arm TREASURE DATA

Dive into PlazmaDB Performance Characteristics And Future Challenges

2018/10/16 TD tech talk

Keisuke Suzuki
Software engineer

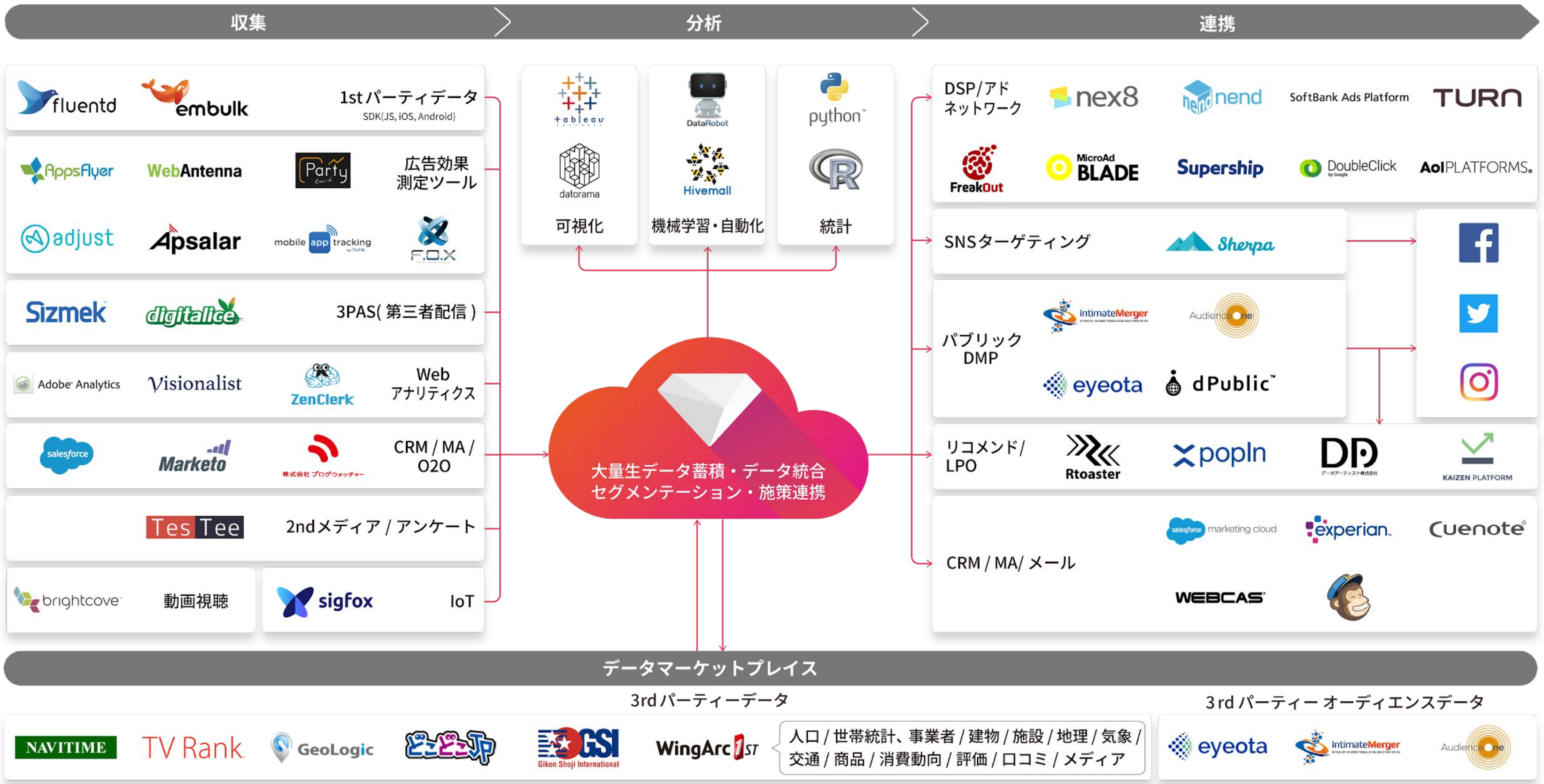
Who am I?

Keisuke Suzuki

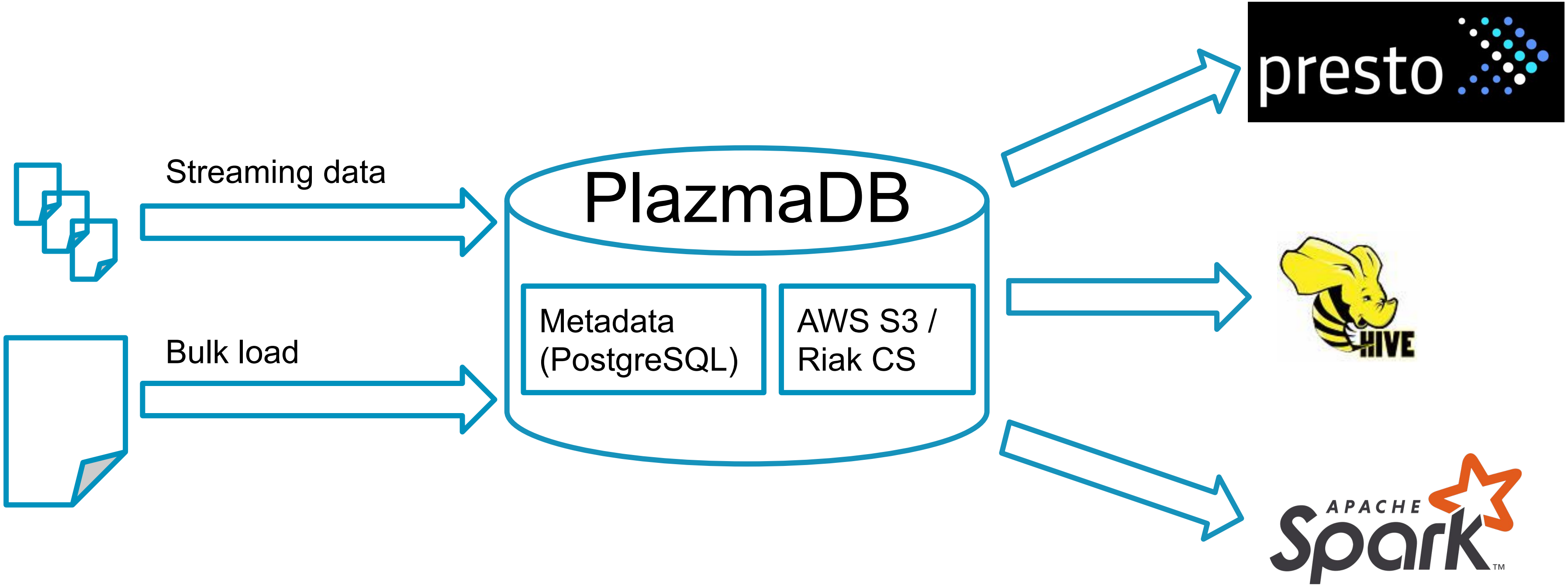
- Backend Engineer @ Treasure Data KK
  - PlazmaDB: distributed storage
  - Datatank: data mart
- DB / Distributed system / Performance optimization
- Twitter: @yajilobee

