How to cope with (unexpected) millupling of your workload?

May 2021

Art van Scheppingen - Senior Database Engineer
MessageBird - Amsterdam
art.vanscheppingen@messagebird.com

DISCLAIMER

Opinions shared are my own and do not necessarily reflect those of MessageBird

Agenda

- MessageBird & me
- Context
- Read offloading
- Parallel replication
- Other restrictions
- Conclusion

MessageBird & me

Who am I and what is MessageBird?

MessageBird

MessageBird powers communication between businesses and their customers — across any channel, always with the right context, and on every corner of the planet. Our products and solutions are the foundational building blocks to business communications across preferred channels, like SMS, Voice, WhatsApp, WeChat, Messenger, Email and more.

For additional information visit:
www.messagebird.com

25,000 customers
We work globally with companies of every size, from start-up to enterprise, across a wide variety of industries.

175 countries of operation
We have teams in every region and timezone, so we can offer 24/7 support to our customers.

700+ employees
Representing 55+ nationalities and based worldwide — we're a "Work Anywhere" company.

We’re hiring!

https://messagebird.com/careers

Context

What is the situation?

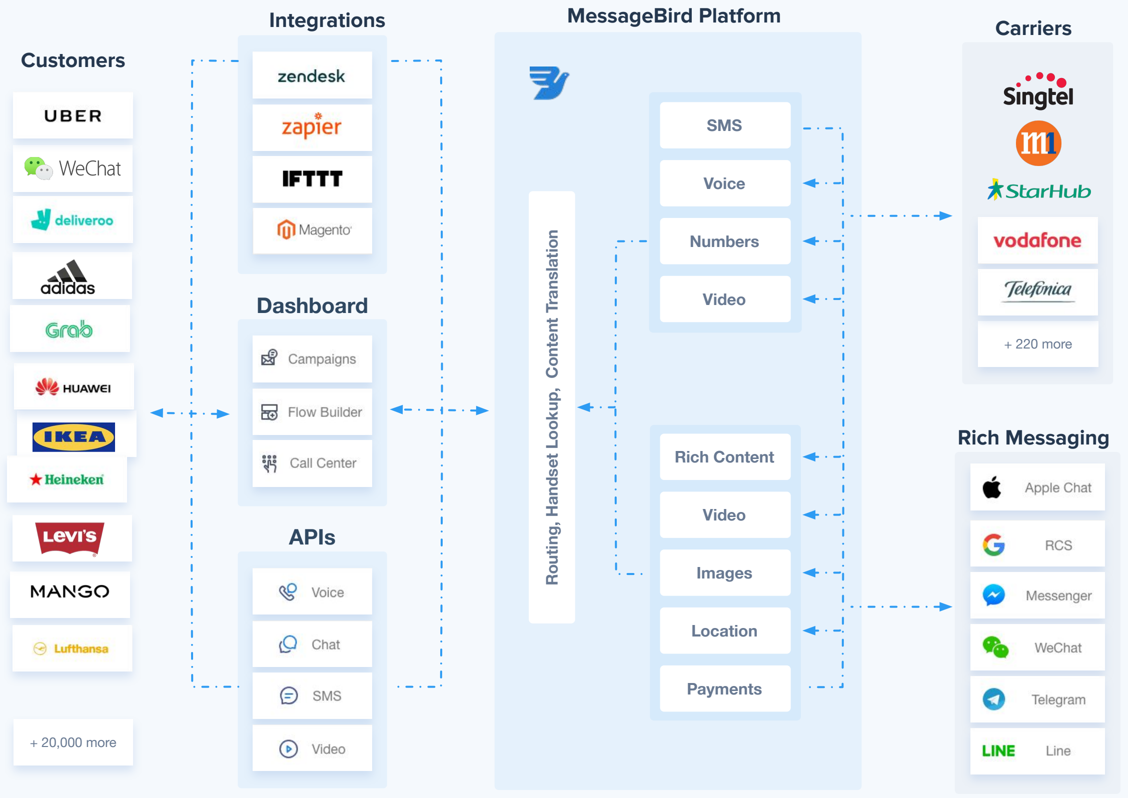
Milluplicate

Mille = 1000, Milluplicate = 1000 times

What do we observe?

