

MySQL Performance Optimization and Troubleshooting with PMM

Peter Zaitsev, CEO, Percona

Percona University Kiev

11 November 2017



Few words about Percona Monitoring and Management (PMM)

Free, Open Source database troubleshooting and performance optimization platform for MySQL and MongoDB

100% Free and Open Source

Based on Industry Leading Technology

Roll your own in and out of the Cloud



PERCONA
Monitoring and Management

Exploring Percona Monitoring and Management

You should be able to install PMM in 15 minutes or less

- <http://bit.ly/InstallPMM>

Would like to follow along in the demo ?

- <https://pmmdemo.percona.com>

In the Presentation

Practical approach to
deal with some of the
common MySQL
Issues

Assumptions

You're looking to Have your MySQL Queries Run Faster

You want to troubleshoot sudden MySQL Performance Problem

You want to find way to run more efficiently (use less Resources)

How to Look at MySQL Performance

Query Based Approach

- All the users (developers) care is how quickly their queries perform

Resource Based Approach

- Queries use resources. Slow Performance often caused by resource constraints

Primary Resources

CPU

Disk IO

Memory

Network

Low Resource Usage + Poor Performance

Contention

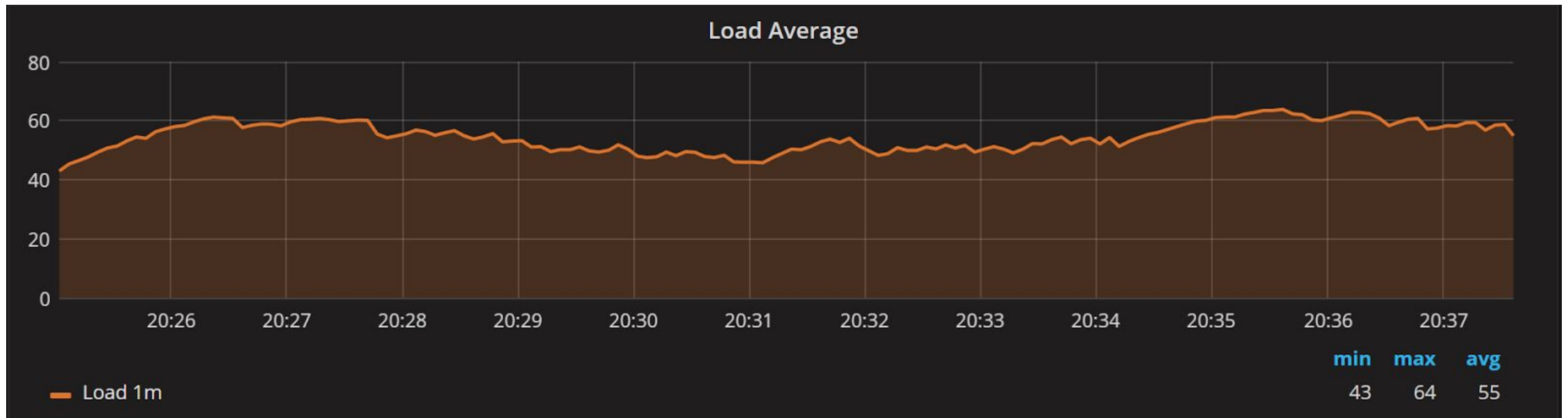
- Table Locks/Row Level Locks
- Locking/Latching in MySQL and Kernel

Mixed Resource Usage

- Single worker spending 33% on CPU
- 33% Waiting on Disk
- 33% on Network
- Will not be seen as directly constrained by any resource

Load Average

- What can you tell me about server load ?



Problems with Load Average

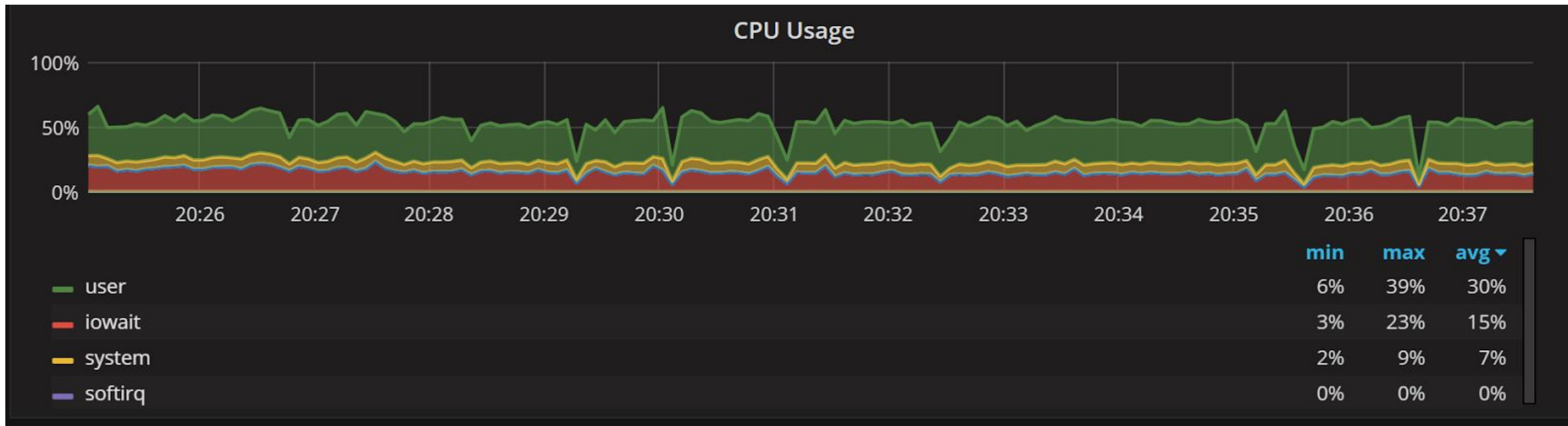
Mixes CPU and IO resource usage (on Linux)

Is not normalized for number of CPU cores available

Does not keep into account Queue Depth Needed for optimal storage performance

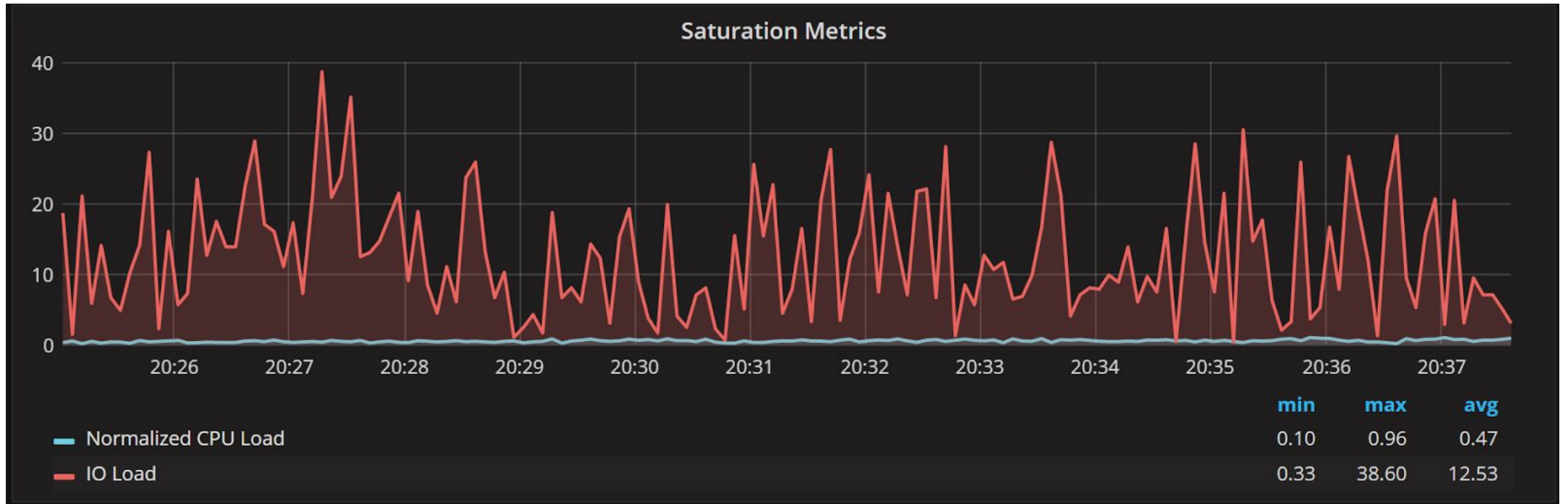
CPU Usage

- Can observe overall or per core
- Matching Load Average in the previous screen



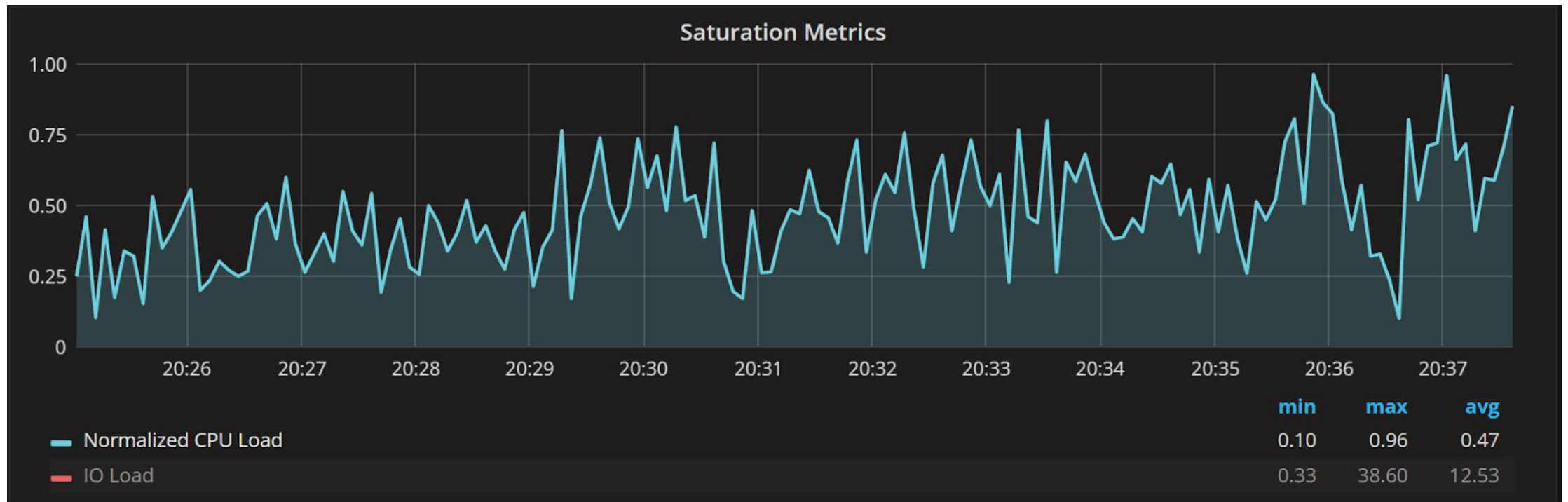
Saturation Metrics

- Good to understand where waits are happening
- IO Load is not normalized



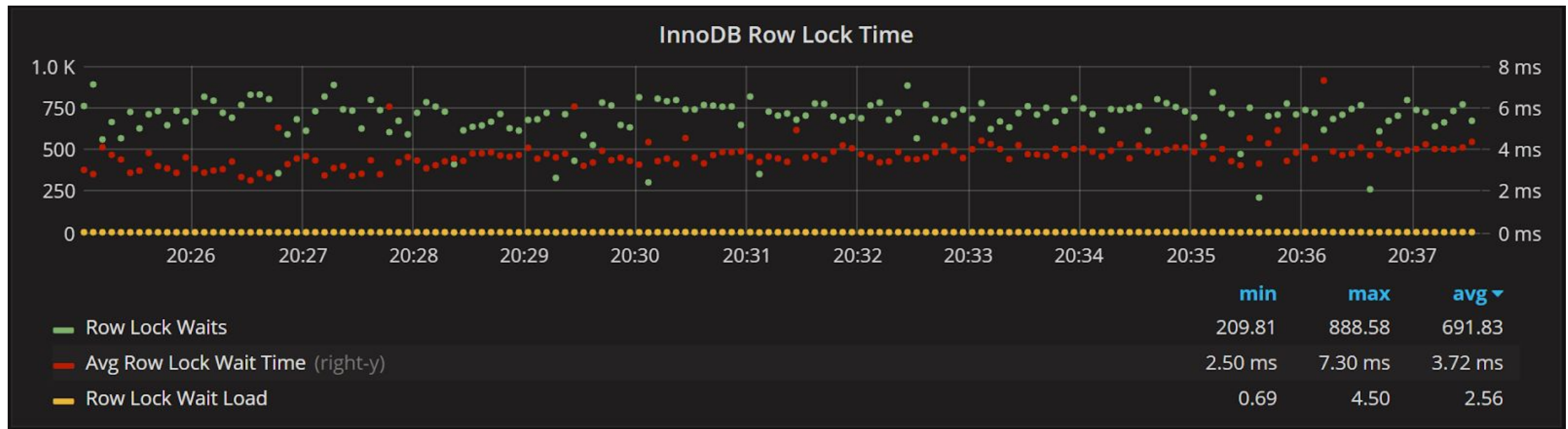
Looking at CPU Saturation Separately

- Can normalize CPU Saturation based on number of threads



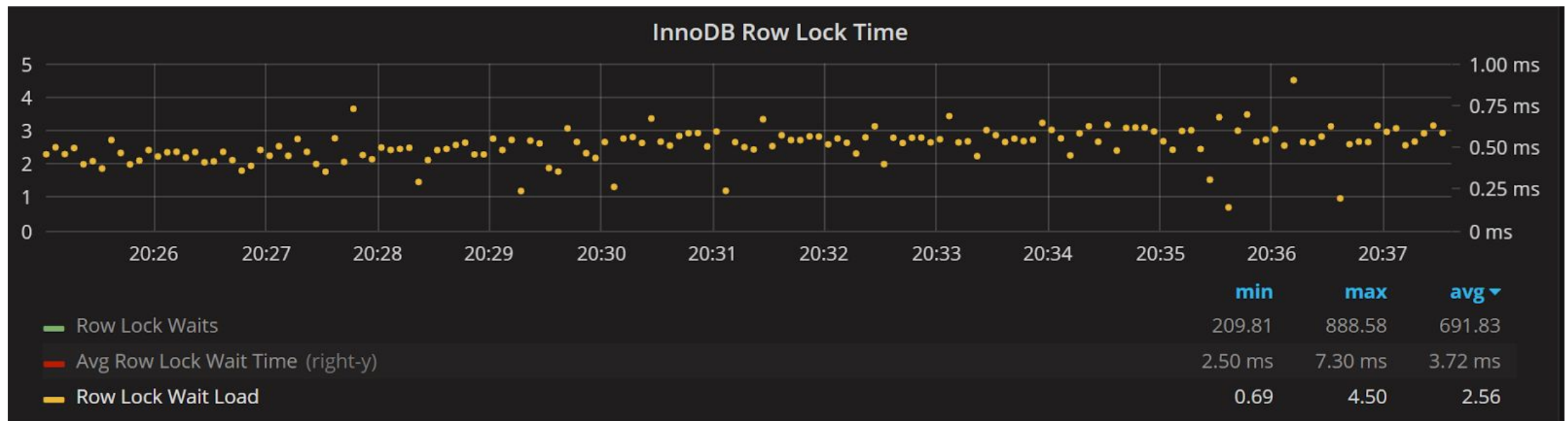
Row Locks – Logical Contention

- Row Locks are often declared by transaction semantics
- But more transactions underway also mean more locks