Treasure Data & PlazmaDB

Arm Treasure Data eCDP



PlazmaDB


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

Daily Workload & Storage Size

Import

500 Billion Records / day
~ 5.8 Million Records / sec

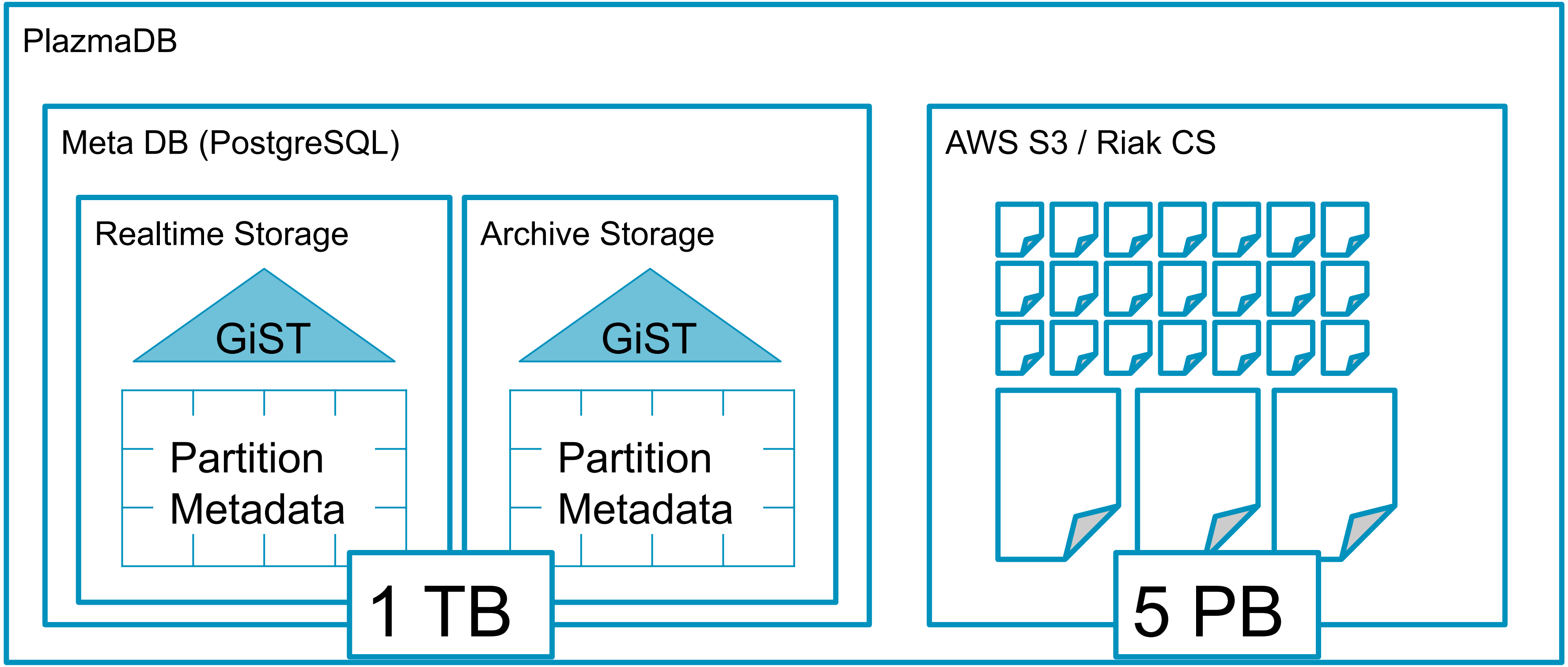
Query

600,000 Queries / day
15 Trillion Records / day

Storage size

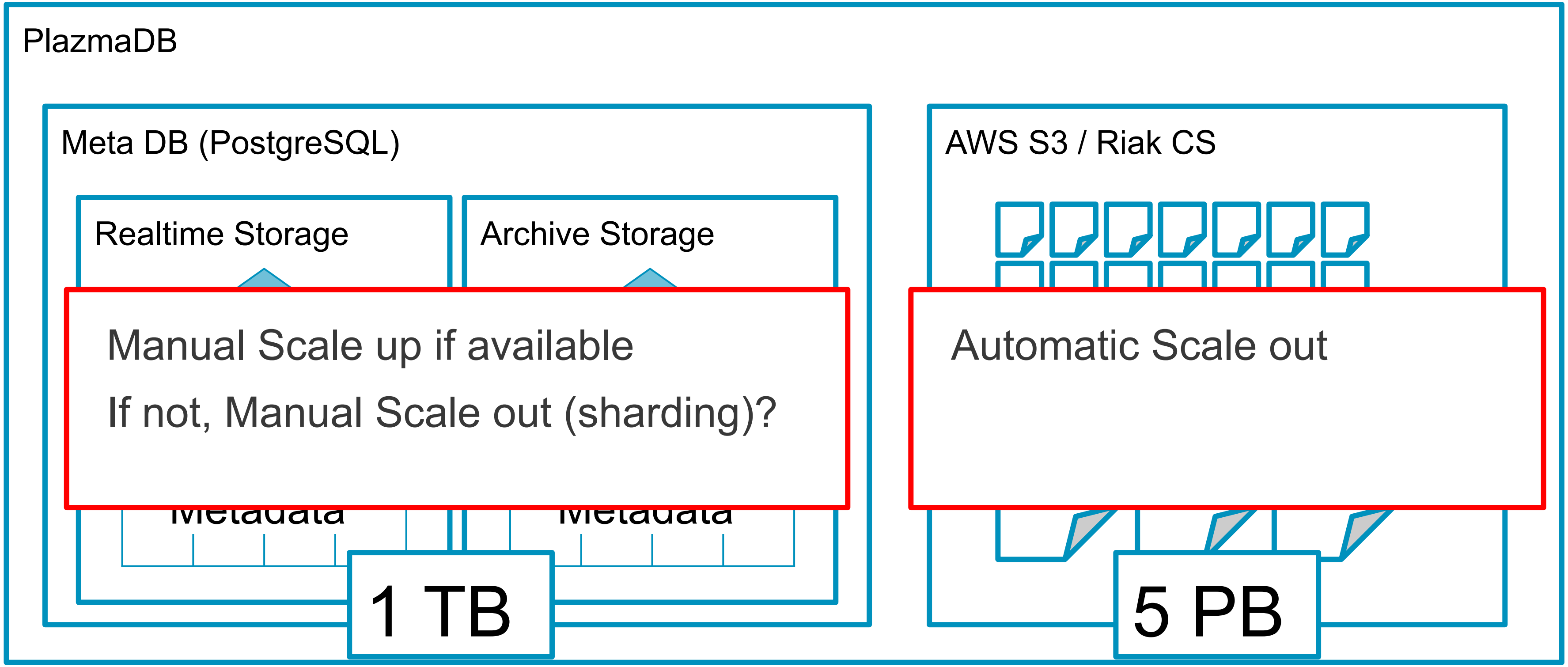
5 PB (+5~10 TB / day)
55 Trillion Records

Data Volume

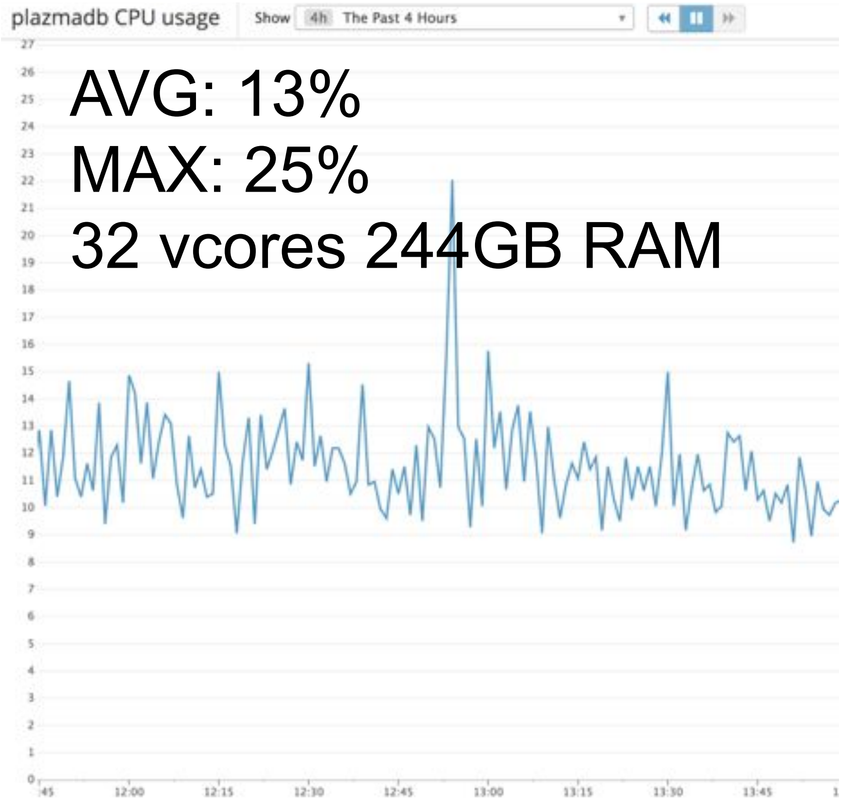


PlazmaDB
Meta DB (PostgreSQL)
Realtime Storage
GiST
Partition
Metadata
Archive Storage
GiST
Partition
Metadata
1 TB
AWS S3 / Riak CS
5 PB

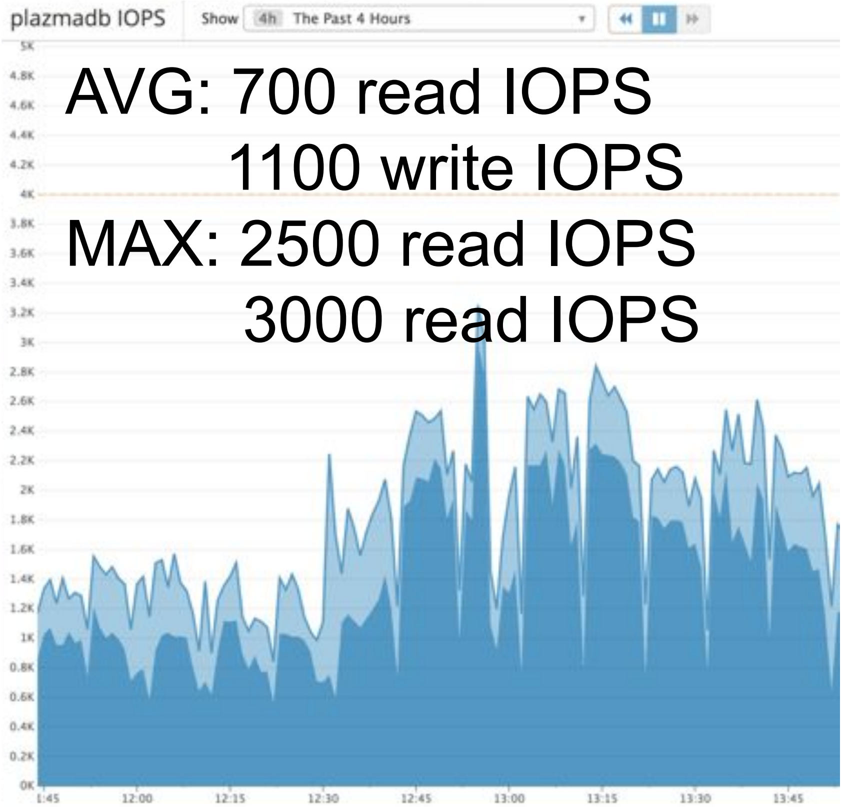
Scaling Strategy



Current MetaDB CPU / IO utilization



AVG: 13%
MAX: 25%
32 vcores 244GB RAM



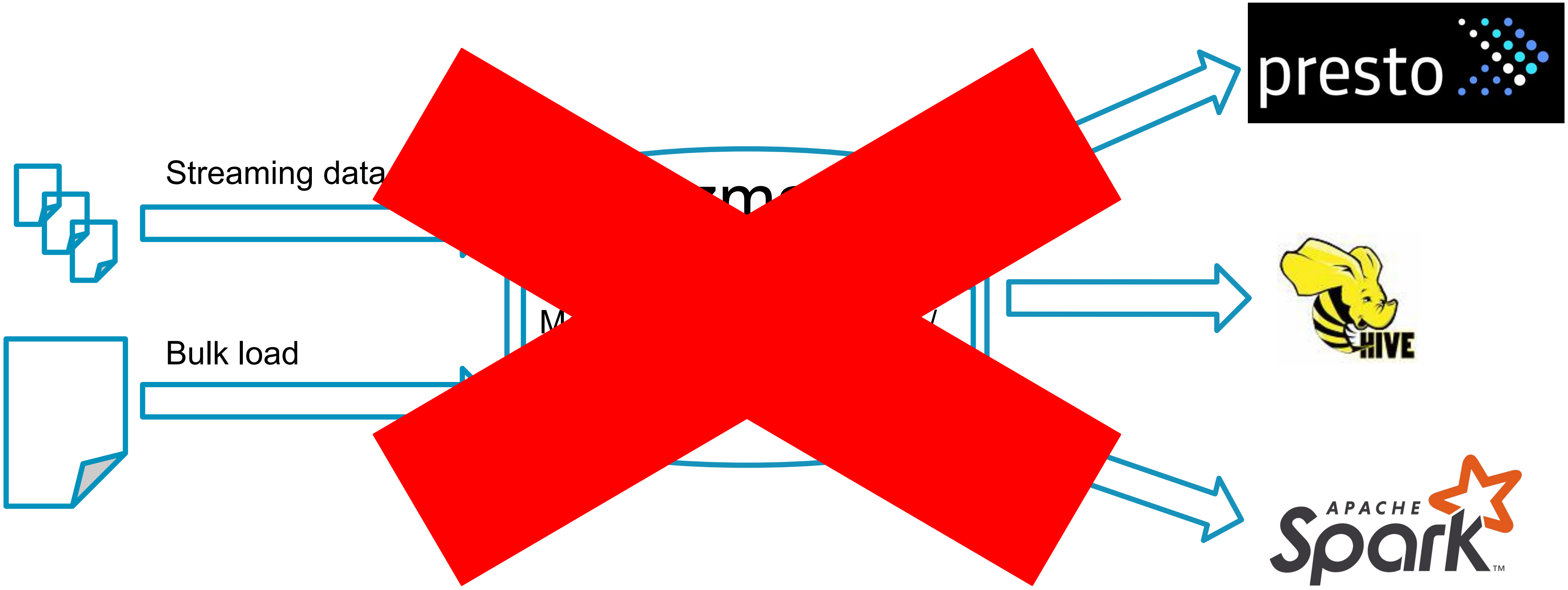
AVG: 700 read IOPS
1100 write IOPS
MAX: 2500 read IOPS
3000 read IOPS

arm TREASURE DATA

Q: What is MetaDB capacity?