Traffic spike that greatly exceeds our normal traffic

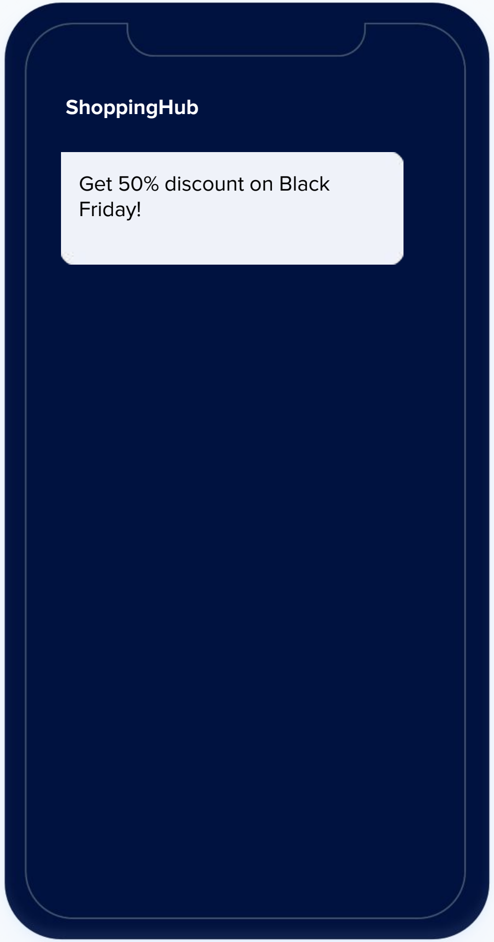
Very high level overview



Batch API

Bulk
Meant to send out messages in bulk and has low priority over other messages

1-to-n
Meant for reaching many recipients with a single message



Batch API

Bulk

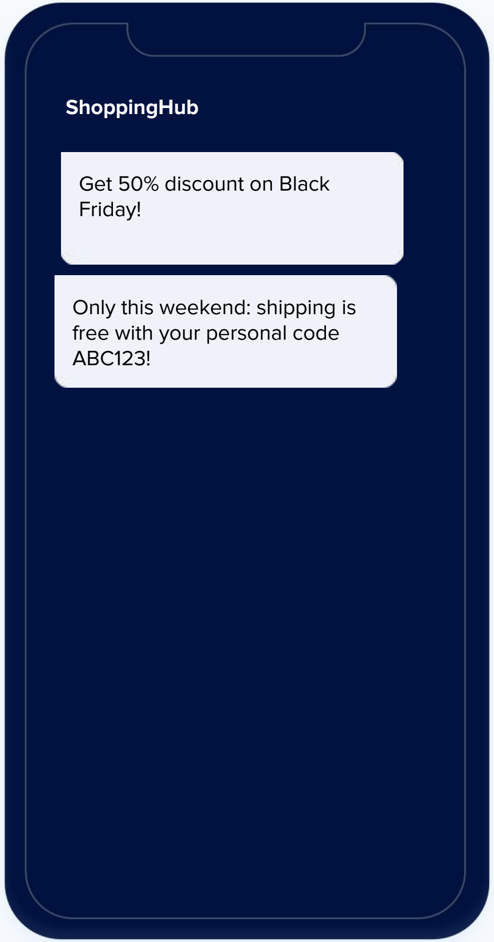
Meant to send out messages in bulk and has low priority over other messages

1-to-n

Meant for reaching many recipients with a single message

1-to-1

But you can also send a single message to a single person



ShoppingHub

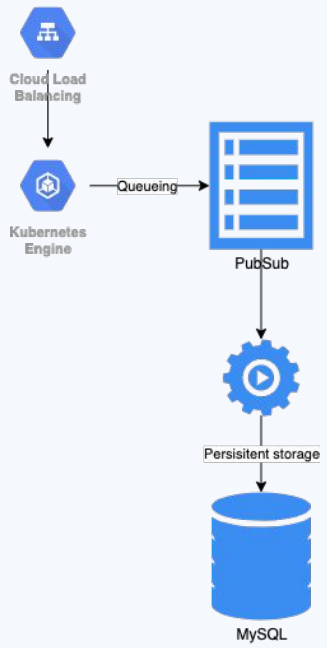
Get 50% discount on Black Friday!

Only this weekend: shipping is free with your personal code ABC123!

What were the issues with a sudden influx of bulk?