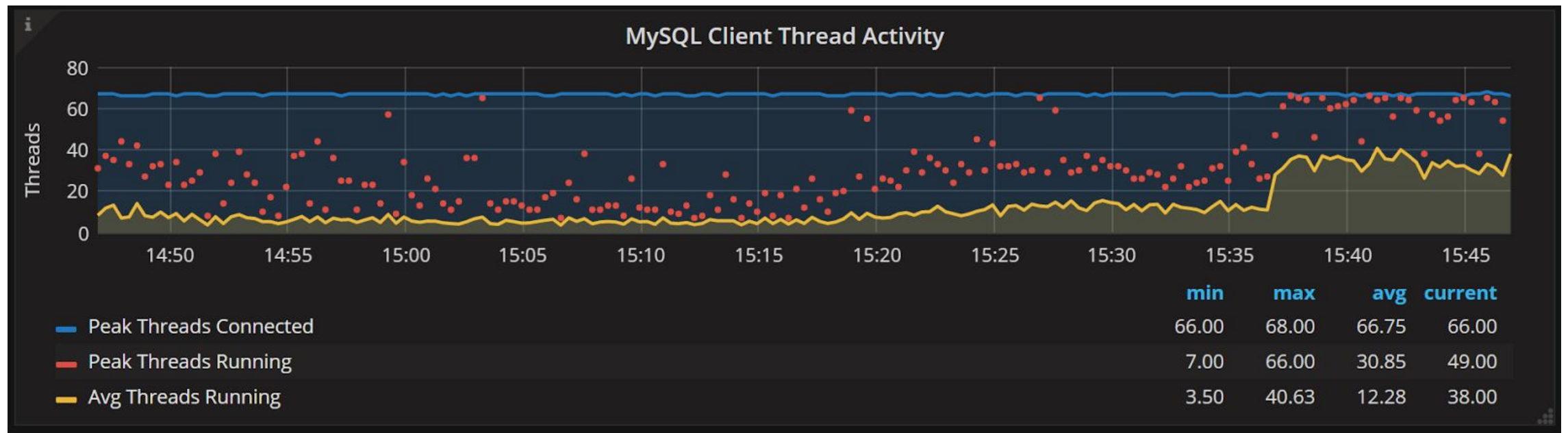
Zooming in on Row Locks Wait Load

- How many MySQL Connections are Blocked because of Row Level Lock Waits



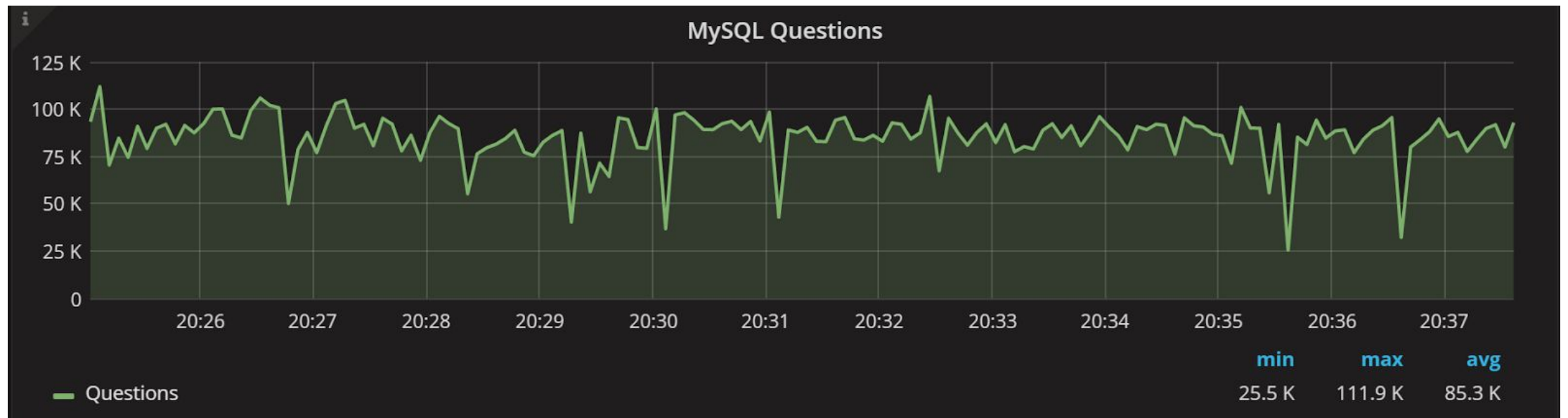
“Load at MySQL Side”

- “threads_running” - MySQL is busy handling query
- CPU ? Disk ? Row Level Locks ? Need to dig deeper



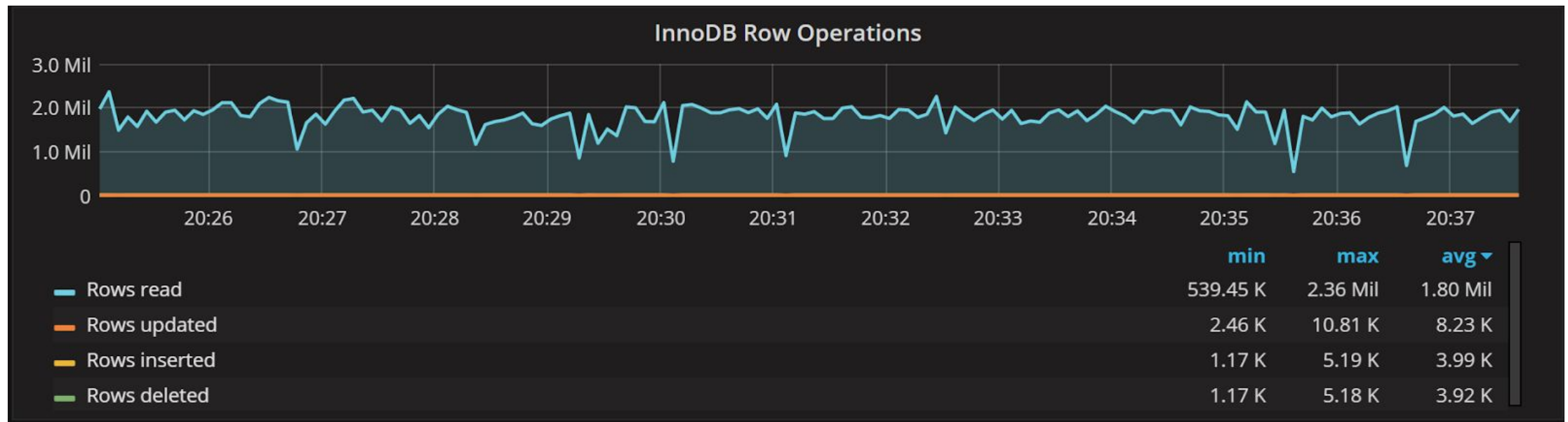
MySQL Questions – Inflow of Queries

- Are we serving more queries or less queries ?
- Any spikes or dips ?



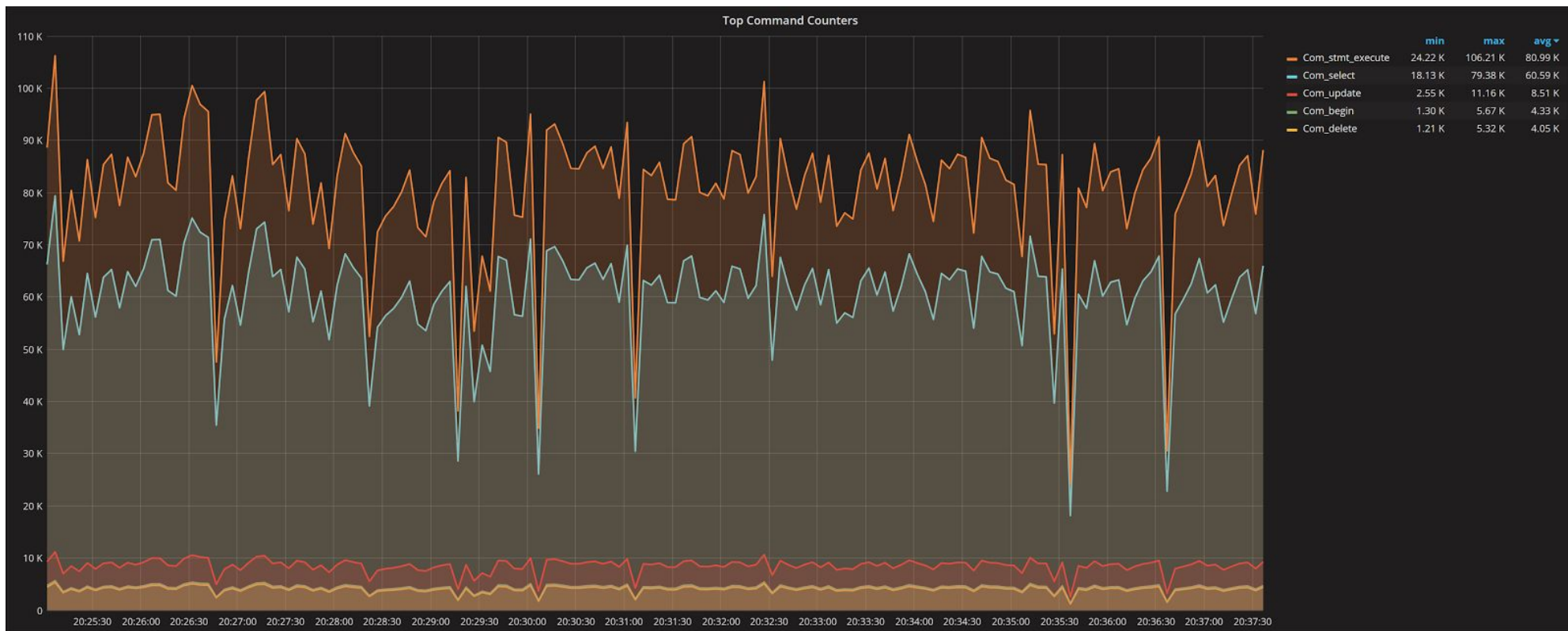
InnoDB Rows – Actual Work Being Done

- Better number to think re system capacity
- Not all rows are created equal, but more equal than queries



Commands – What kind of operations

- Note if prepared statements are used MySQL is “double counting”



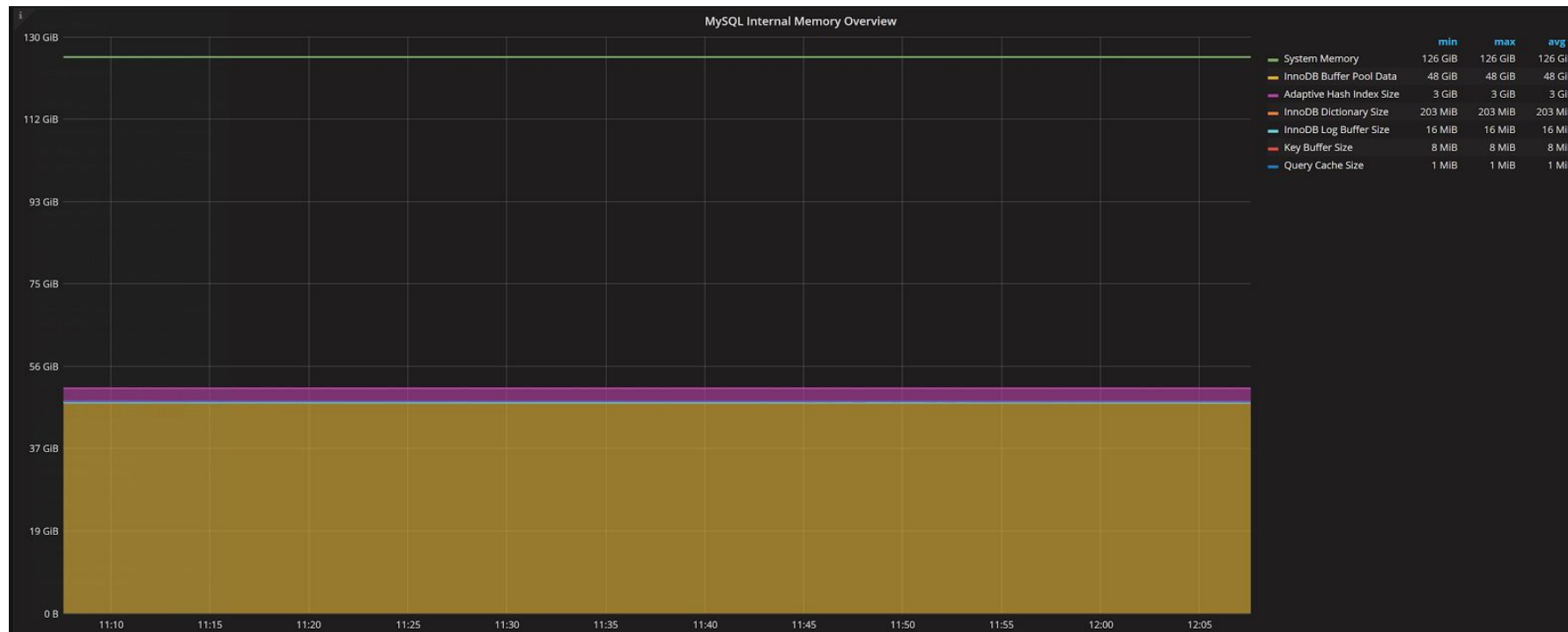
MySQL “Handlers” low lever row access

- Works for all storage engines
- Gives more details on access type
- Mixes Temporary Tables and Non-Temporary tables together



Memory usage by MySQL

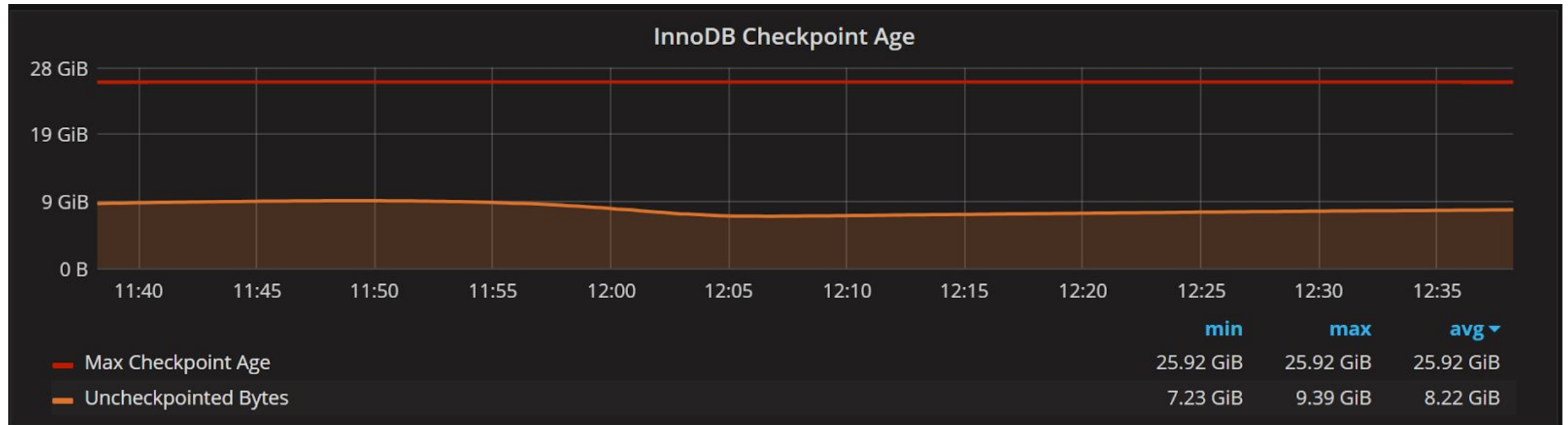
Leave some memory available for OS Cache and other needs