A: Nobody knew

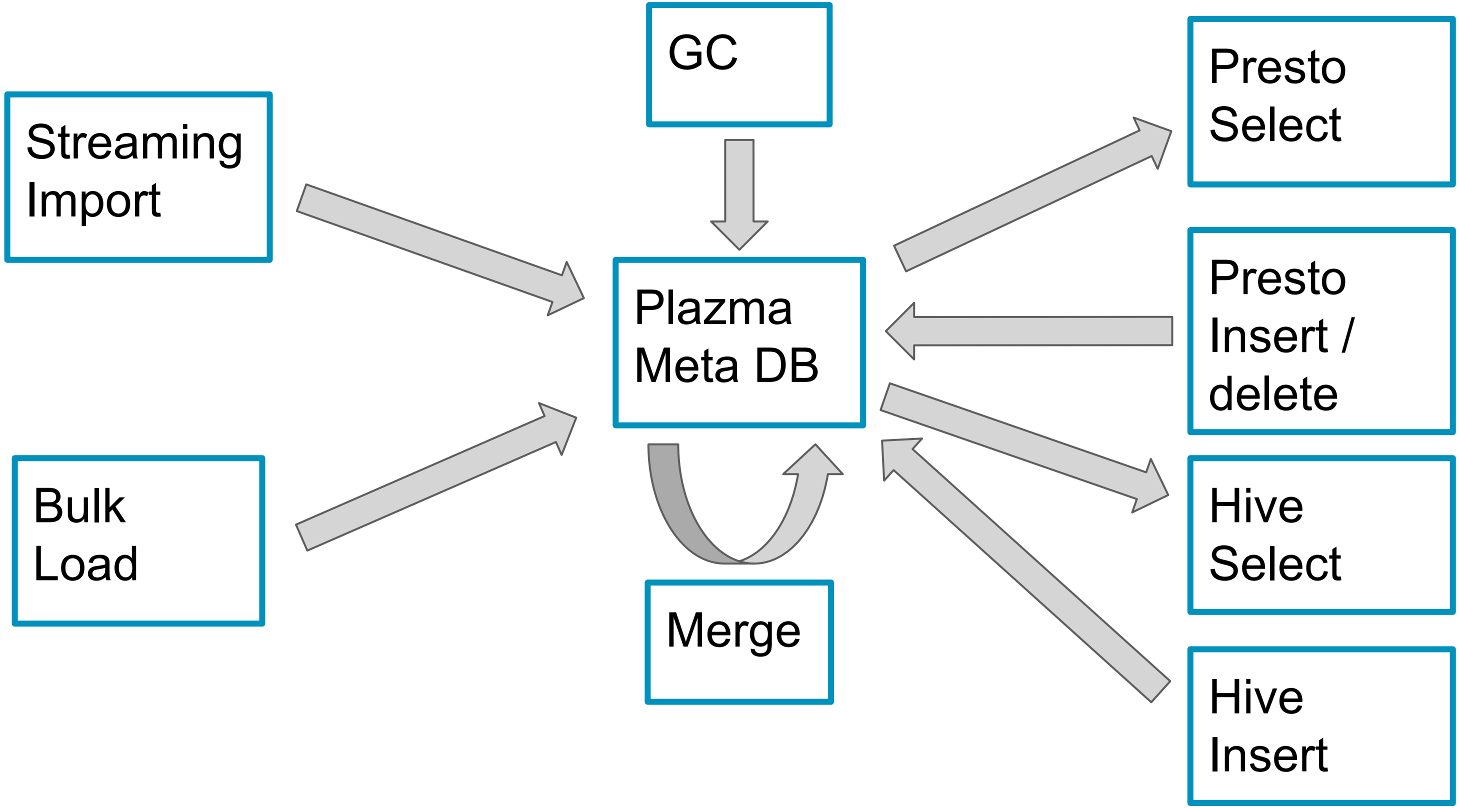
If PlazmaDB is down..



If PlazmaDB down..

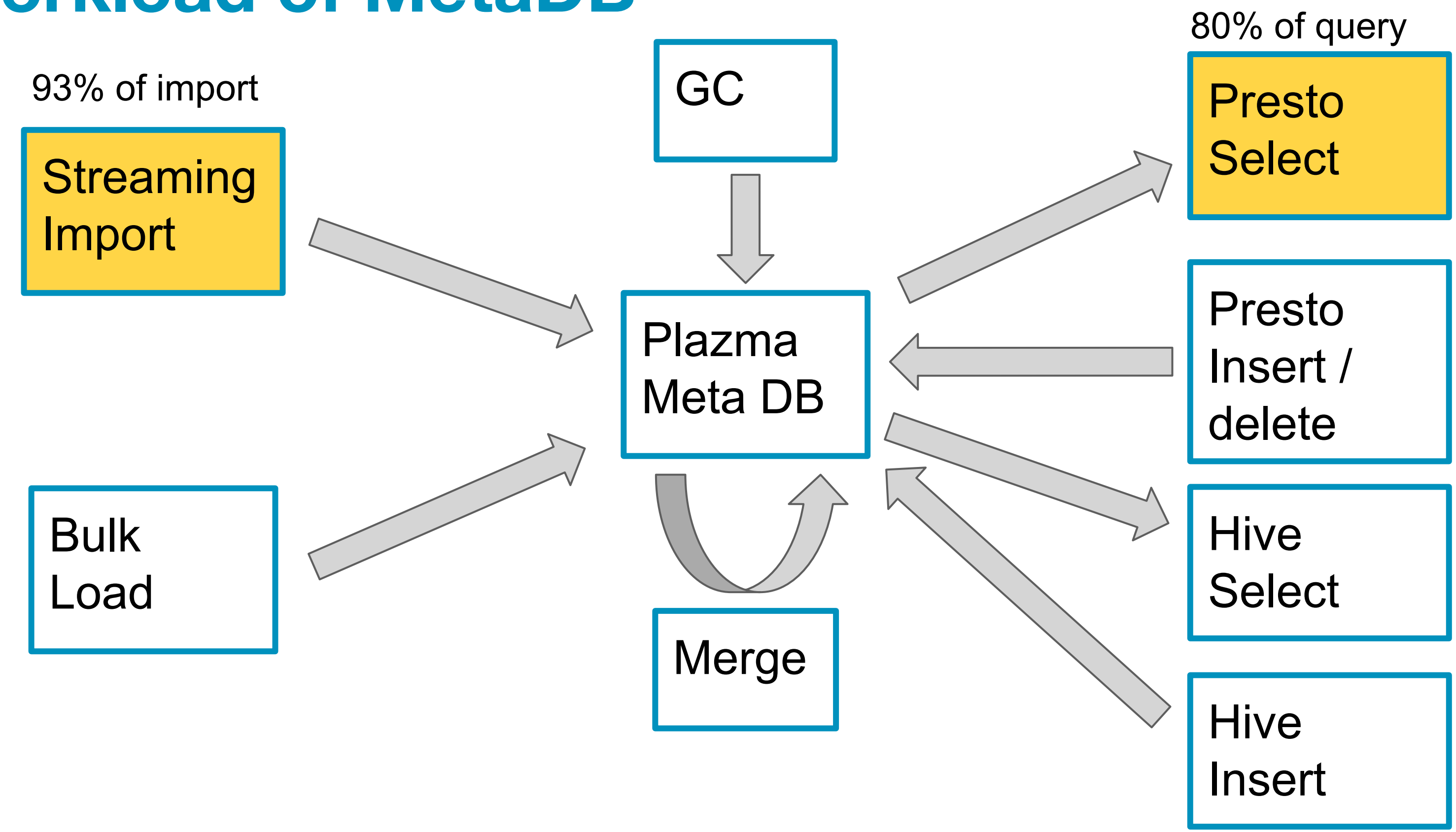
Benchmarking Plazma MetaDB

Workload of MetaDB

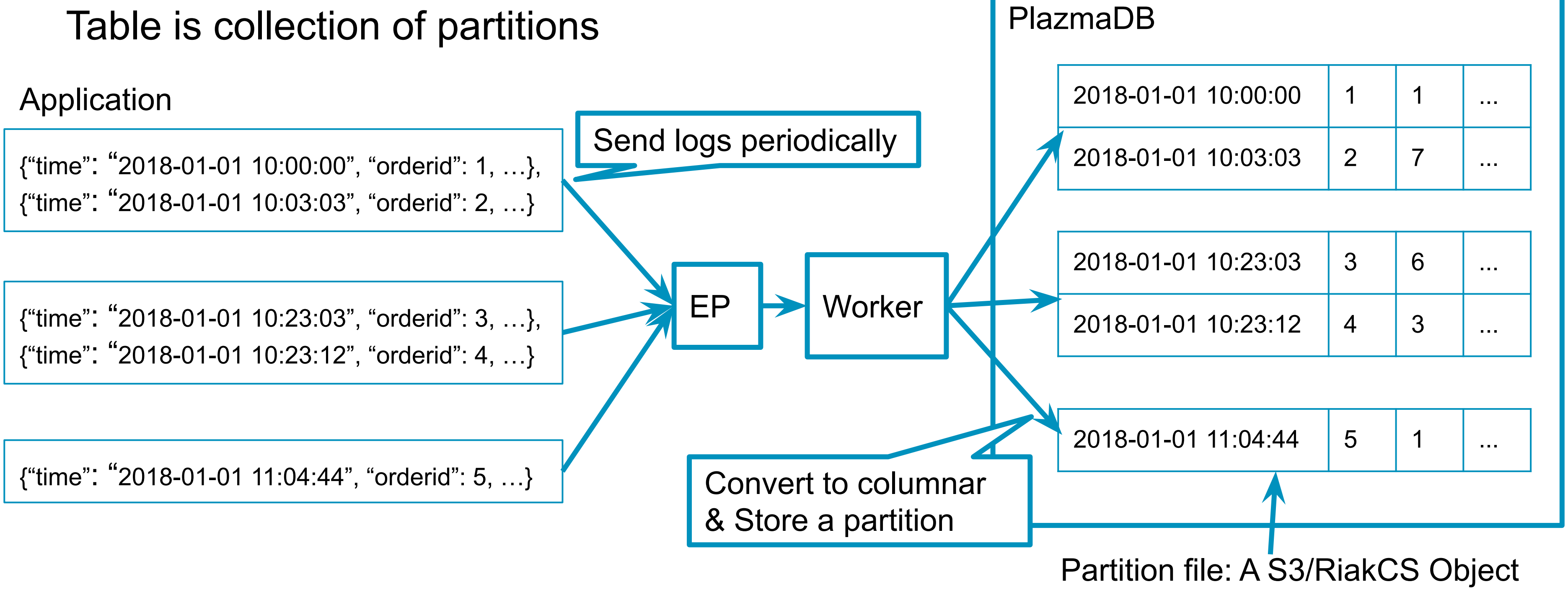


arm TREASURE DATA

Workload of MetaDB



PlazmaDB Streaming Import



PlazmaDB Metadata

PlazmaDB is Multi tenant

data_set_id: ID combination of User, Database, Table

Meta DB (PostgreSQL)

data_set_id	path	...
1		
1		
1		
2		

AWS S3 / RiakCS

Data set 1

2018-01-01 10:00:00	1	1	...
2018-01-01 10:03:03	2	7	...

