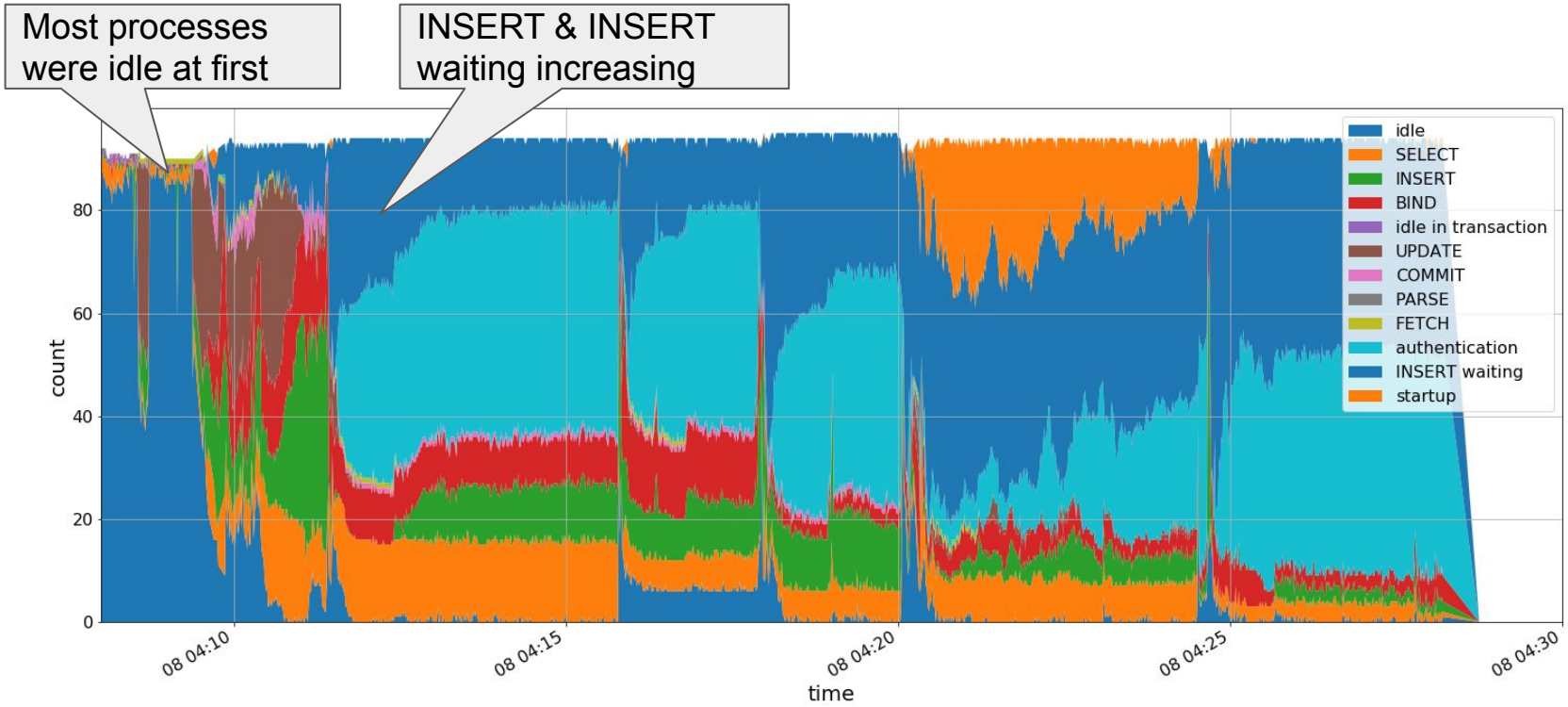
(continue to next slides)