Innodb in Depth

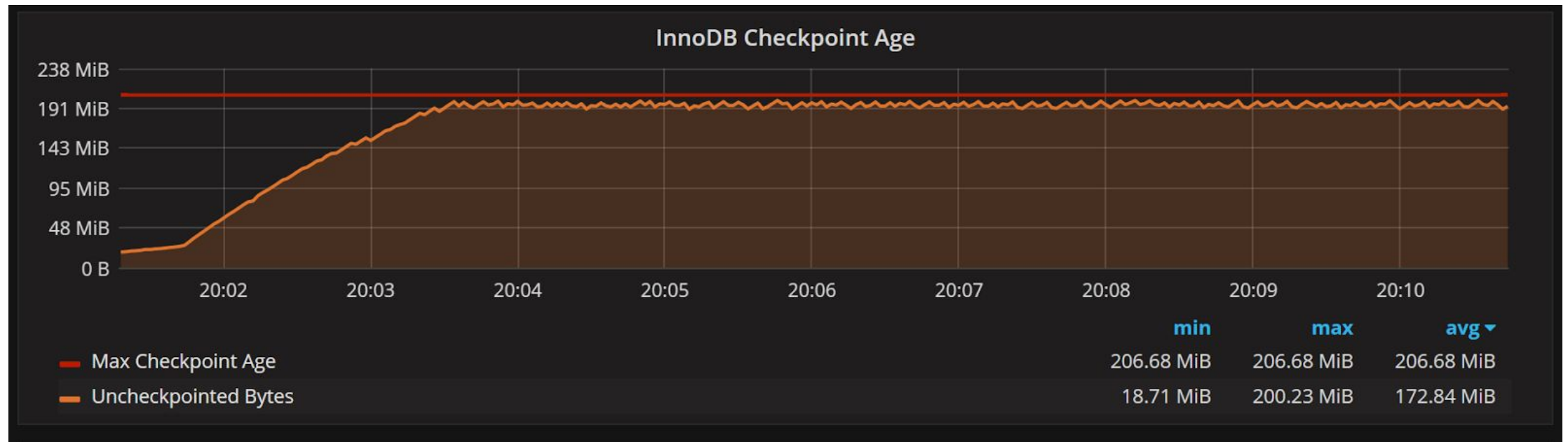
InnoDB Checkpointing

- The log file size is good enough as Uncheckpointed bytes are fraction of log file size



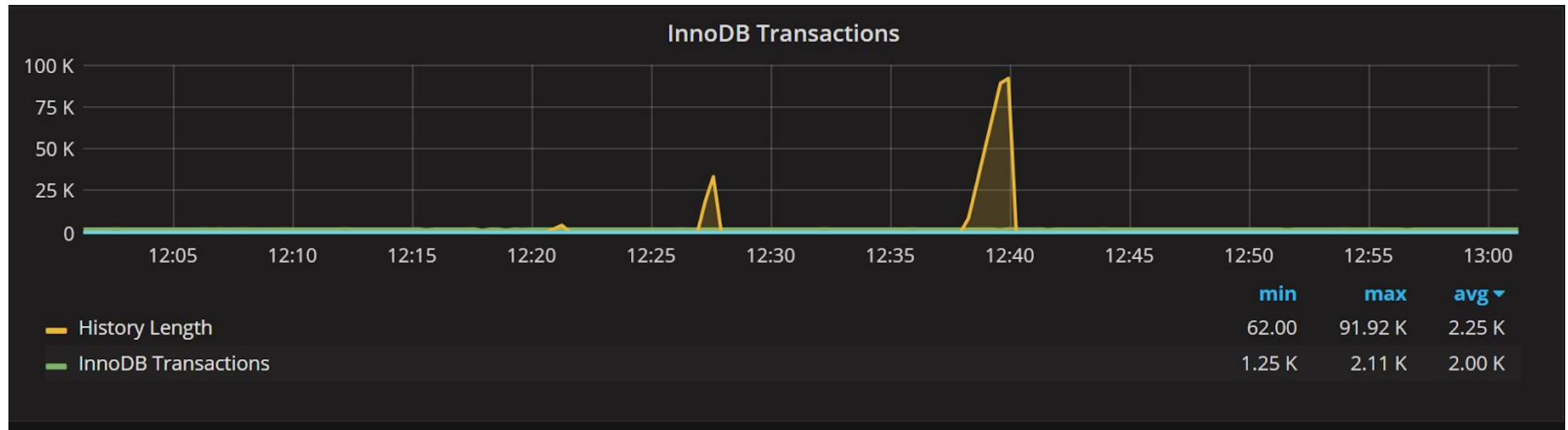
Innodb Checkpointing

- Very Close – Innodb Log File Size too small for optimal performance



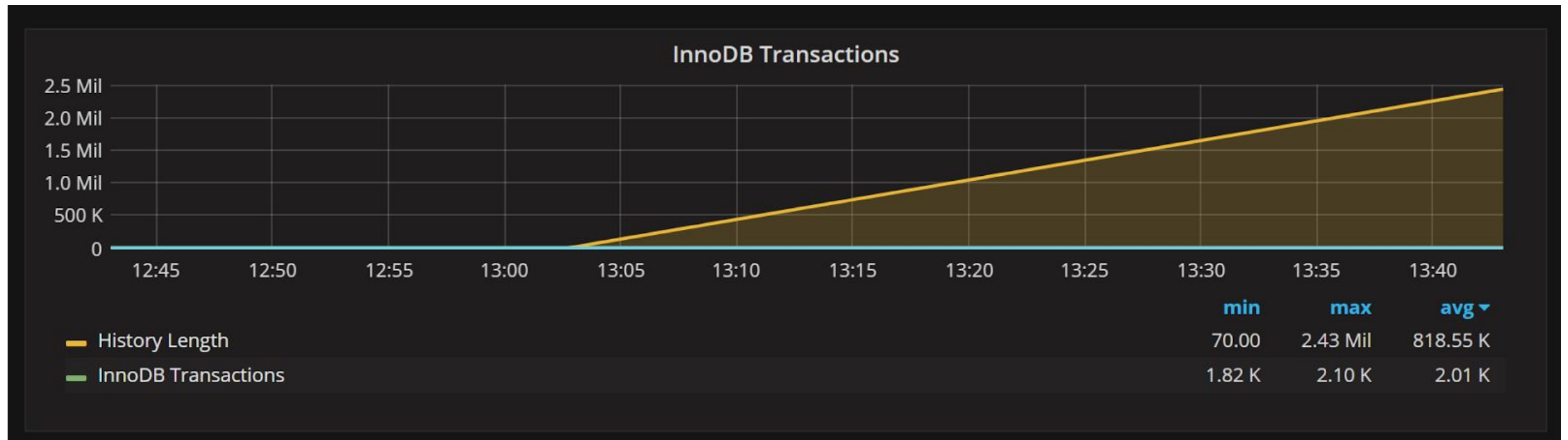
InnoDB Transaction History - not yet Purged Transactions

- Short term spikes are normal if some longer transactions are ran on the system



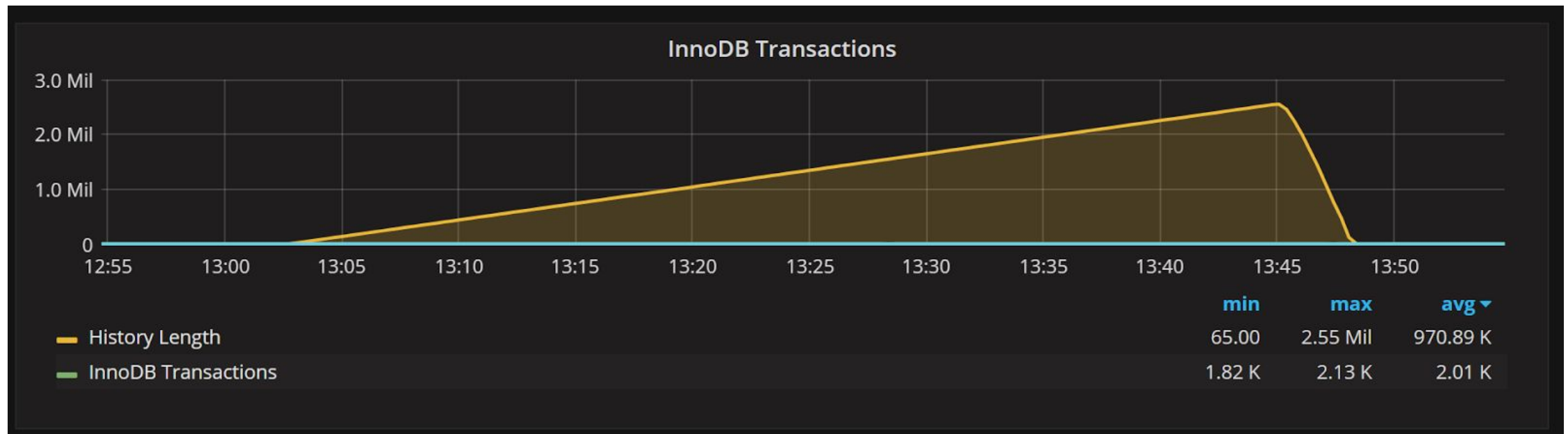
InnoDB Transaction History

- Growth over long period of time without long queries in the processlist
- Often identifies orphaned transactions (left open)



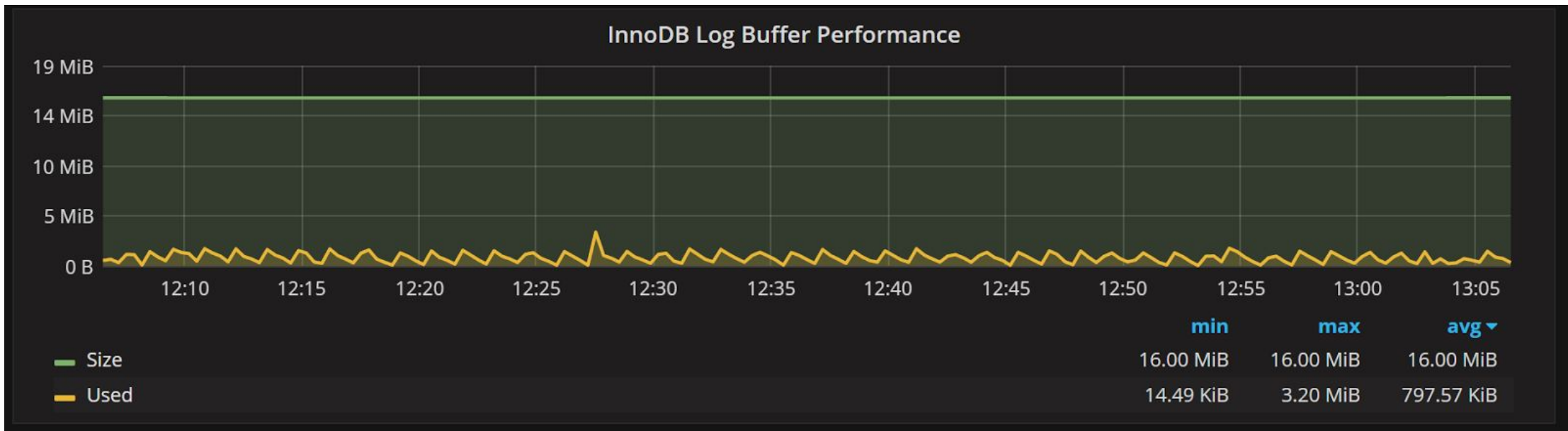
Transaction History Recovery

- If Backlog is resolved quickly it is great
- If not you may be close to the limit of purge subsystem



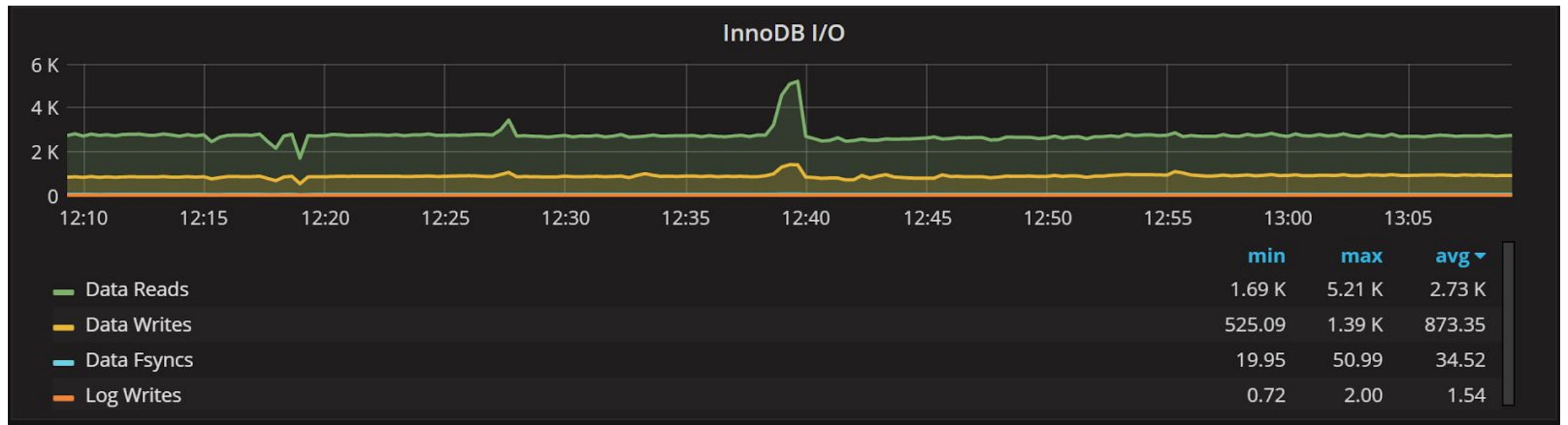
Is your InnoDB Log Buffer Large Enough?