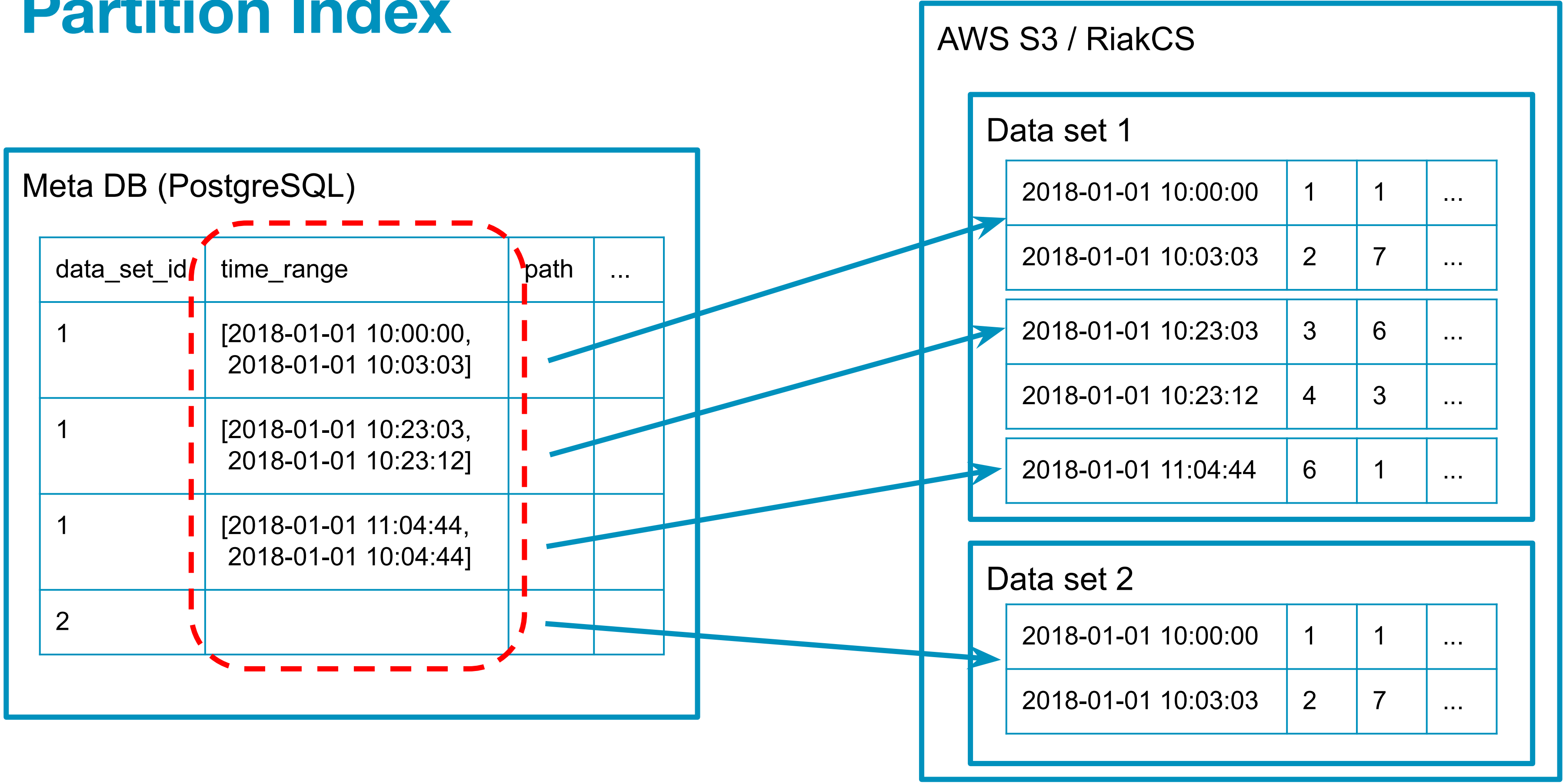
2018-01-01 10:23:03	3	6	...
2018-01-01 10:23:12	4	3	...

2018-01-01 11:04:44	5	1	...

Data set 2

2018-01-01 10:00:00	1	1	...
2018-01-01 10:03:03	2	7	...

Partition Index



Partition Lookup on Analytical Query Processing

Meta DB (PostgreSQL)

data_set_id	time_range	path	...
1	[2018-01-01 10:00:00, 2018-01-01 10:03:03]		
1	[2018-01-01 10:23:03, 2018-01-01 10:23:12]		
1	[2018-01-01 11:04:44, 2018-01-01 10:04:44]		
2			

```
SELECT
  region,
  SUM(price)
FROM
  orders  -- assume this is data set 1
WHERE TD_TIME_RANGE(time,
'2018-01-01 10:00', '2018-01-01 11:00')
GROUP BY
  region
```

Real World Application is Complicated..

Many Performance Related Factors

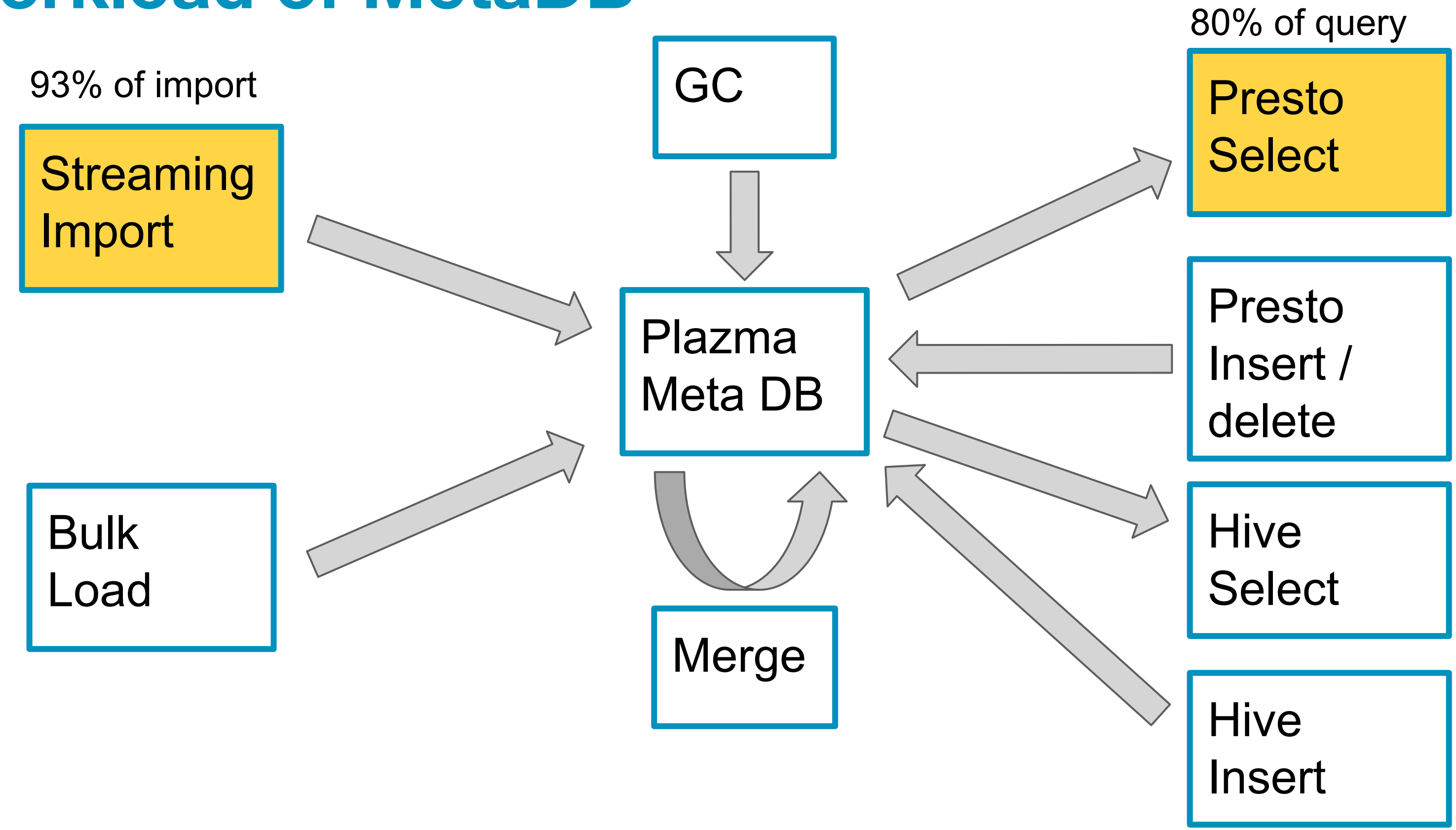
- Type of Workload
- Users' behaviour
  - # of data sets (tables)
  - # of import data
  - # of analytical query request
  - Data skew
- Metadata Storage size
- Server Size (CPU cores, RAM, Storage, Network)
- etc..

Goal of Benchmarking

Define the end of Benchmark task

- Capacity Planning
- Regression Test
- Compare Performance
- Parameter Tuning
- etc..

Workload of MetaDB



arm TREASURE DATA

Benchmarking Streaming Import

arm

Model of Streaming Import Workload

data_set_id	path	...
3		

Insert a partition metadata

Meta DB (PostgreSQL)

data_set_id	path	...
1		
2		
3		
2		

Performance related factors

Concurrency	(1, 2, 4, 8, 16, 32, 64, 128, 256)
Size of tuple	random based on normal distribution (185 byte on average)

Benchmarking Environment

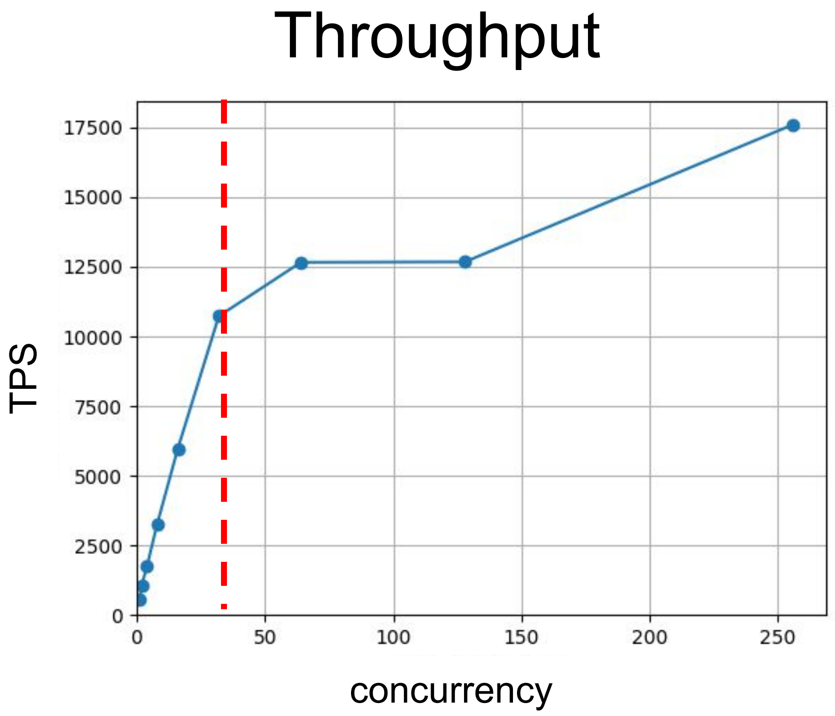
- AWS RDS PostgreSQL

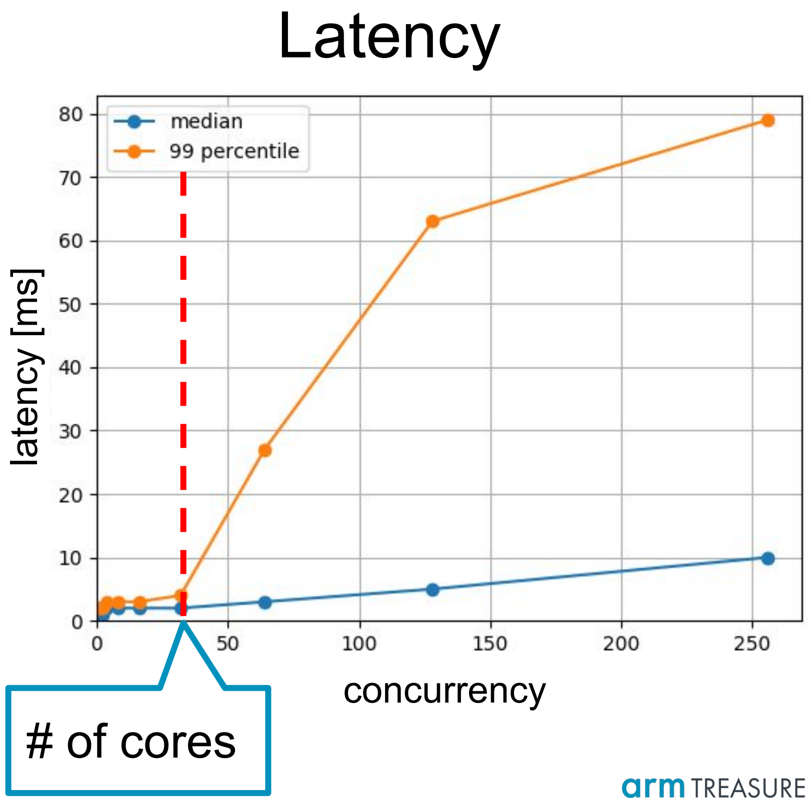
Instance type	db.r3.x8large (32 vcores, 244GB RAM)
Provisioned IOPS	4k
PostgreSQL version	9.4.17