- Every API call inserts into the database directly
  - Thousands of parallel threads
  - Increasing the number of CPUs won’t help
    - Counter intuitive: decrease
- 1-to-n
  - A few API calls will cause many messages to be sent
- 1-to-1
  - Many API calls will cause many messages to be sent
- Solution:
  - Start queuing messages!
  - Handle less important messages async

Adding a queue in front of the data store



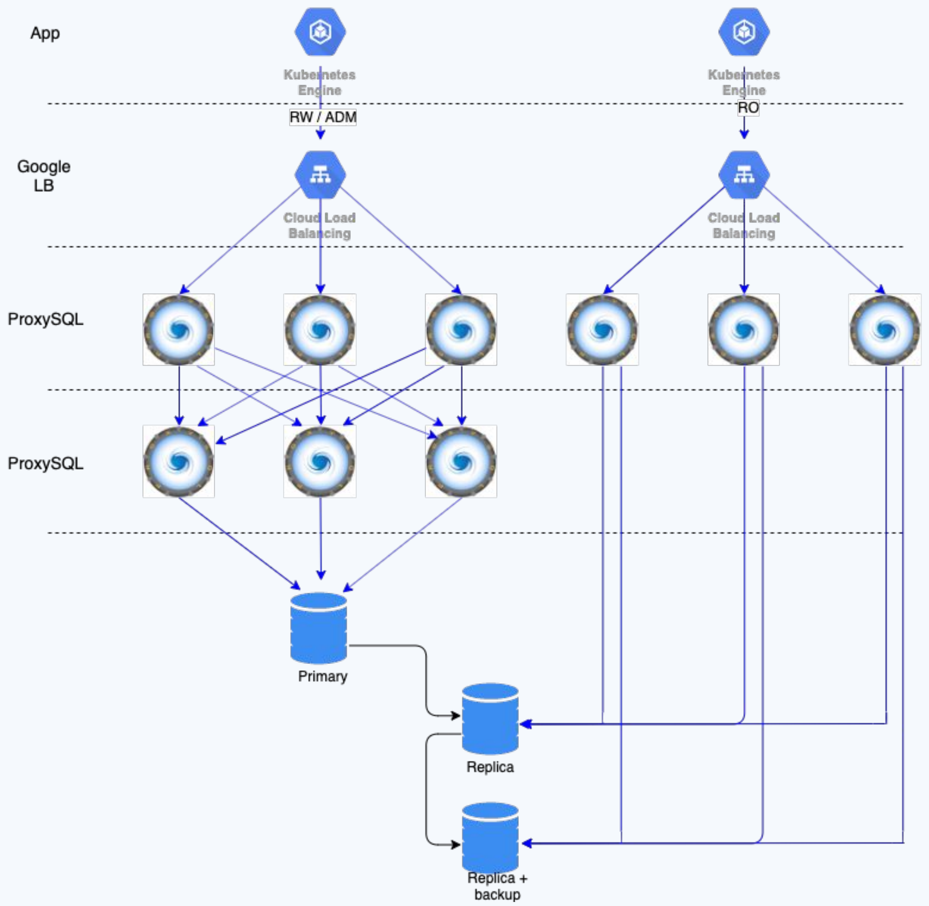
What were the issues after queueing?

- Queuing
  - We don’t persist messages immediately
  - PubSub is extremely reliable
  - Multi-region to ensure availability
- Concurrency
  - Queries overloading the primary
  - Hardly any read-offloading to replicas
- Solution:
  - Start offloading reads to replicas
    - Message status requests
    - Customer status overviews