- You will be surprised to see how little log buffer space InnoDB needs



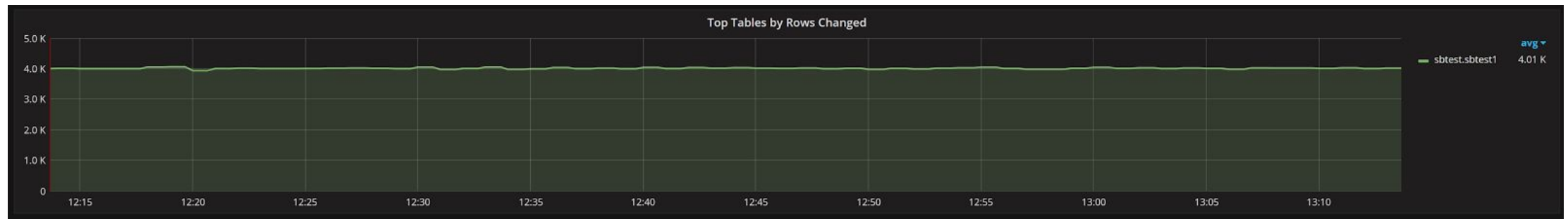
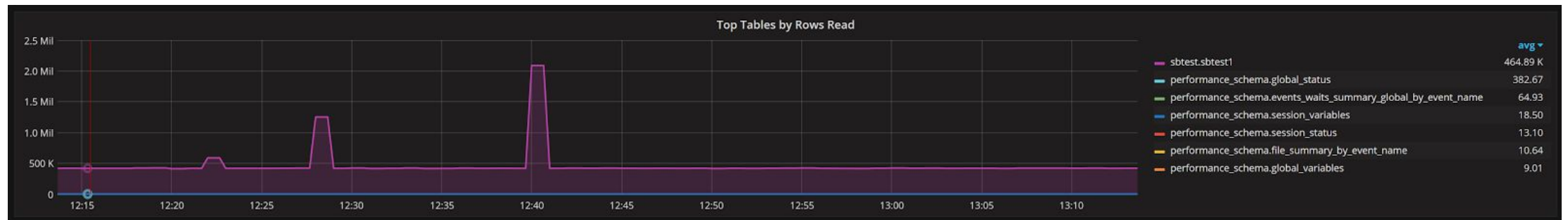
Innodb IO

- Will often roughly match disk IO
- Allows to see the writes vs fsyncs



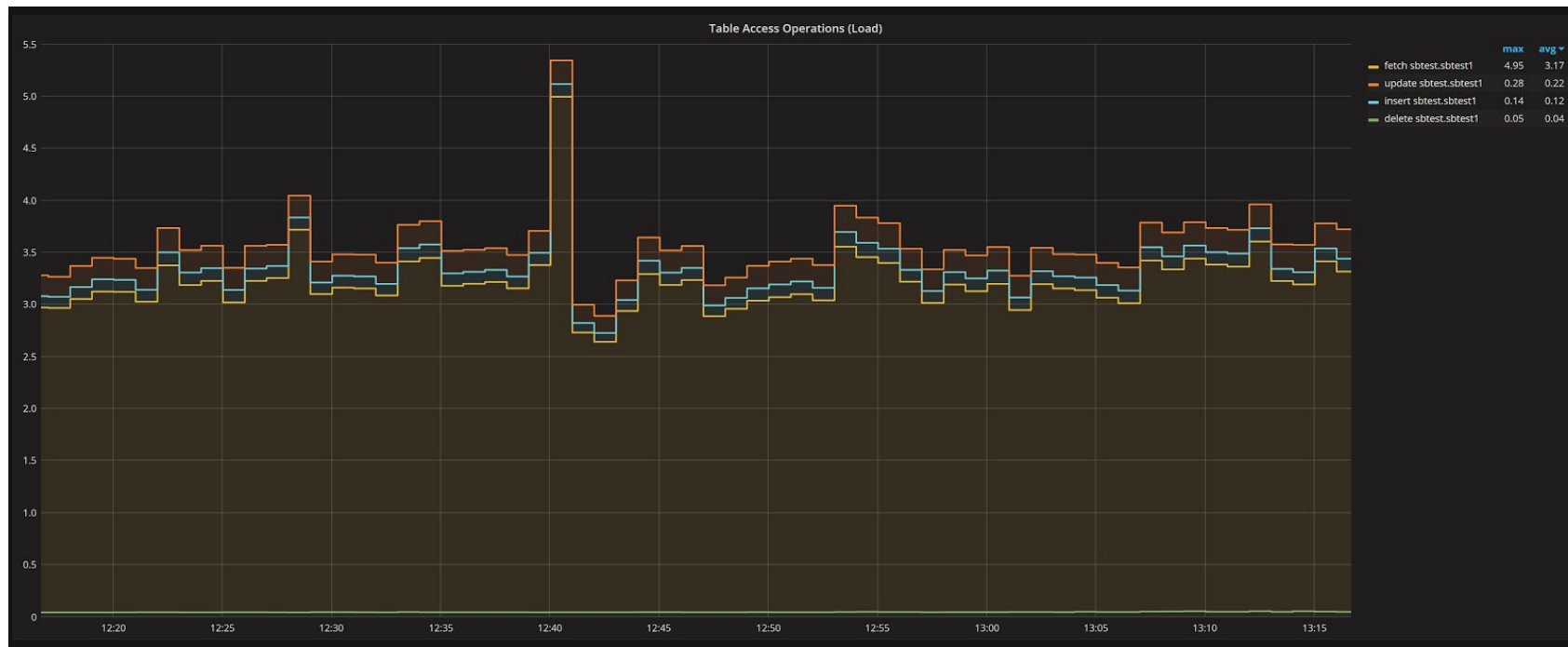
Hot Tables

- It is often helpful to know what tables are getting most Reads
- And Writes



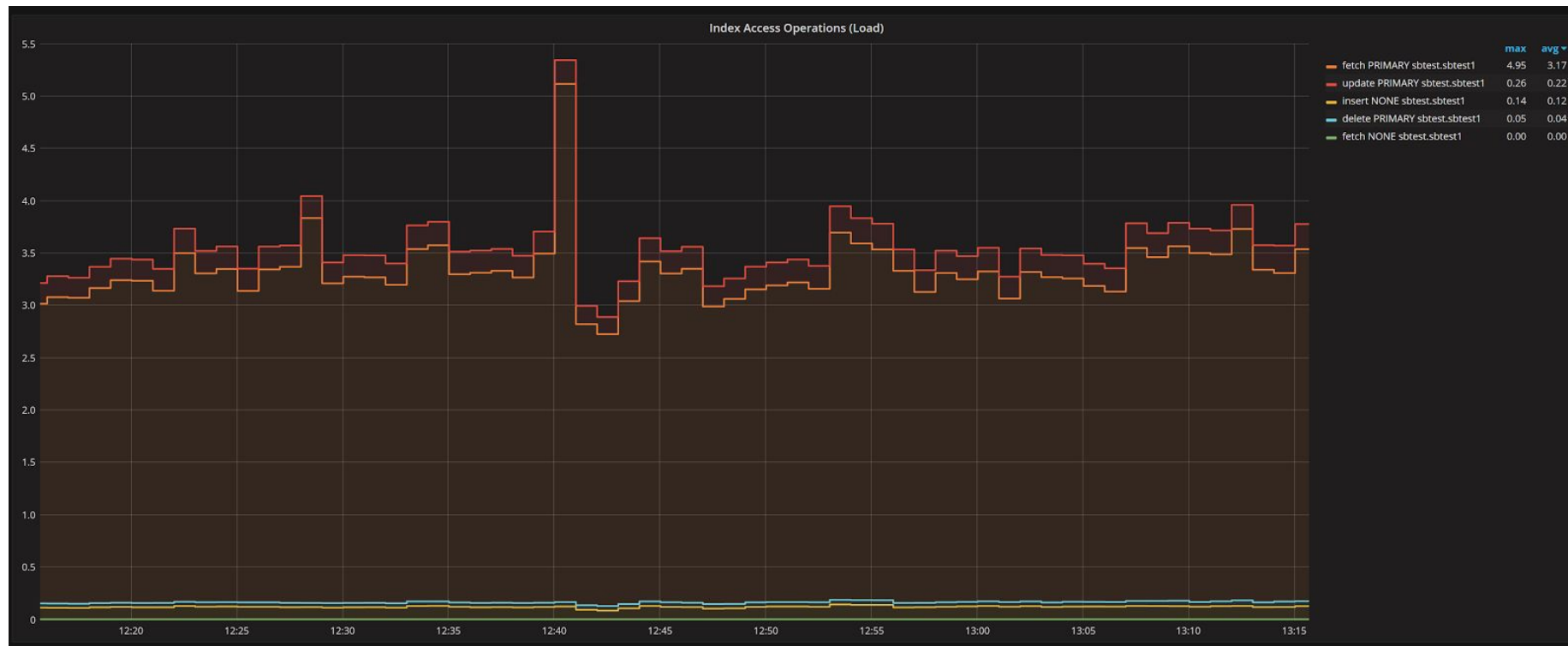
Hot Tables through Performance Schema

- Even more details available in Performance Schema
- Load is a better measure of actual cost than number of events



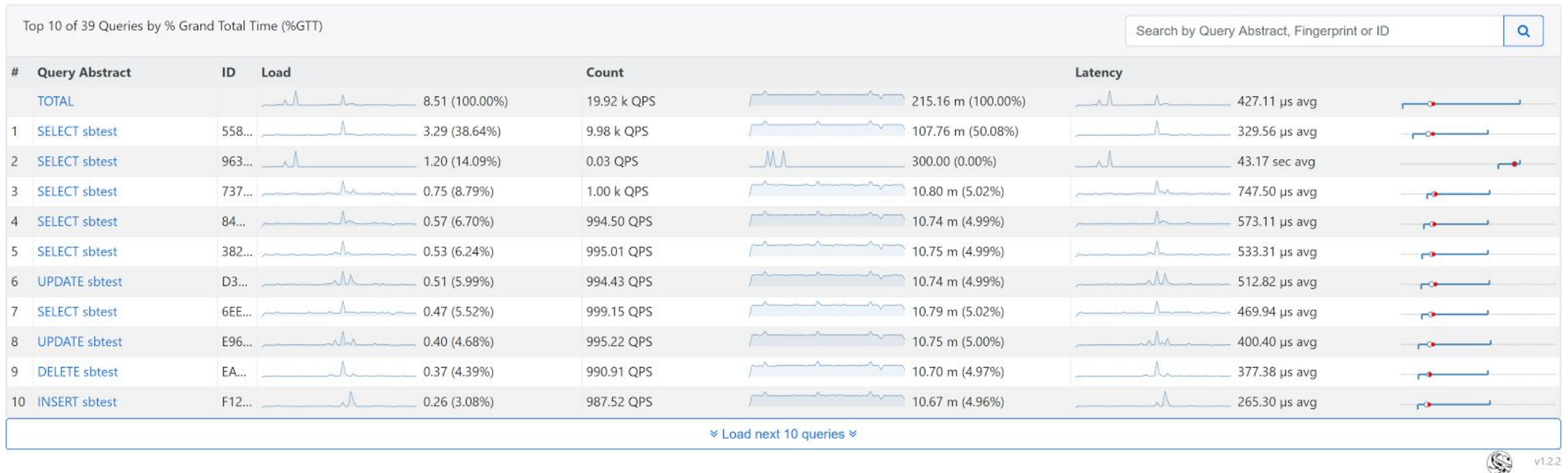
Most Active Indexes

- See through which index queries access tables



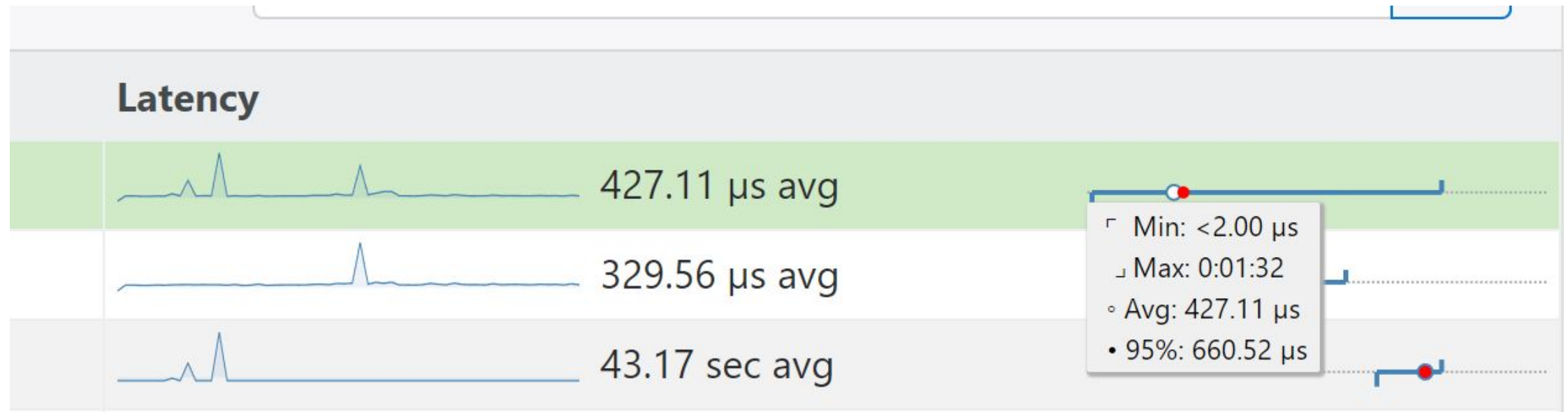
What about Queries causing the most load?

- Can examine through Query Analytics application



Latency Details Explored

- Not enough to look at Average Latency



What are Top Queries ?

Queries Sorted by their “Load”




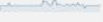



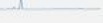
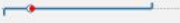


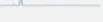
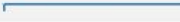



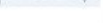

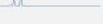
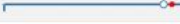

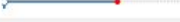
Query ran 10 times over second each time taking 0.2 sec will be load 2

Not making a difference between queries “causing” the load or just impacted by it

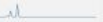
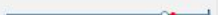




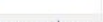
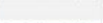
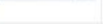

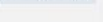


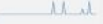

Whole Server Summary #1

- Server Summary Gives a good idea what is going on query wise

Server Summary

Metrics				
Metrics	Rate/Sec	Sum	Per Query Stats	
Query Count	20.03 k (per sec) 	216.37 m		
Query Time	8.55 load 	1 days, 1:38:36	426.45 µs avg	
Lock Time	0.52 (avg load) 	1:34:25 6.14% of query time	26.18 µs avg	
InnoDB Row Lock Wait	0.04 (avg load) 	0:07:25 0.00% of total 0.54% of query time	<2.29 µs avg	
InnoDB IO Read Wait	2.60 (avg load) 	7:47:22 33.76% of query time	143.96 µs avg	
InnoDB Read Ops	4.30 k (per sec) 	46.42 m	0.08 avg	
InnoDB Read Bytes	70.42 MB (per sec) 	760.54 GB 16.38 KB avg io size	3.91 KB avg	
InnoDB Distinct Pages	-	-	3.12 avg	
Rows Sent	312.39 k (per sec) 	3.37 b	15.08 avg	
Bytes Sent	40.24 MB (per sec) 	434.59 GB 128.81 Bytes bytes/row	2.01 KB avg	
Rows Examined	2.20 m (per sec) 	23.74 b 7.04 per row sent	109.03 avg	
Rows Affected	3.98 k (per sec) 	42.99 m	0.00 avg	

Whole Server Summary #2

Rows Examined	2.20 m (per sec) 	23.74 b 7.04 per row sent	109.03 avg 
Rows Affected	3.98 k (per sec) 	42.99 m	0.00 avg 
External Sorts (Filesort)	2.01 k (per sec) 	21.67 m 10.01% of queries	-
Cartesian Products (Full Joins)	0.03 (per sec) 	300.00 <0.01% of queries	-
Full Table Scans	3.03 (per sec) 	32.70 k 0.02% of queries	-
Queries Requiring Tmp Table In Memory	1.01 k (per sec) 	10.89 m 5.03% of queries	-
Number of Tmp table in Memory	1.02 k (per sec) 	11.05 m 101.49% per query with tmp table	0.00 avg 
Queries Requiring Tmp Table on Disk	0.19 (per sec) 	2.00 k <0.01% of queries	-
Number of Tmp Tables on Disk	5.45 (per sec) 	58.90 k 2945.00% per query with disk tmp table	0.00 avg 
Total Size of Tmp Tables	127.90 MB (per sec) 	1.38 TB 6.38 KB per query	6.38 KB avg 

 v1.2.2

Specific Query – Update Query

- Significant part of response time comes from row level lock waits

UPDATE sbtest

D30AD7E3079ABCE7

Metrics				
Metrics	Rate/Sec		Sum	Per Query Stats
Query Count	1.00 k (per sec)		10.80 m 4.99% of total	
Query Time	0.51 load		1:32:19 6.00% of total	513.38 µs avg
Lock Time	0.06 (avg load)		0:11:00 11.65% of total 11.91% of query time	61.12 µs avg
Innodb Row Lock Wait	0.01 (avg load)		0:02:16 30.47% of total 2.44% of query time	12.51 µs avg
Innodb IO Read Wait	0.10 (avg load)		0:18:01 3.85% of total 19.51% of query time	100.16 µs avg
Innodb Read Ops	121.37 (per sec)		1.31 m 2.82% of total	0.00 avg
Innodb Read Bytes	1.99 MB (per sec)		21.48 GB 2.82% of total 16.38 KB avg io size	1.99 KB avg
Innodb Distinct Pages	-	-	-	7.15 avg
Bytes Sent	52.10 KB (per sec)		562.64 MB 0.13% of total 0.00 Bytes bytes/row	52.00 Bytes avg
Rows Examined	997.01 (per sec)		10.77 m 0.05% of total 0.00 per row sent	0.16 avg
Rows Affected	997.01 (per sec)		10.77 m 25.05% of total	0.16 avg