- PostgreSQL parameters

shared_buffers	160GB (~ 60% of RAM)
checkpoint_segments	1500 (24GB)

Scalability of Streaming Import Workload

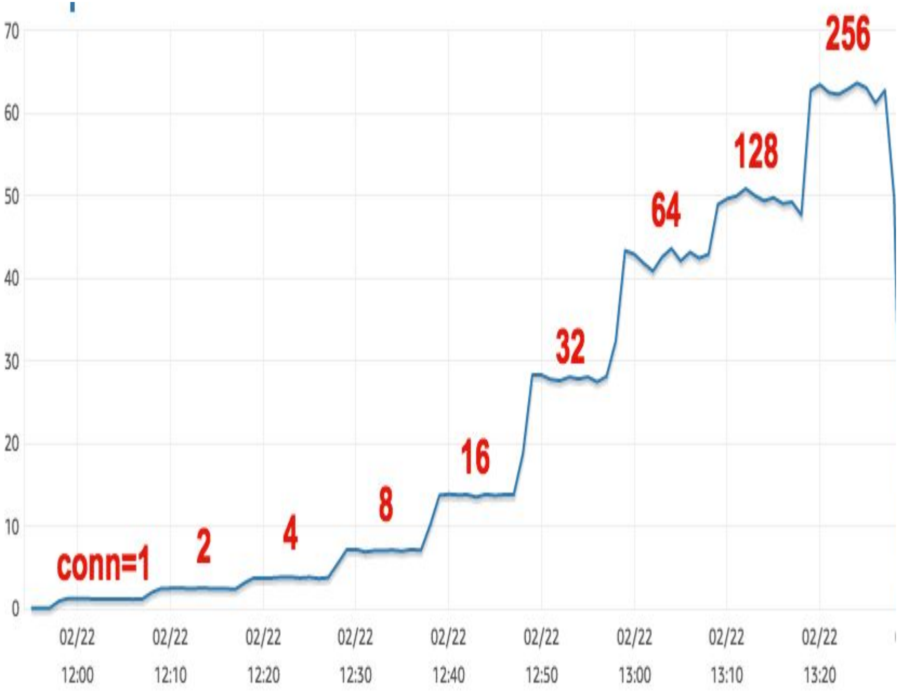




arm TREASURE DATA

Resource Consumption

CPU utilization

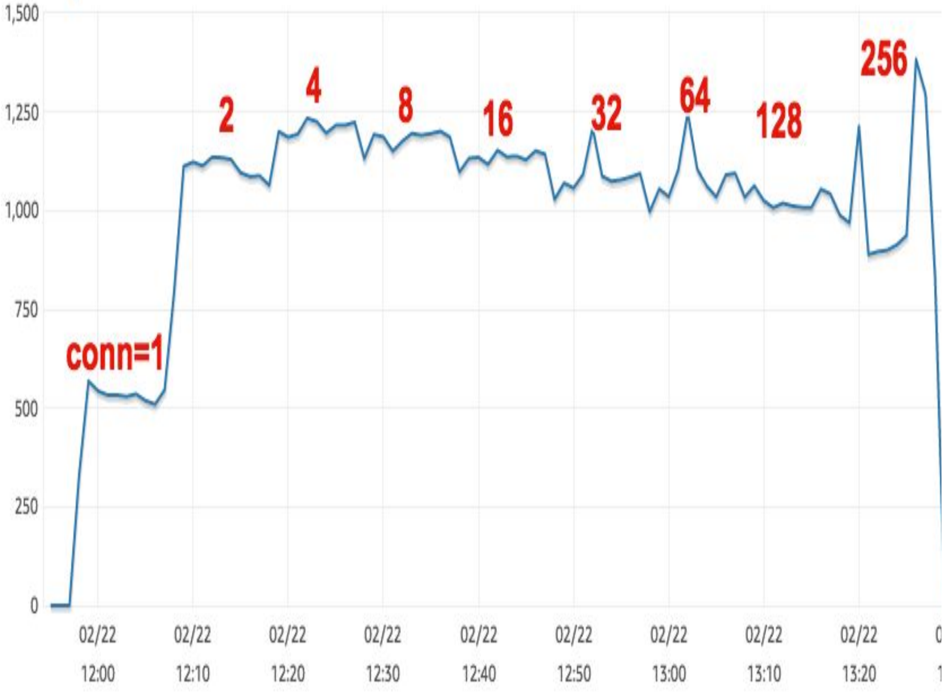


Write IO throughput

Write IO

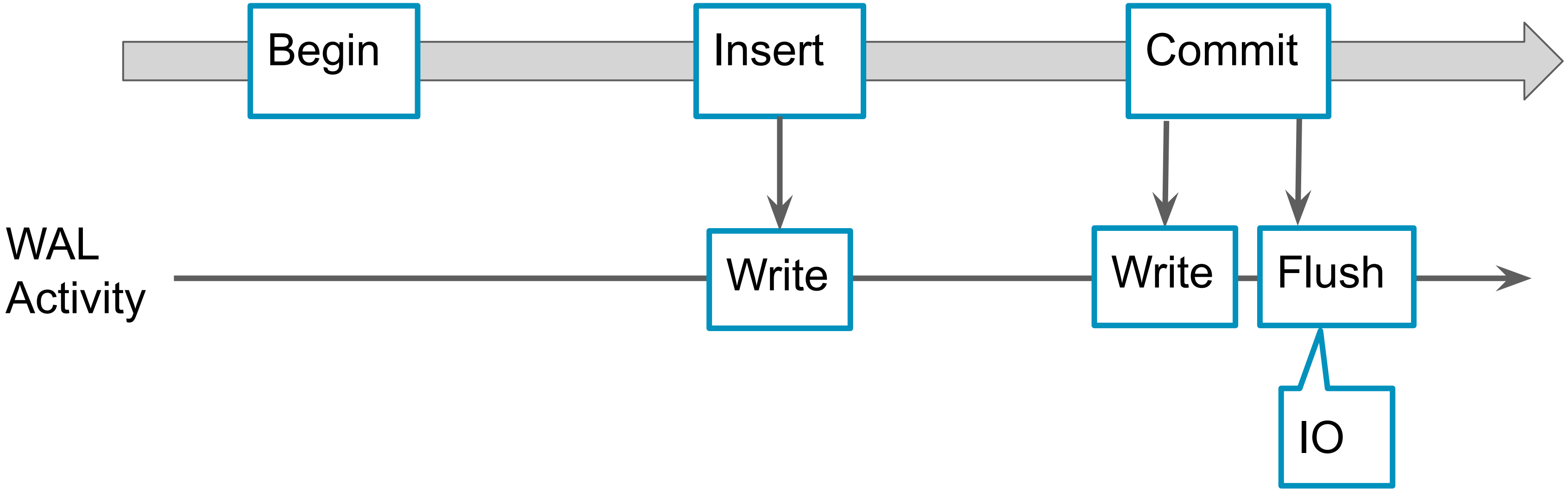
Write IO throughput

Write IOPS

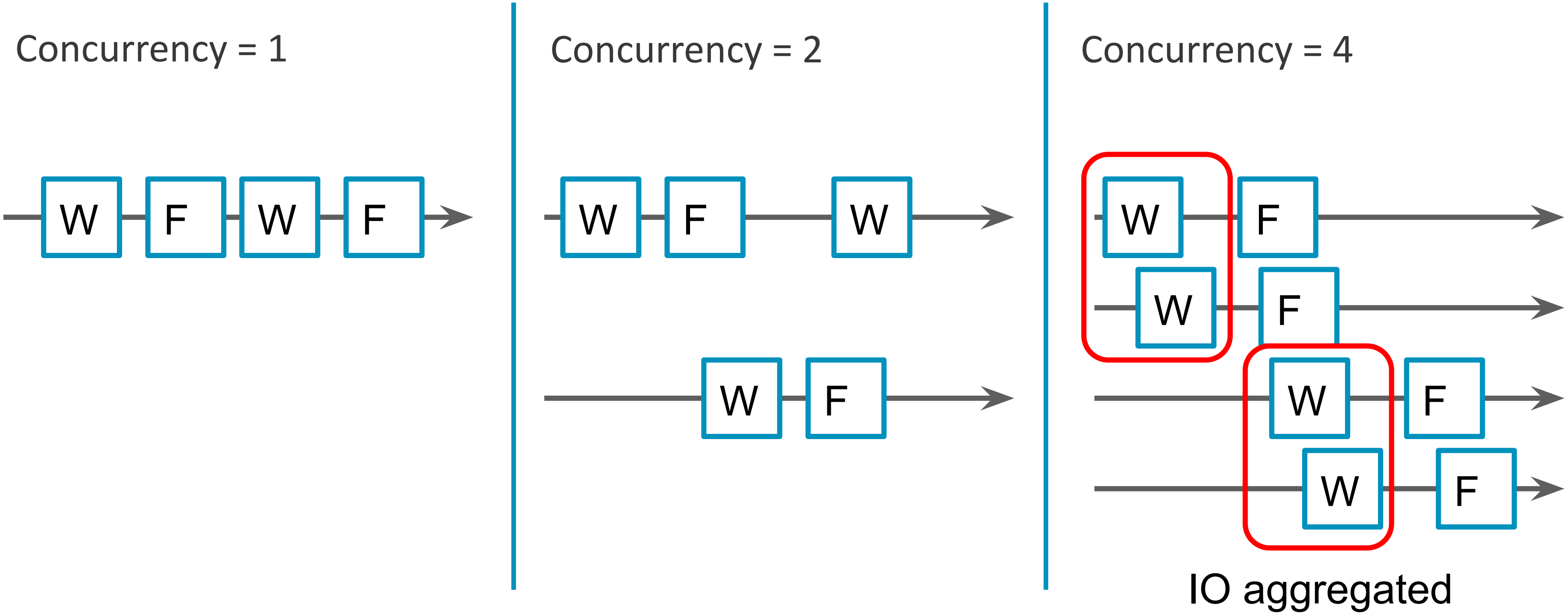


arm TREASURE DATA

When Write IO issued?