Many INSERT Queries Waited Extension





Connections were Occupied by INSERT



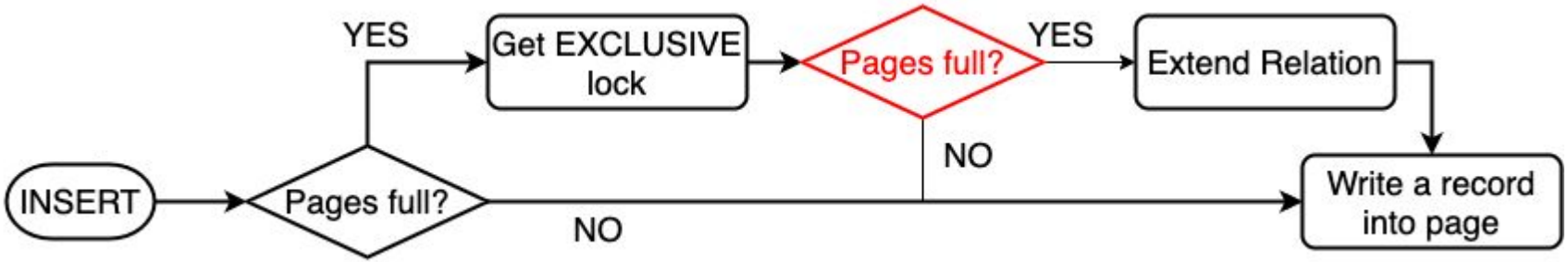
(From “processList” of RDS Enhanced Monitoring)

How did the issue happen?

1. Read workload spike from query engine applications
2. Run out Provisioned IOPS and IO latency became large temporarily
3. Dirty page flush became very slow due to high IO latency
4. Too many Insert queries were queued to extend the relation
5. Streaming import application get new connections due to lack of idle connections
   - DB connections were occupied -> other applications can't get new connections
6. Queries were timed out because of statement timeout (1 min) while lock wait
   - Insert queries flooded due to retry
7. Queries were retried but not succeed due to long lock wait time again
   - Most of lock wait time was wasted by timeout
     -> DB looked like hanging as it could hardly do meaningful work

Summary of the Issue

- Bottleneck changed from disk IO to locks
  - Read IO spike was short time -> couldn't find on 1 mins resolution metrics
- Queries flood by retry made recovery difficult
  - Reboot solved the issue by eliminating sticking queries
- Patch had been applied on PostgreSQL 9.6
  - ours was 9.4 -> resolved by upgrade



Learnings (1/2)

- Performance issues can happen anywhere
  - Our applications don’t use lock levels conflicting each other
  - No DDL was running
  - But still a strong lock was taken internally

- Metrics should be collected for all resources
  - Utilization / Saturation / Errors for all resources to apply USE method
    - We didn’t have metrics of lock utilization and saturation
  - Reproduction may take long time
    - It took 1 week between 1st and 2nd occurrences
  - Visualize metrics to reduce time to understand a trend
    - Checked lock metrics on log files at first but it took time

Anti-Methods to Waste Cycles

- Streetlight Anti-Method
  - Check only where is easy to observe or you are familiar

- Blame-Someone-Else Anti-Method
  - “Maybe it’s network. Can you check with the network team?“
  - Can be wasteful for both you and other teams
  - Lack of data leading to the hypothesis causes this

Unknown Unknowns

- Known Unknowns
  - Things you know but you haven’t observed yet

- Unknown Unknowns
  - Things that you do not know its existence
  - Tough to notice if a root cause is here

Expand Known Unknowns

- USE method
  - Systematically increase coverage

- Collect metrics to share Known Unknowns
  - Experienced engineers has more Known Unknowns
    but as long as it's only in your brain, knowledge transfer doesn't happen

  - Visualize and share it with teams
    - More eyes will reduce bias and bring more chances to notice a problem

Learnings (2/2)

- Take care of metrics source & resolution
  - 1 minute resolution wasn’t enough
- Understand workload characteristics
  - To detect the condition of the issue
  - To come up with a remediation plan
- Keep update libraries / middlewares versions
  - The issue doesn’t happen if PostgreSQL is up to date

Summary

- Collecting metrics is a matter of high priority
  - Lack of metrics leads to have a wrong point to start investigation
    - You will waste more time than time to implement it from the beginning
  - Collect and visualize all metrics as long as cost and security allow
  - Filter on read time, not on collecting time

- Check all resources exhaustively
  - Use systematic approach, not ad hoc check
    - USE method vs Anti-methods
    - Eventually reduce diagnose cycle iteration to reach a root cause

arm
TREASURE DATA

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
تشكر