Expensive SELECT Query

- Examining lots of rows per each row sent

SELECT sbtest

9632B579B0F37132

Metrics			
Metrics	Rate/Sec	Sum	Per Query Stats
Query Count	<0.01 (per sec)	1.00 <0.01% of total	
Query Time	<0.01 load	33.93 sec 0.15% of total	33.93 sec avg
Lock Time	<0.01 (avg load)	120.00 µs <0.01% of total <0.01% of query time	120.00 µs avg
Innodb IO Read Wait	<0.01 (avg load)	6.00 sec 0.07% of total 17.67% of query time	6.00 sec avg
Innodb Read Ops	5.14 (per sec)	18.50 k 0.20% of total	18.50 k avg
Innodb Read Bytes	84.21 KB (per sec)	303.15 MB 0.20% of total 16.38 KB avg io size	303.15 MB avg
Innodb Distinct Pages	-	-	65.33 k avg
Rows Sent	<0.01 (per sec)	1.00 <0.01% of total	1.00 avg
Bytes Sent	0.02 (per sec)	67.00 Bytes <0.01% of total 67.00 Bytes bytes/row	67.00 Bytes avg
Rows Examined	13.89 k (per sec)	50.00 m 1.96% of total 50.00 m per row sent	50.00 m avg

Check Query Example

- Expensive Query not poorly optimized one

▼Example

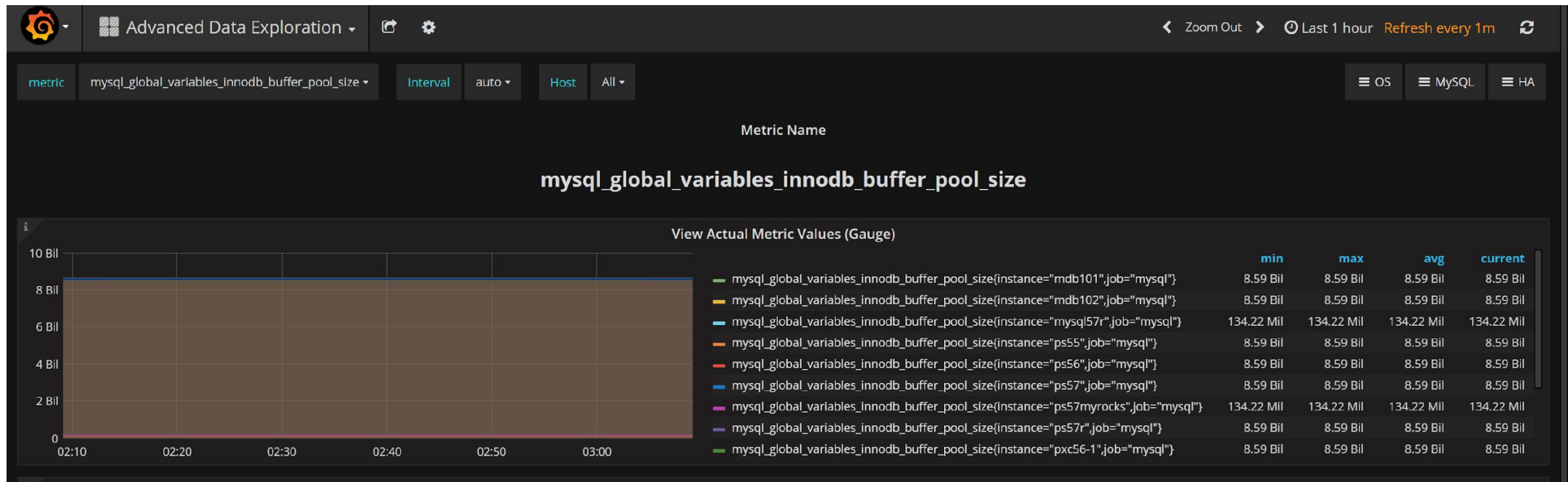
```
SELECT avg(k)
FROM sbtest1
WHERE id<1000000000
```


Explain and JSON Explain

```
▼JSON
{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "78703653.66"
    },
    "table": {
      "table_name": "sbtest1",
      "access_type": "range",
      "possible_keys": [
        "PRIMARY"
      ],
      "key": "PRIMARY",
      "used_key_parts": [
        "id"
      ],
      "key_length": "4",
      "rows_examined_per_scan": 193712066,
      "rows_produced_per_join": 193712066,
      "filtered": "100.00",
      "cost_info": {
        "read_cost": "39961240.47",
        "eval_cost": "38742413.20",
        "prefix_cost": "78703653.67",
        "data_read_per_join": "34G"
      },
      "used_columns": [
        "id",
        "k"
      ],
      "attached_condition": "(`sbtest`.`sbtest1`.`id` < 100000000)"
    }
  }
}
```

Explore Any Captured Metrics

- Standard Dashboards are only tip of the iceberg
- You can also use Prometheus directly



Lets Look at Couple of Case Studies

Impact Of Durability ?

Running sysbench with **rate=1000** to inject 1000 transactions every second

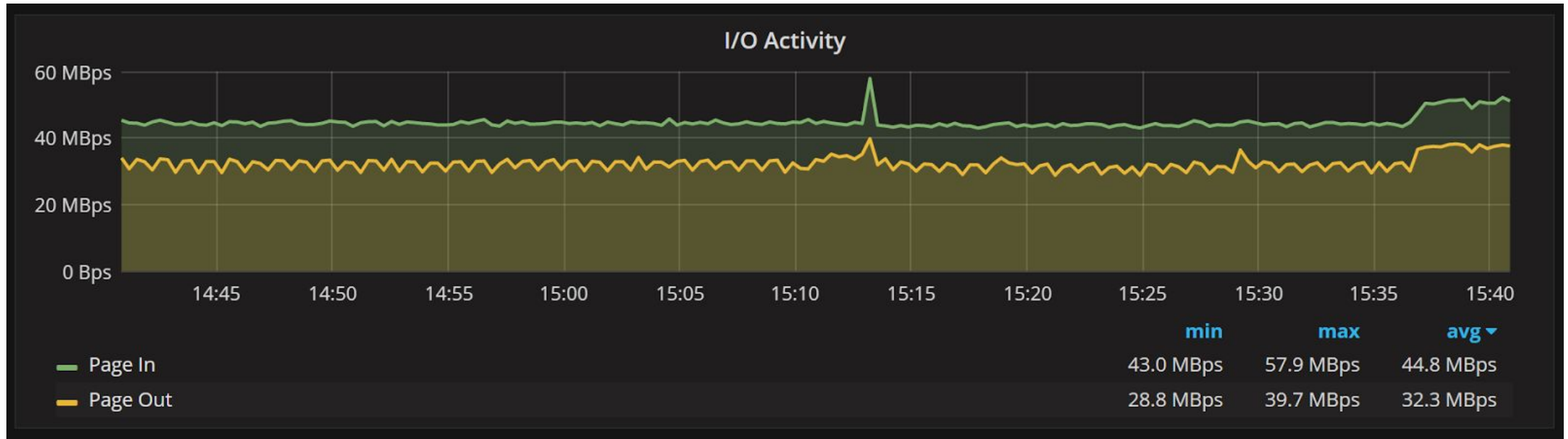
System can handle workloads with both settings

System previously running with **sync_binlog=0** and **innodb_flush_log_at_trx_commit=0**

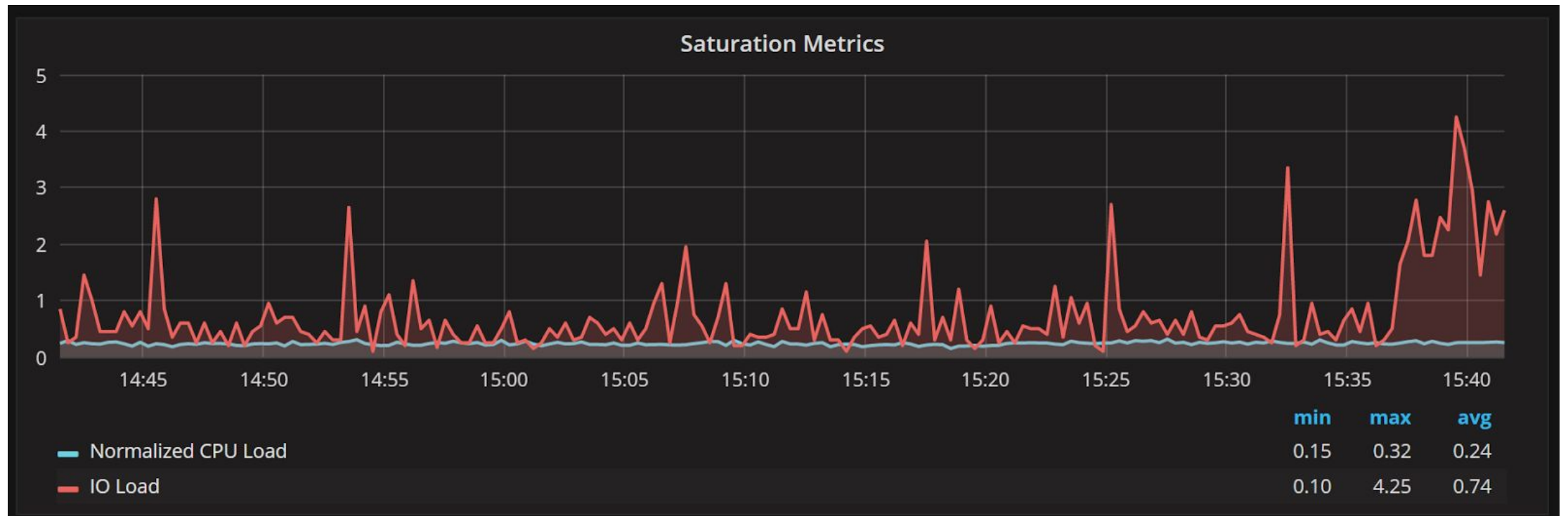
Set them to **sync_binlog=1** and **innodb_flush_log_at_trx_commit=1**