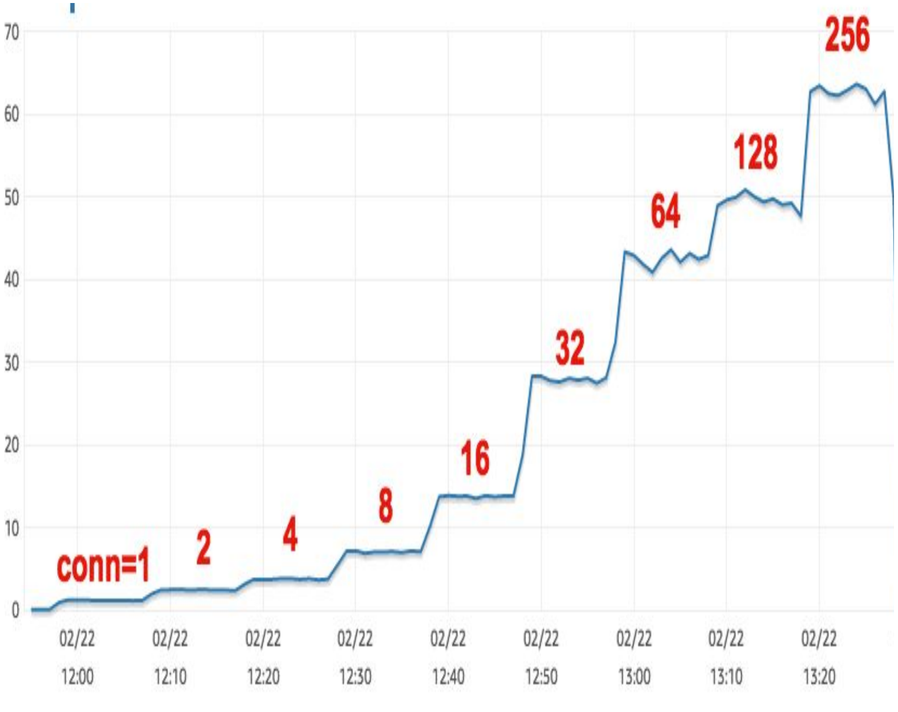


Concurrency and Write IO

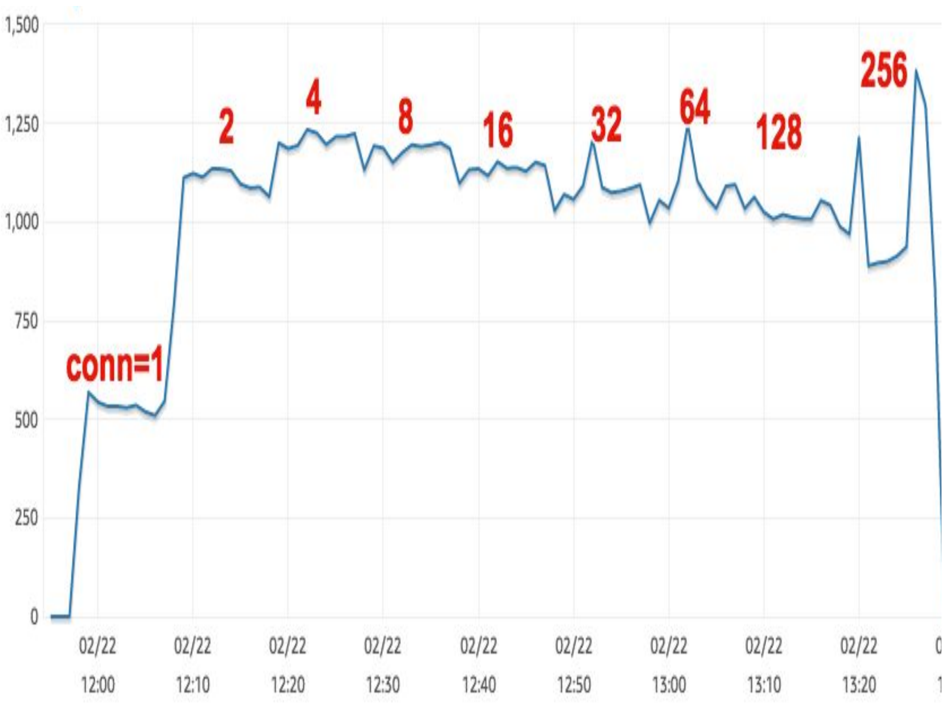


Bottleneck

CPU utilization



Write IOPS



arm TREASURE DATA

Summary of Streaming Import Workload

- CPU bottleneck
  - Scale almost linearly when concurrency is less than # of cores
  - Throughput can increase after that, but tail latency increases as well
  - Write IOPS doesn’t increase as increasing concurrency because of IO aggregation

Benchmarking Presto Select

arm

Model of Presto Select Workload

Meta DB (PostgreSQL)

data_set_id	time_range	path	...
1	[18-01-01 10:00, … 11:00]	a	
2	[18-01-01 10:00, … 11:00]	b	
1	[18-01-01 11:00, … 12:00]	c	
3	[18-01-01 13:00, … 14:00]	d	
2	[18-01-01 13:30, … 14:00]	e	
1	[18-01-01 16:00, … 17:00]	f	
...			
1	[18-01-02 03:00, … 04:00]	m	

data_set_id=1 and
time_range &&
[18-01-01 00:00, 18-01-02 00:00]

path
a
c
f

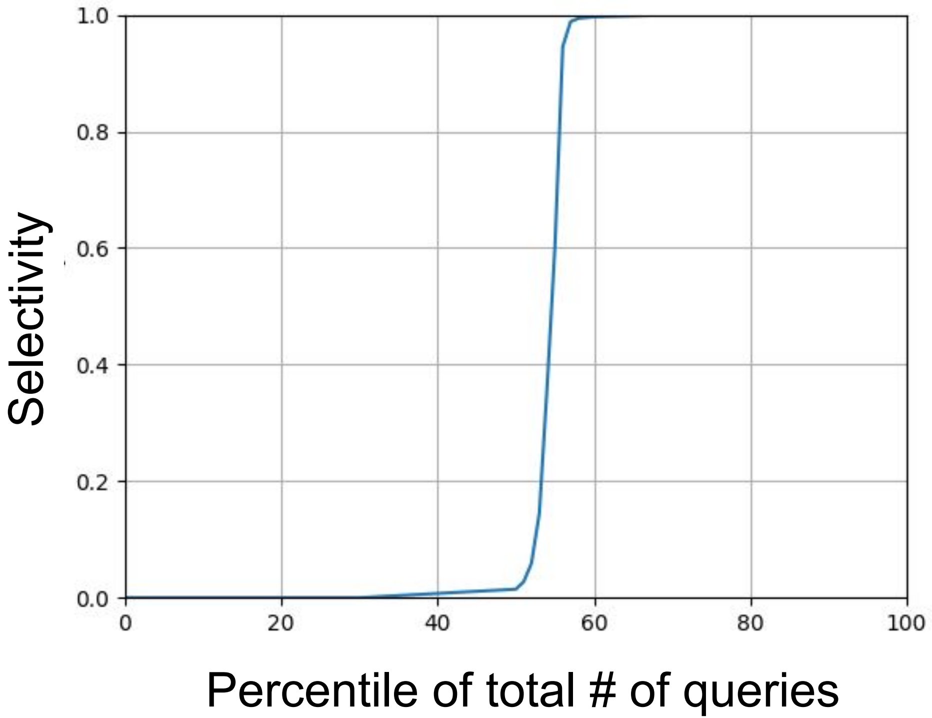
Index scan for data_set_id and time_range

Performance related factors

Concurrency	(16, 32, 64, 128, 256)
Metadata size	600GB (Dummy data based on actual trend)
# of data sets	30k
Time range to scan (selectivity)	(next slide)
Distribution of data set access frequency	(next slide)

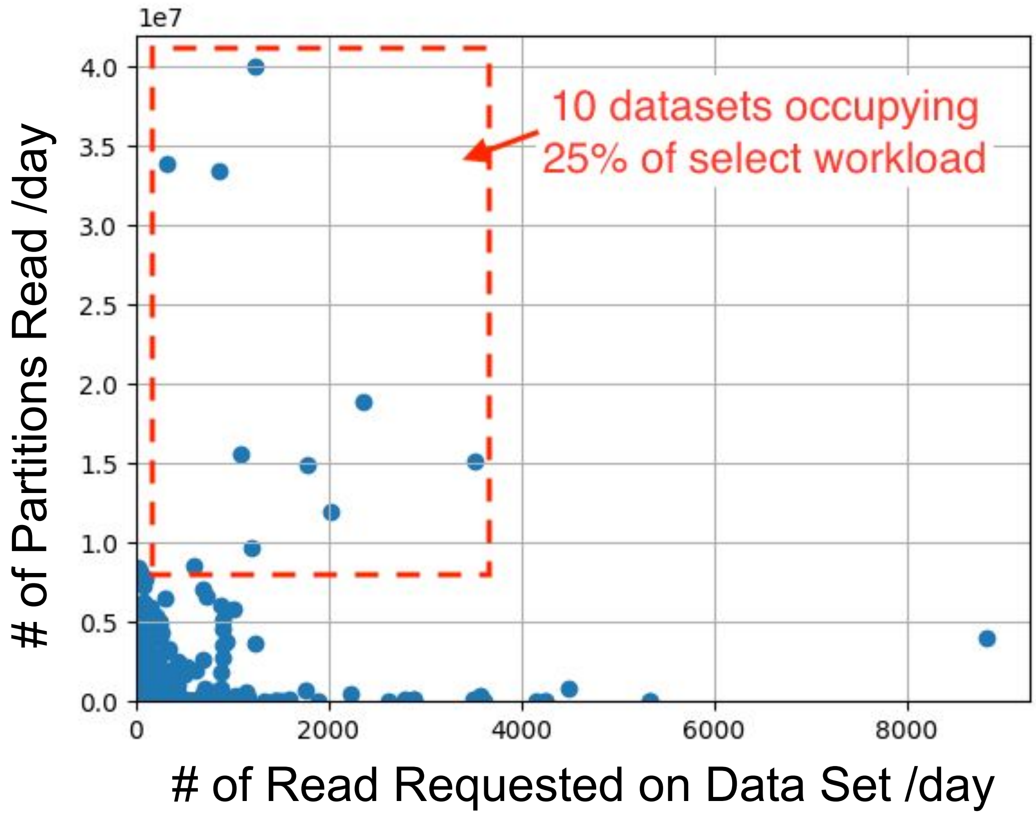
Selectivity

- Random sampling from actual selectivity distribution
  e.g.)
  - 40% queries: sl = 1
  - 5% queries: sl = [0.01, 0.5]
  - 5% queries: sl = [0.5, 0.99]