IO Bandwidth

- IO Bandwidth is not significantly impacted

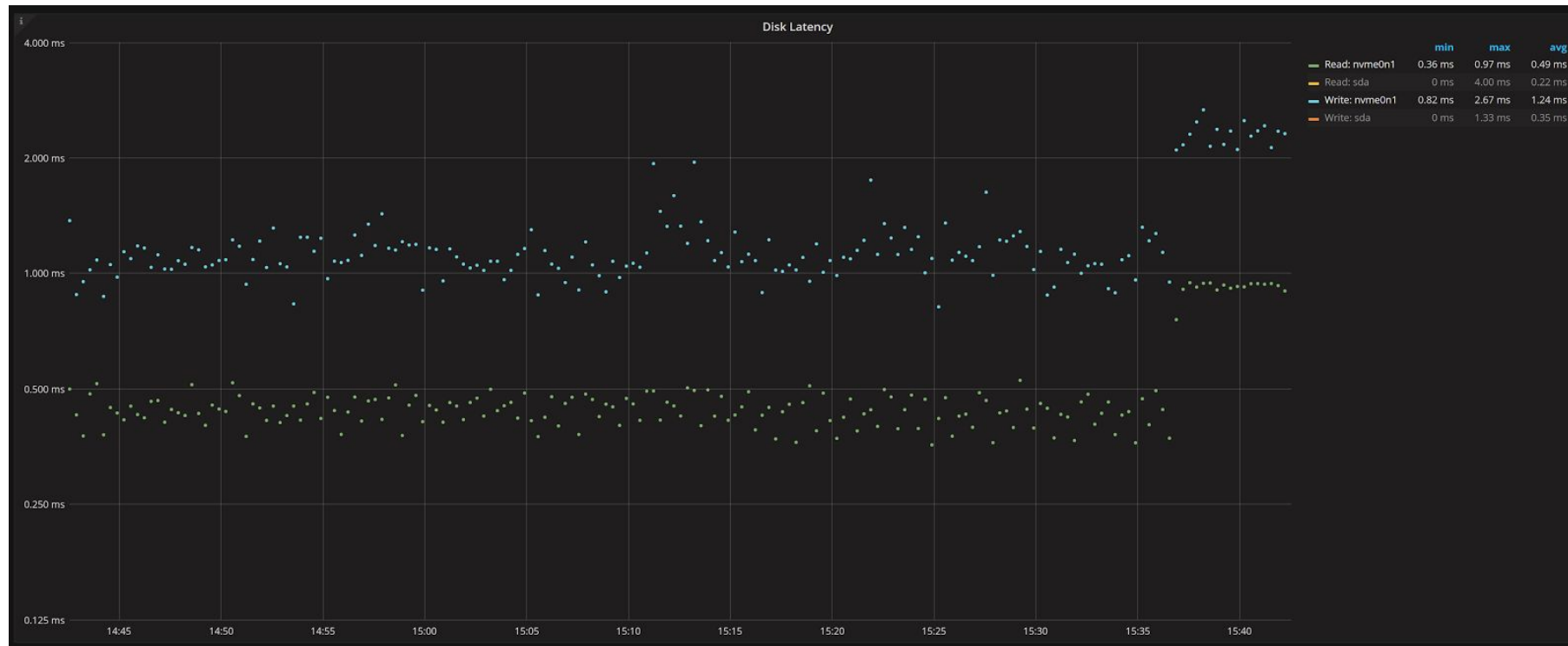


IO Saturation Jumps a Lot



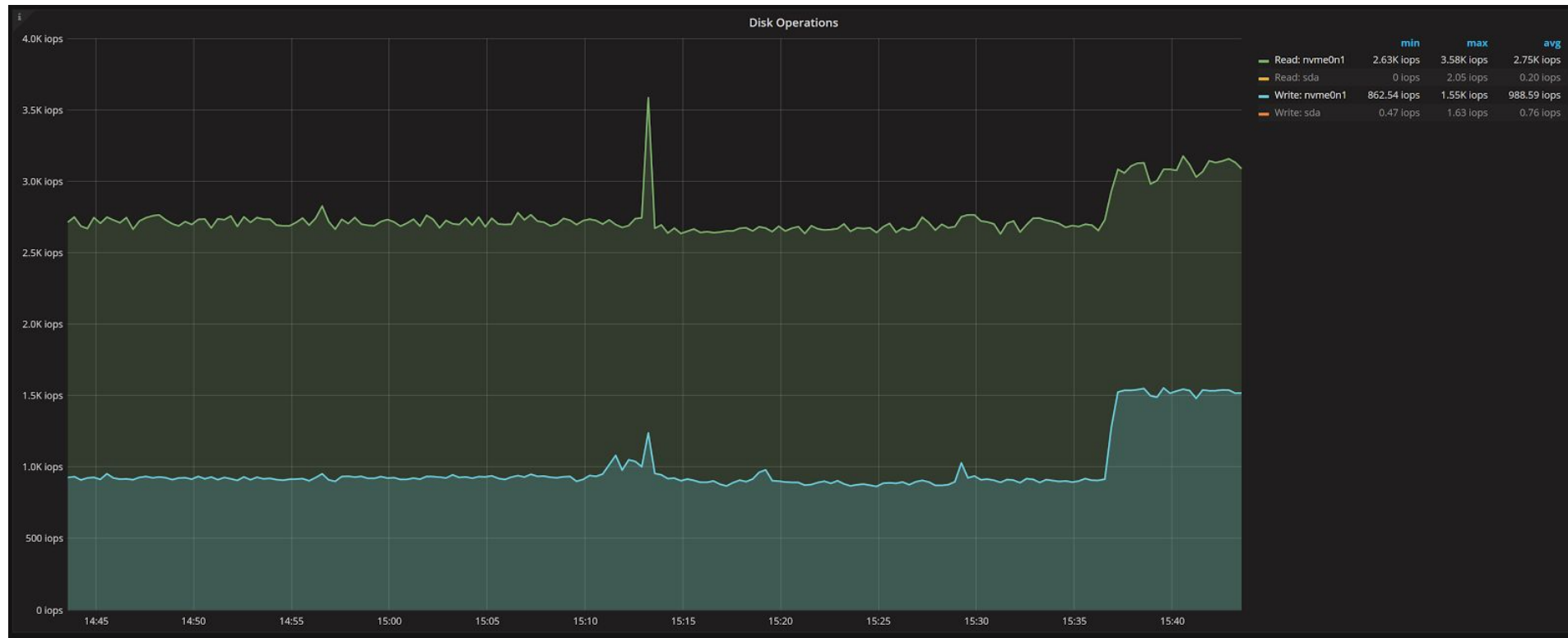
Read and Write Latencies are Impacted

- This SSD (Samsung 960 Pro) Does not like fsync() calls



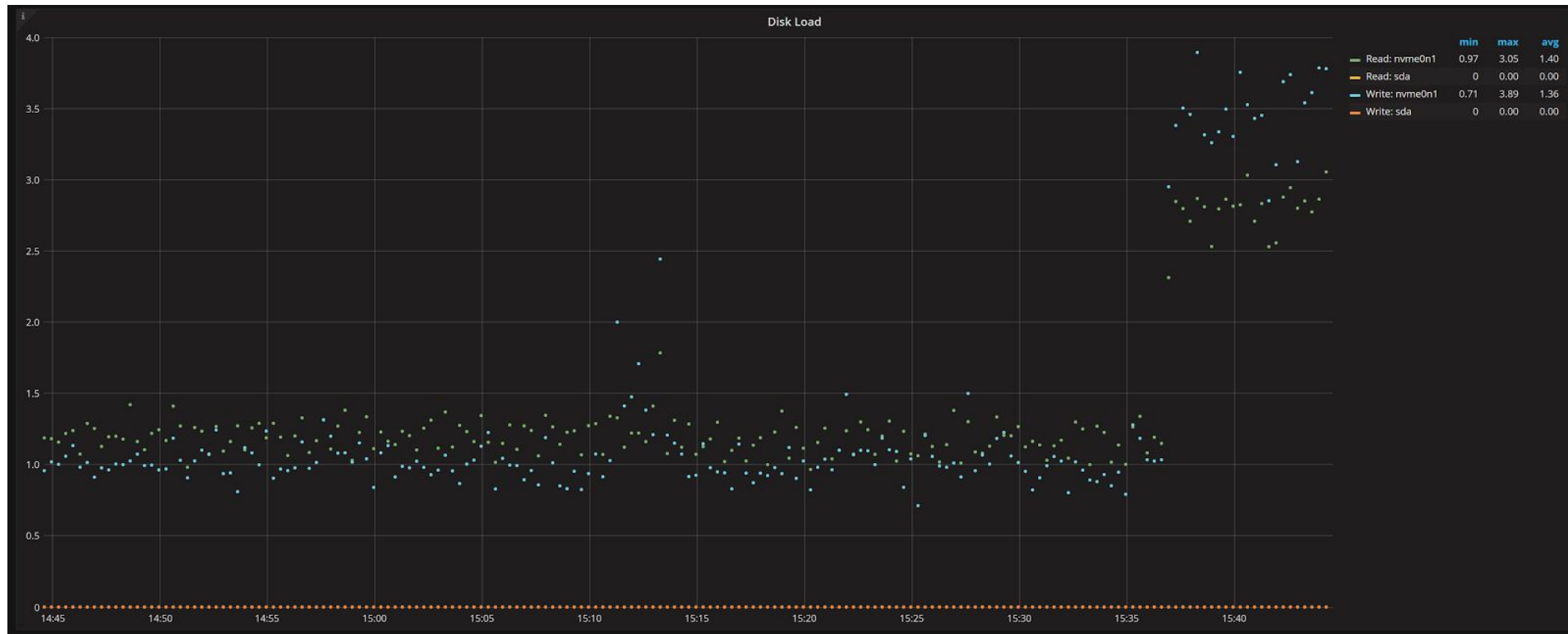
More Disk IO Operations

- Frequent Fsync() causes more writes of smaller size to storage



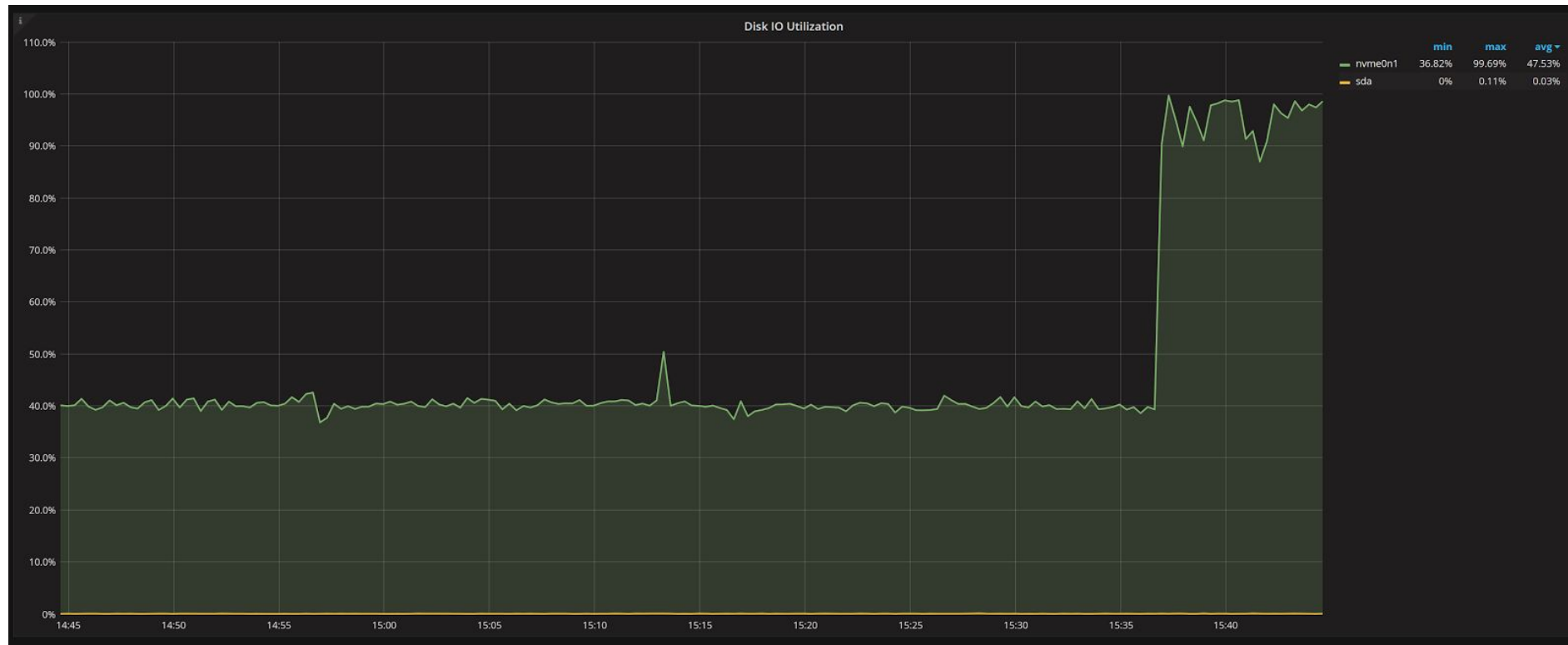
Increase In Disk IO Load

- IO Avg Latency Increase + More IOPs = Load Increase



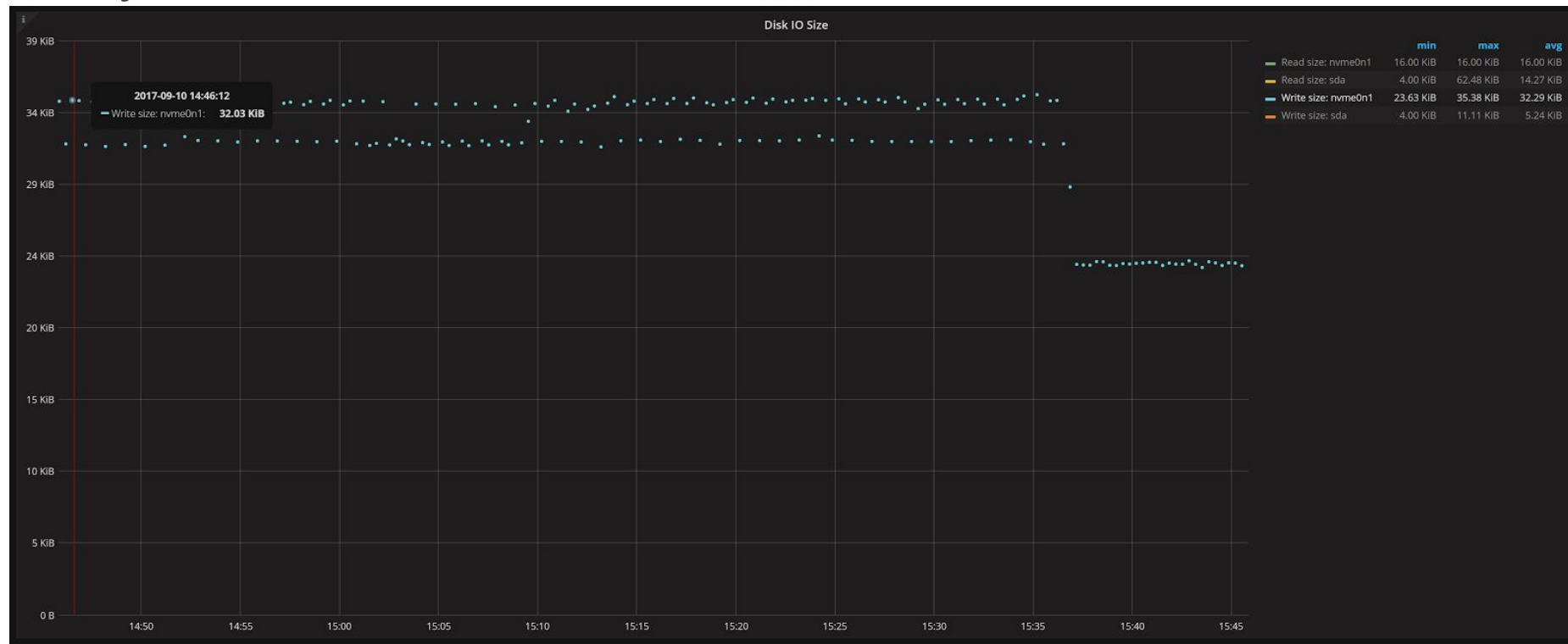
Disk IO Utilization jumps to 100%

- There is at least one disk IO Operation in flight all the time



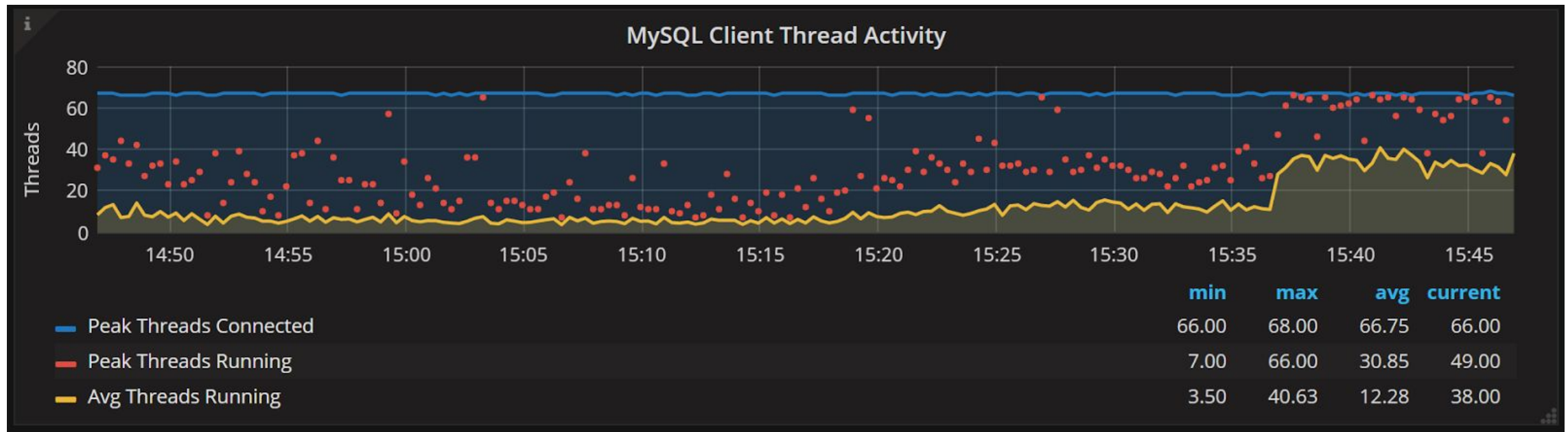
Average IO Size is down

- Large block writes to binlog and innodb transaction logs do not happen any more



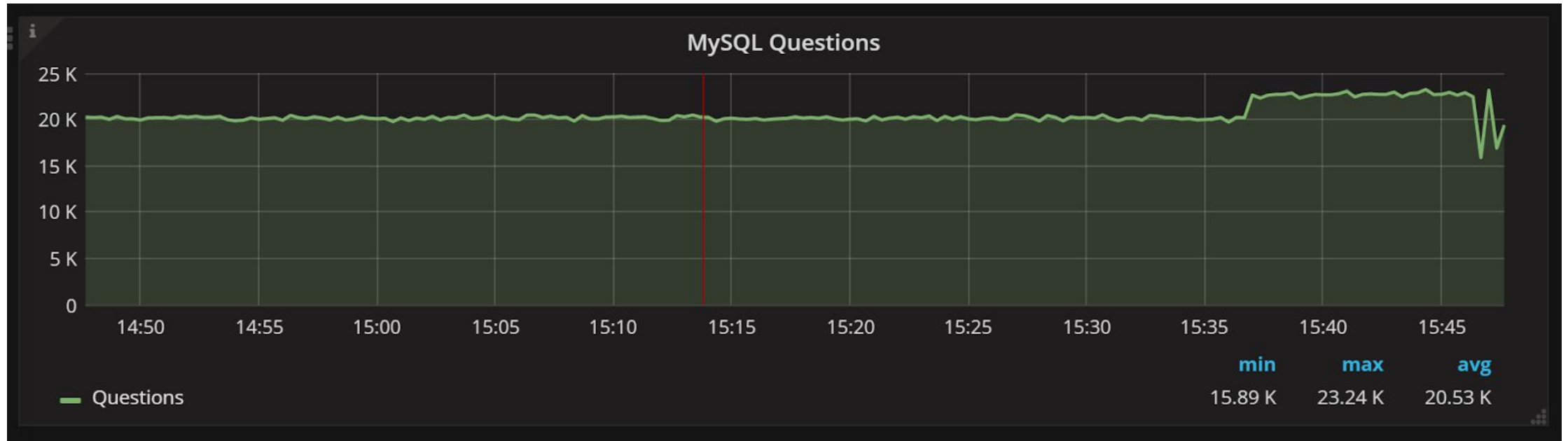
Number of Running Threads Impacted

- Need higher concurrency to be able to drive same number of queries/sec



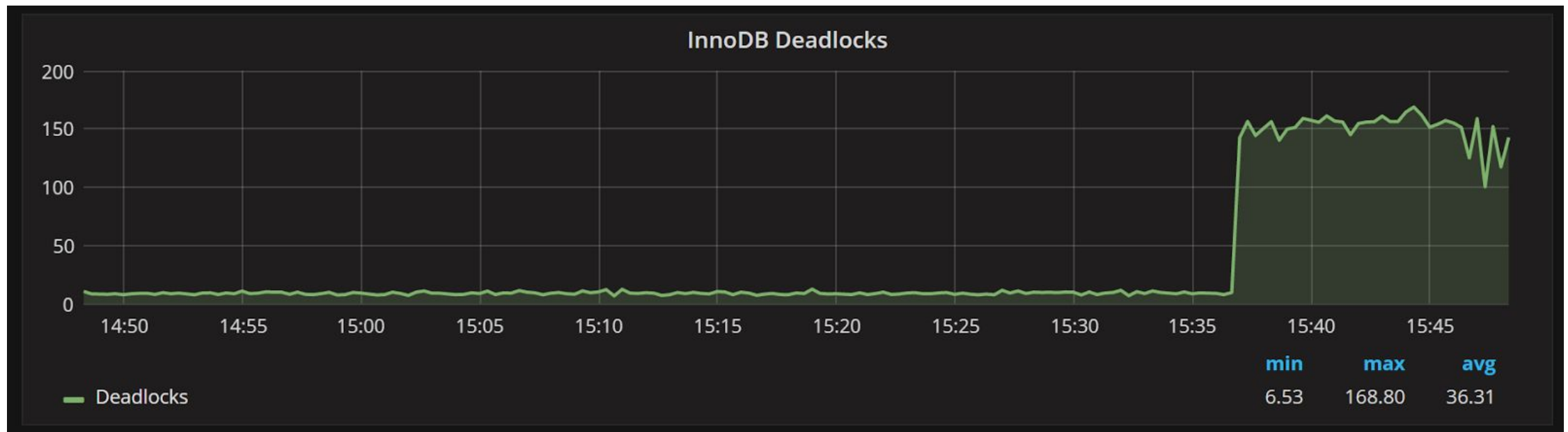
MySQL Questions

- Why does it increase with same inflow of transactions ?



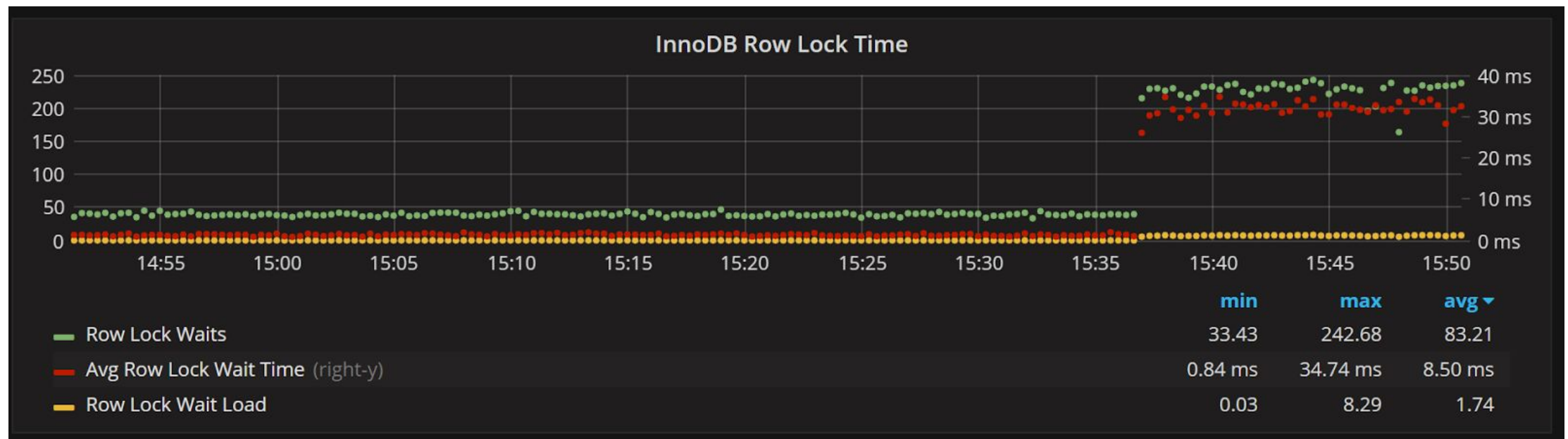
Because of Deadlocks

- Some transactions have to be retried due to deadlocks
- Your well designed system should behave the same

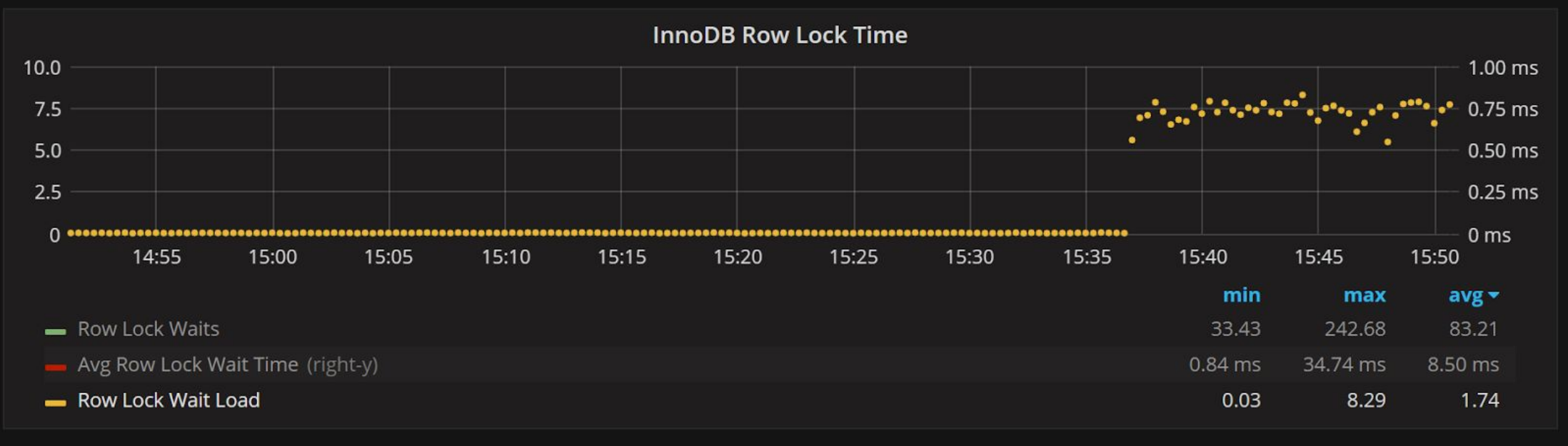


Higher Row Lock Time

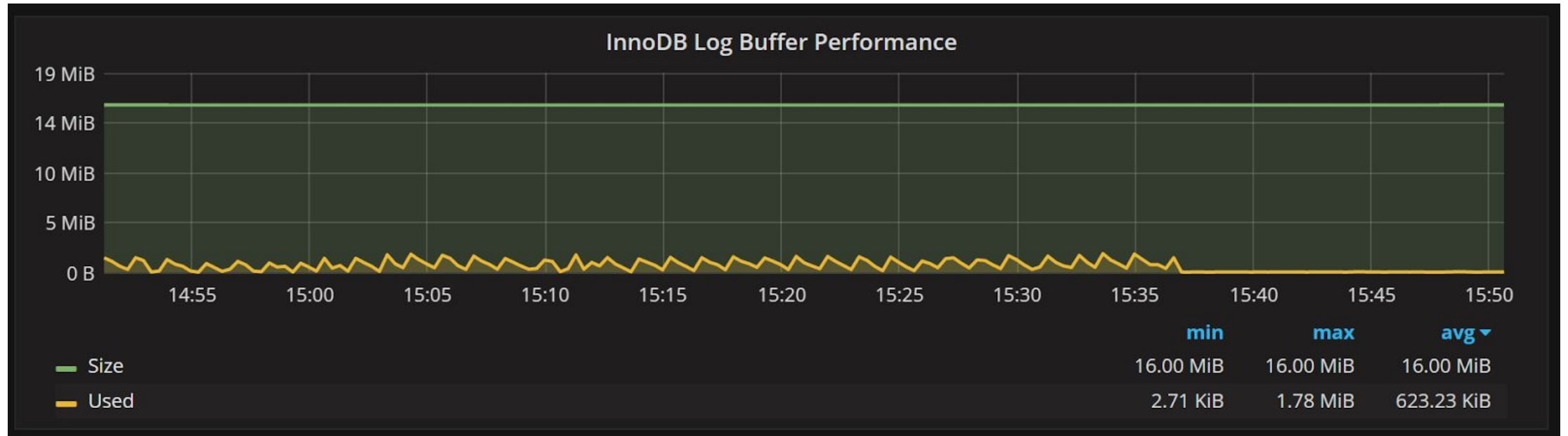
- Rows Locks can be only released after successful transaction commit
- Which now takes longer time due to number of fsync() calls



And Load Caused by Row Locks

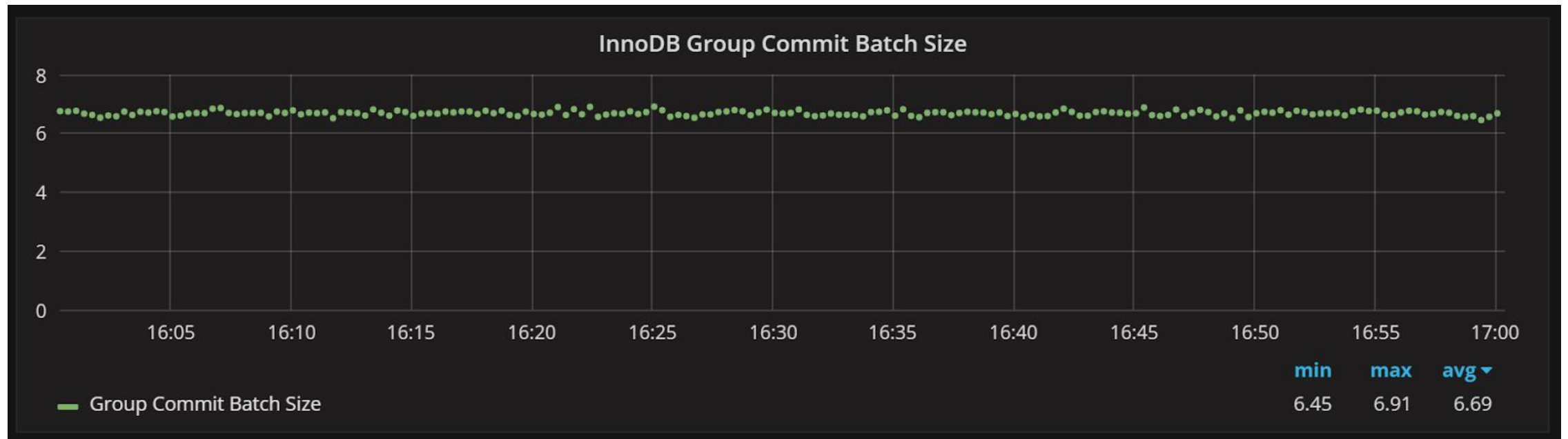


Log Buffer Used even less with durability on



Is Group Commit Working ?

- Do we rely on Group Commit for our workload


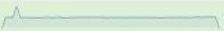



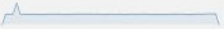

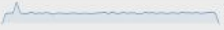





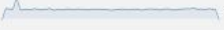






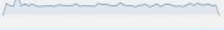
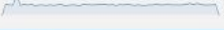


Top Queries Impacted

- Commit is now the highest load contributor

Top 10 of 34 Queries by % Grand Total Time (%GTT)

Search by Query Abstract, Fingerprint or ID

#	Query Abstract	ID	Load	Count	Latency
	TOTAL		 35.61 (100.00%)	22.37 k QPS	 80.54 m (100.00%)
1	COMMIT	813...	 17.64 (49.55%)	981.64 QPS	 3.53 m (4.39%)
2	SELECT sbtest	558...	 4.34 (12.19%)	11.46 k QPS	 41.25 m (51.21%)
3	UPDATE sbtest	E96...	 3.67 (10.31%)	1.08 k QPS	 3.88 m (4.82%)
4	DELETE sbtest	EA...	 3.48 (9.76%)	1.03 k QPS	 3.71 m (4.60%)
5	UPDATE sbtest	D3...	 3.46 (9.70%)	1.14 k QPS	 4.10 m (5.09%)
6	SELECT sbtest	737...	 0.87 (2.45%)	1.14 k QPS	 4.09 m (5.08%)
7	SELECT sbtest	84...	 0.68 (1.91%)	1.14 k QPS	 4.10 m (5.09%)
8	SELECT sbtest	382...	 0.61 (1.71%)	1.13 k QPS	 4.08 m (5.07%)
9	SELECT sbtest	6EE...	 0.60 (1.69%)	1.14 k QPS	 4.09 m (5.08%)
10	INSERT sbtest	F12...	 0.21 (0.59%)	983.33 QPS	 3.54 m (4.40%)

⌵ Load next 10 queries ⌵

Changing Buffer Pool Size

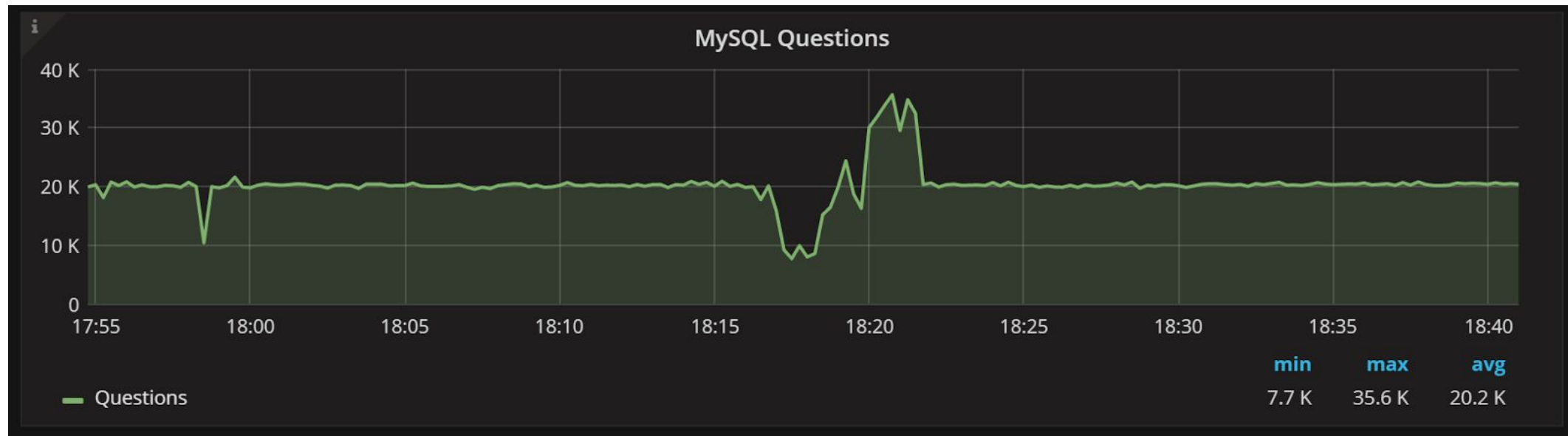
MySQL 5.7 Allows to change BP Online

- Changing buffer pool from 48GB to 4GB online

```
mysql> set global  
innodb_buffer_pool_size=4096*1024*1024;  
Query OK, 0 rows affected (0.00 sec)
```