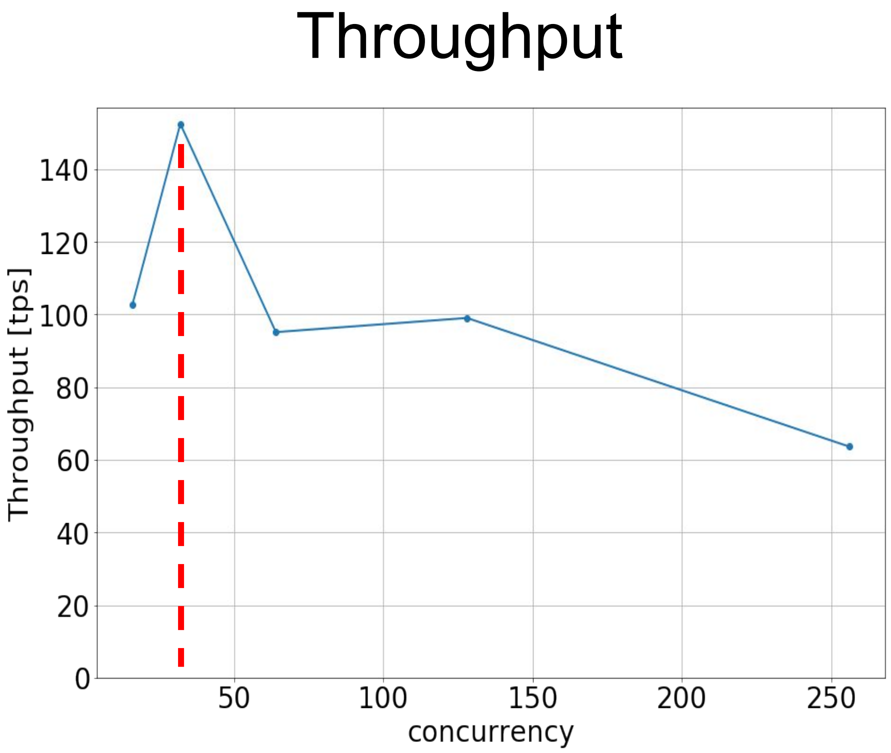


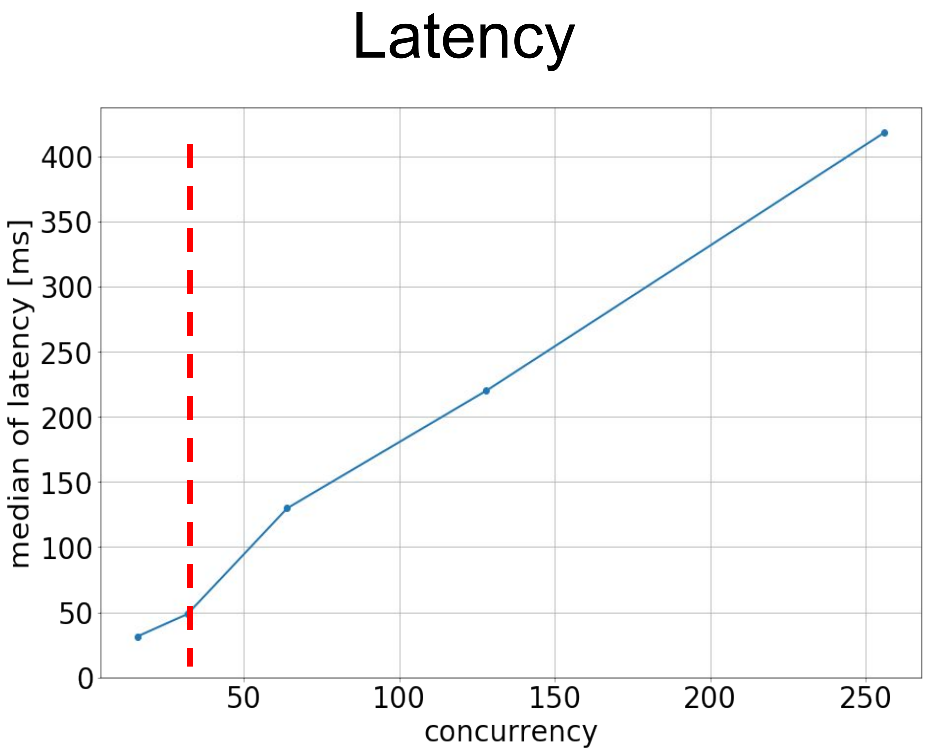
Distribution of Data Set Access Frequency

- Metadata size = 600GB
- Shared Buffer size = 160GB
- But, Hot Data size is smaller than Shared Buffer e.g.)
  - 85% of workload comes from 1% data sets
  - 95% of workload comes from 5% data sets



Scalability of Presto Select Workload



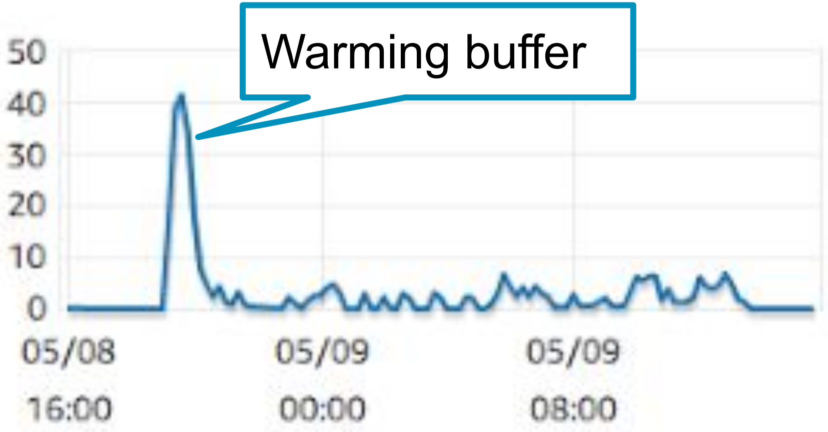


arm TREASURE DATA

Resource Consumption (Concurrency=128)

CPU utilization [%]

Read IO throughput [MB/s]



arm TREASURE DATA

Summary of Presto Select Workload

- CPU bottleneck
  - Scale almost linearly when concurrency is less than # of cores
  - Throughput decreases when concurrency is higher than # of cores
  - Hot data is small enough to fit into DB shared buffer

Benchmarking Mixed Workload

Mixing Streaming Import & Presto Select

data_set_id	...
3	

Insert

Meta DB (PostgreSQL)

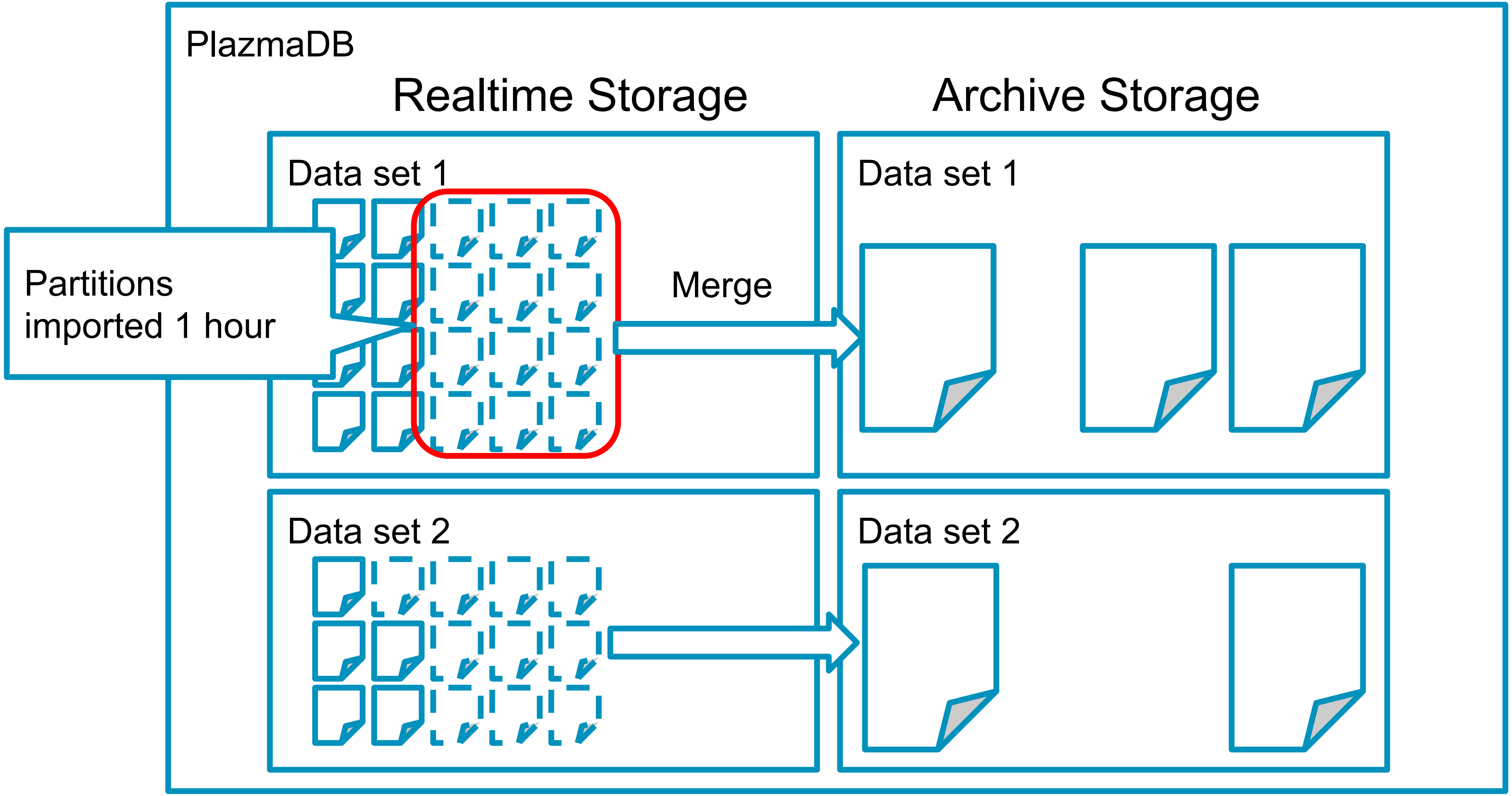
data_set_id	time_range	path	...
1	[18-01-01 10:00, … 11:00]	a	
2	[18-01-01 10:00, … 11:00]	b	
1	[18-01-01 11:00, … 12:00]	c	
3	[18-01-01 13:00, … 14:00]	d	
2	[18-01-01 13:30, … 14:00]	e	
1	[18-01-01 16:00, … 17:00]	f	
...			