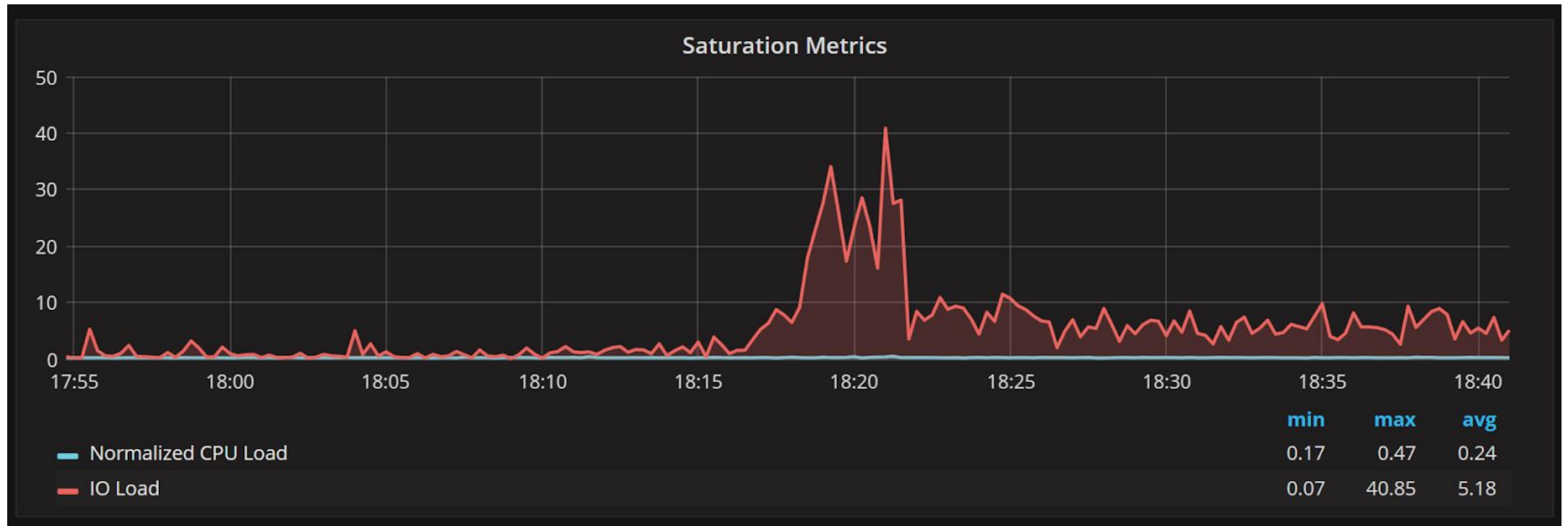
QPS Impact

- While resizing is ongoing capacity is limited – Queueing happens
- After resize completed backlog has to be worked off having higher number of queries



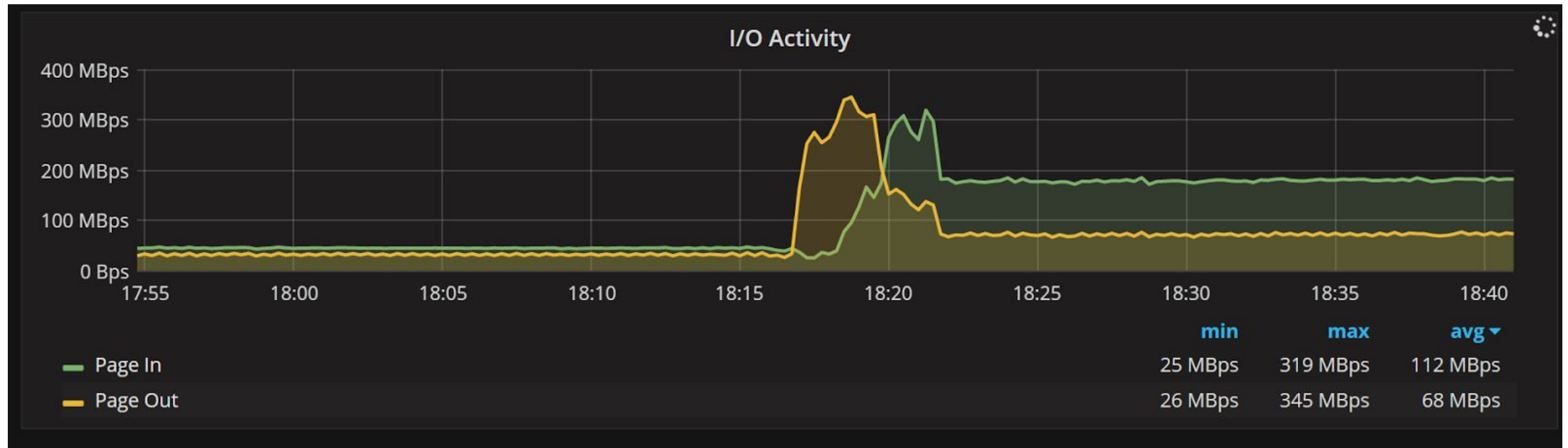
Saturation spike and when stabilizing on higher level

- Guess why the spike with lower QPS Level ?



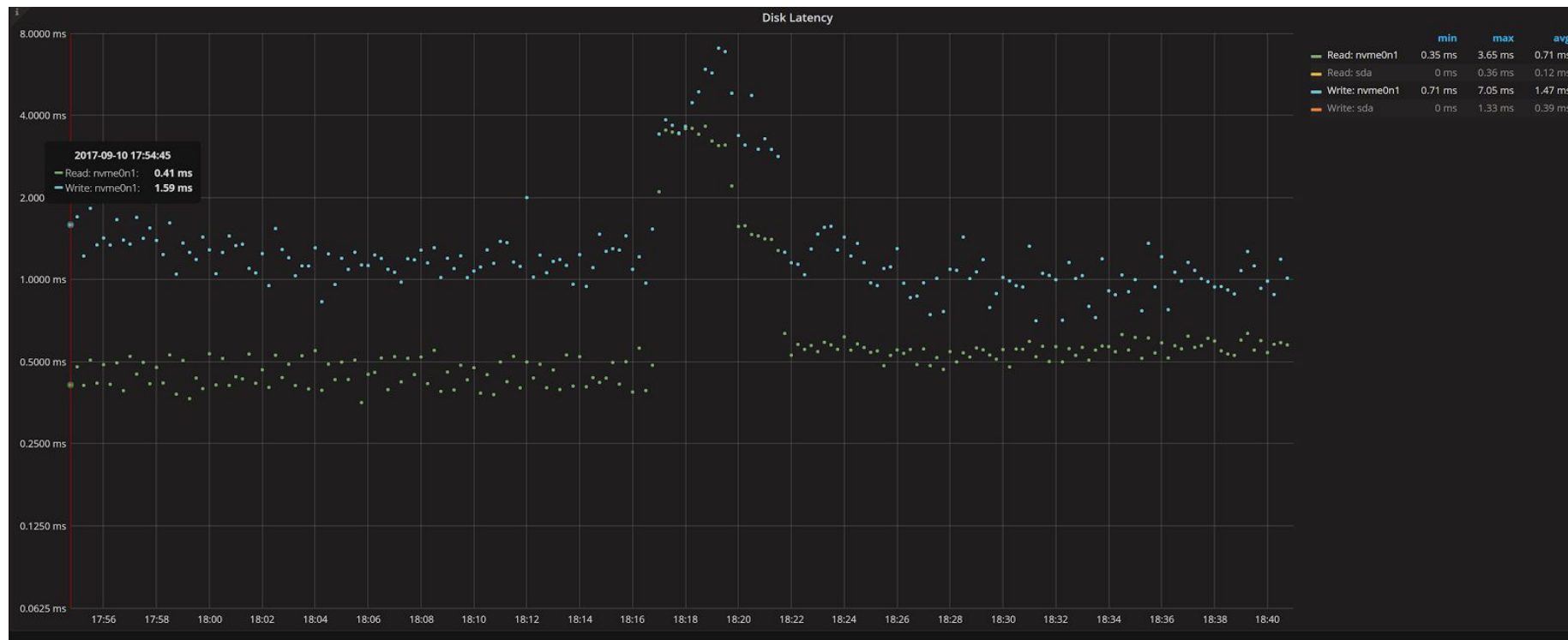
Two IO Spikes

- First to Flush Dirty Pages
- Second to work off higher query rate

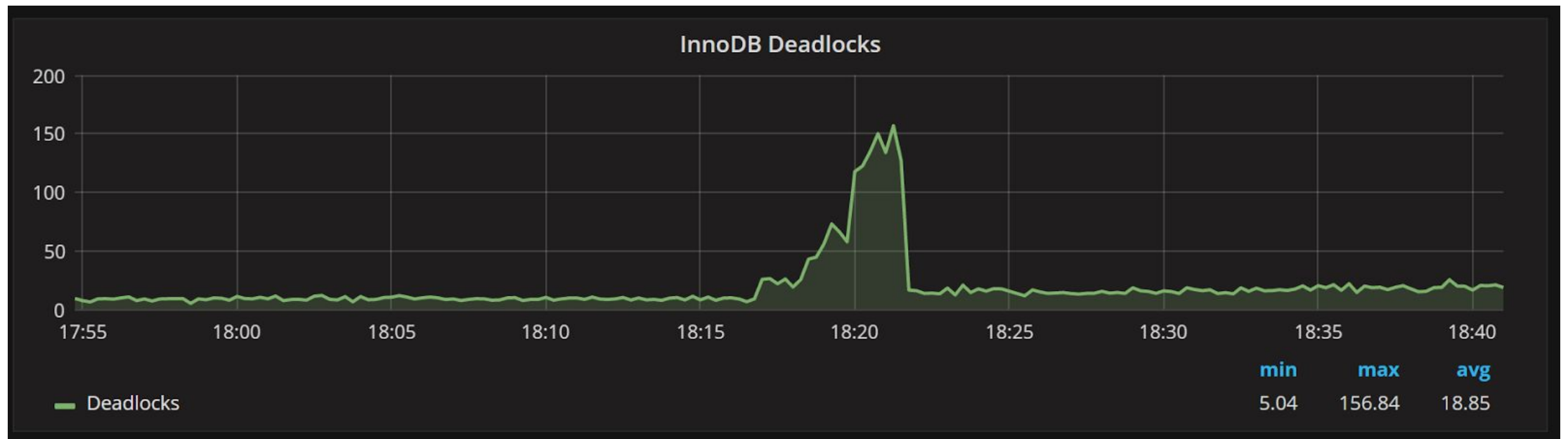


What is about Disk IO Latency ?

- Higher Number of IOPS does not always mean much higher latency

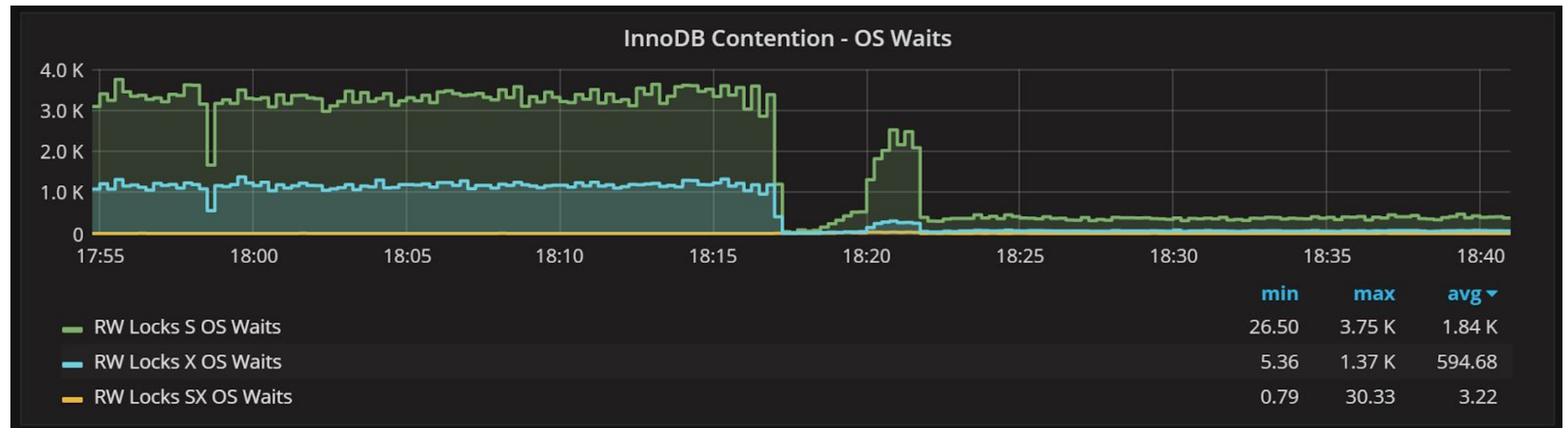


Longer Transactions = More Deadlocks



More IO Load Less Contention ?

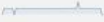


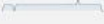



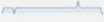




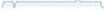

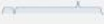



- Unsure why this is the case
- Note not ALL contention is shown in those graphs



Now we see query 80% IO Bound

SELECT sbtest

558CAEF5F387E929

Metrics			
Metrics	Rate/Sec	Sum	Per Query Stats
Query Count	9.75 k (per sec) 	35.09 m 50.15% of total	
Query Time	5.63 load 	5:37:50 38.73% of total	577.43 µs avg 
Lock Time	0.23 (avg load) 	0:13:39 28.03% of total 4.04% of query time	23.34 µs avg 
Innodb IO Read Wait	4.48 (avg load) 	4:29:03 46.96% of total 79.64% of query time	459.85 µs avg 
Innodb Read Ops	4.47 k (per sec) 	16.11 m 43.01% of total	0.00 avg 
Innodb Read Bytes	73.31 MB (per sec) 	263.91 GB 43.01% of total 16.38 KB avg io size	7.52 KB avg 
Innodb Distinct Pages	-	-	3.00 avg 
Rows Sent	9.75 k (per sec) 	35.09 m 3.21% of total	0.98 avg 
Bytes Sent	1.85 MB (per sec) 	6.67 GB 4.73% of total 190.00 Bytes bytes/row	189.98 Bytes avg 
Rows Examined	9.75 k (per sec) 	35.09 m 1.40% of total 1.00 per row sent	0.98 avg 

Summary

Can get a lot of Insights in MySQL Performance with PMM

Great tool to have when you're challenged troubleshoot MySQL

A lot of insights during benchmarking and evaluation

SAVE THE DATE!

April 23-25, 2018
Santa Clara Convention Center



PERCONA

LIVE

CALL FOR PAPERS OPENING SOON!

www.perconalive.com