Select

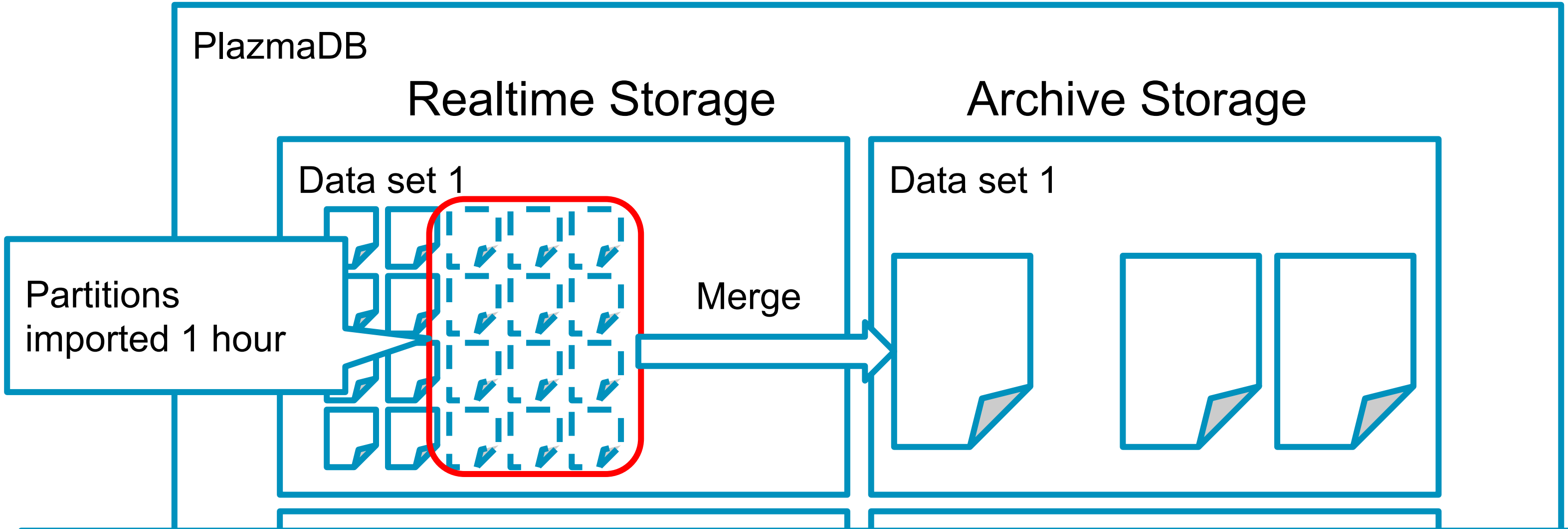
path
a
c
f

Insert : Select = 200 : 1

Realtime Storage & Archive Storage



Realtime Storage & Archive Storage



- Reduced to 1/20 - 1/100 partitions
- Merge can be delayed 5 - 7 hours

-> Metadata will be compressed but accumulate during delay

What is expectation?

- E.g. when concurrency = 64, what throughput will be?
  - Both Streaming Import and Presto Select were CPU bottleneck
- Ref: Single Workload Throughput
  - Streaming Import = 12500 tps
  - Presto Select = 95 tps
- Expectation
  - Streaming Import ~ 6000 tps ?
  - Presto Select ~ 45 tps ?

Result of mixed workload

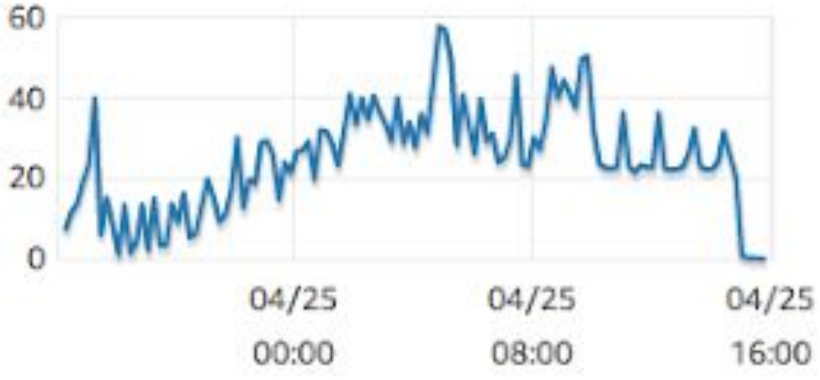
- Throughput when Concurrency = 64

	Streaming Import [tps]	Presto Select [tps]
mix concurrency=64	5547	27.6
Ref: single	12500	95

Resource Consumption

CPU utilization [%]

Read IO throughput [MB/s]



Bottleneck is changed to Disk IO

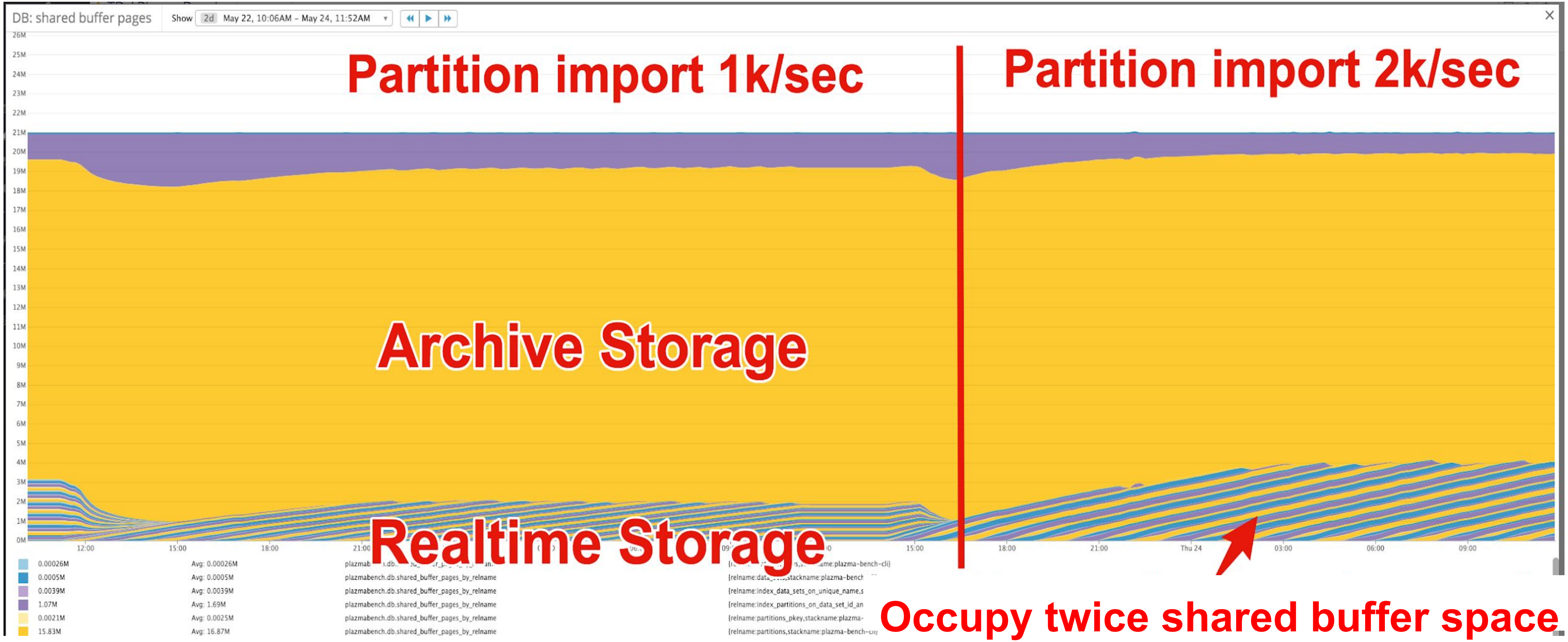
Increase PIOPS

	PIOPS	Streaming Import [tps]	Presto Select [tps]
mix concurrency=64	4k	5488	27.4
mix concurrency=64	20k	7179	35.9
Ref: single	4k	12500	95

Cache (DB Shared Buffer) Hit Ratio

	PIOPS	Streaming Import [tps]	Presto Select [tps]	RT storage size	Cache Hit Ratio
mix con=64	4k	5488	27.4	57GB	93%
mix con=64	20k	7179	35.9	75GB	89.5%
Ref: single	4k	12500	95		

Impact of Streaming Import increase



arm TREASURE DATA

Impact of Cache Miss

- Avg # of selected rows per a Presto Select = 8000
- A postgres page mostly includes only 1 row for a data set
  - A page (8kB) has 20 - 30 rows, but different data sets' data are stored together
- # of pages to scan per a Presto Select = 8000 / 1 = 8000
- # of pages to scan per second = 8000 * TPS
  - E.g. TPS=35 -> 8000 * 35 = 280k pages/sec
    -> 1% cache miss causes 2800 IOPS

Another factor: Auto Vacuum

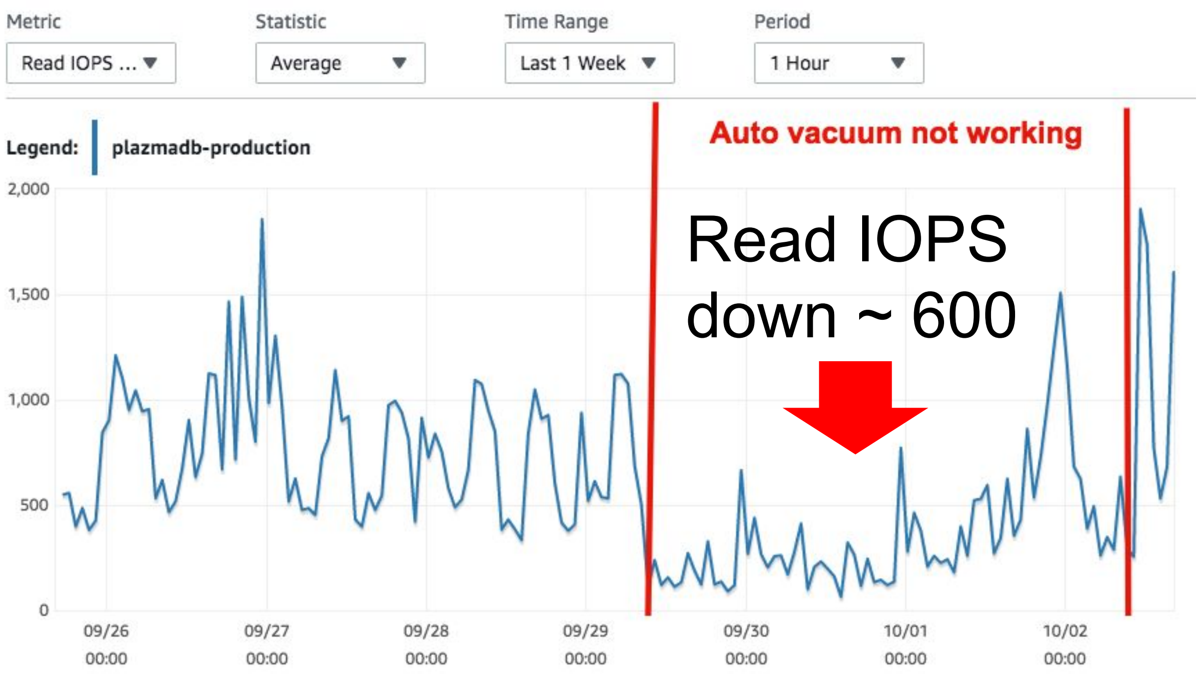
- Vacuum read cold data
  - Especially Vacuum Freeze does force full scan (before postgres 9.6)
- IO related Vacuum parameters

autovacuum_vacuum_cost_limit	400
autovacuum_vacuum_cost_delay	10ms
vacuum_cost_page_miss	10
vacuum_cost_page_dirty	20

- Max read IOPS = (max IO per vacuum) * (vacuum invoked per sec)
  = (400/10) * (1000/10) = 4000

Vacuum is tax of PostgreSQL

Someday auto vacuum stopped accidentally ..



To reduce IO

- Improve cache hit ratio
  - More RAM -> scale up / sharding
  - Improve locality
    -> Partitioning by data set ID: include more relevant rows in a (postgres) page
- Vacuum
  - Avoid full scan of vacuum freeze by using postgres 9.6 or newer

Summary

- IO seems to be a bottleneck
  - Cache miss increases IO dramatically
  - When throughput was increased, we need to pay more tax (vacuum)

- Now we understand what is likely to be a bottleneck
  - Better prioritization of action items
  - Predictable PlazmaDB performance
    Don’t need to worry about spike and increasing demand :)

Thank You!
Danke!
Merci!
谢谢!
Gracias!
Kiitos!

arm
TREASURE DATA