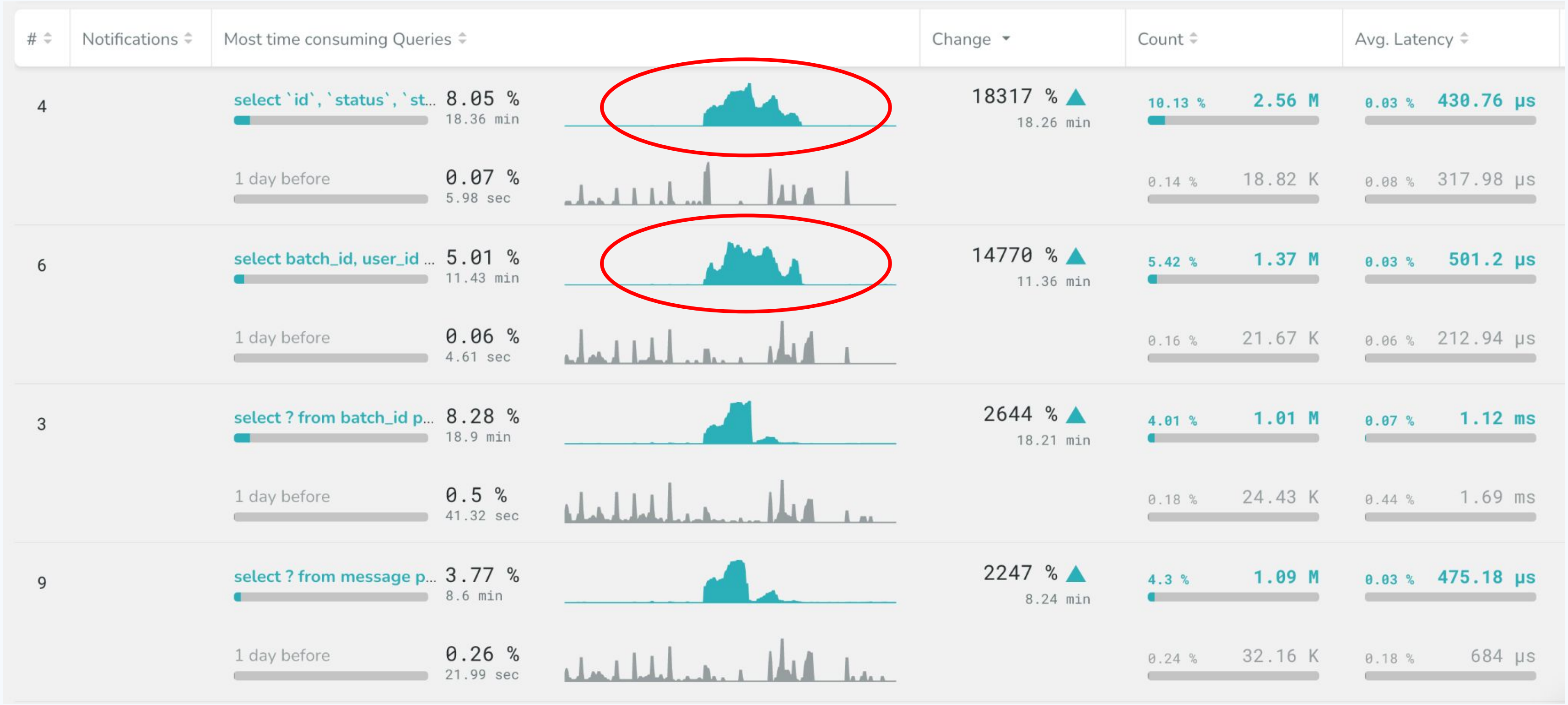
Read-offloading

What else are those replicas for?

A typical database at Messagebird



What we observed on replicas



#	Notifications	Most time consuming Queries		Change	Count		Avg. Latency	
4		select `id`, `status`, `st...	8.05 % 18.36 min	18317 % ▲ 18.26 min	10.13 %	2.56 M	0.03 %	430.76 µs
		1 day before	0.07 % 5.98 sec		0.14 %	18.82 K	0.08 %	317.98 µs
6		select batch_id, user_id ...	5.01 % 11.43 min	14770 % ▲ 11.36 min	5.42 %	1.37 M	0.03 %	501.2 µs
		1 day before	0.06 % 4.61 sec		0.16 %	21.67 K	0.06 %	212.94 µs
3		select ? from batch_id p...	8.28 % 18.9 min	2644 % ▲ 18.21 min	4.01 %	1.01 M	0.07 %	1.12 ms
		1 day before	0.5 % 41.32 sec		0.18 %	24.43 K	0.44 %	1.69 ms
9		select ? from message p...	3.77 % 8.6 min	2247 % ▲ 8.24 min	4.3 %	1.09 M	0.03 %	475.18 µs
		1 day before	0.26 % 21.99 sec		0.24 %	32.16 K	0.18 %	684 µs

What were the issues after read-offloading?

- Replication lag
  - Replicas get overloaded with read-requests
    - Message status isn’t there yet
    - Customers and automated systems will repeatedly retry
- Throttling
  - We can only handle X-amount of messages per second
  - More than X-number of msg we will get replication lag
- Solution:
  - Parallel replication

Parallel replication

Why would we need parallel replication?

Why we didn’t enable parallel replication from the start

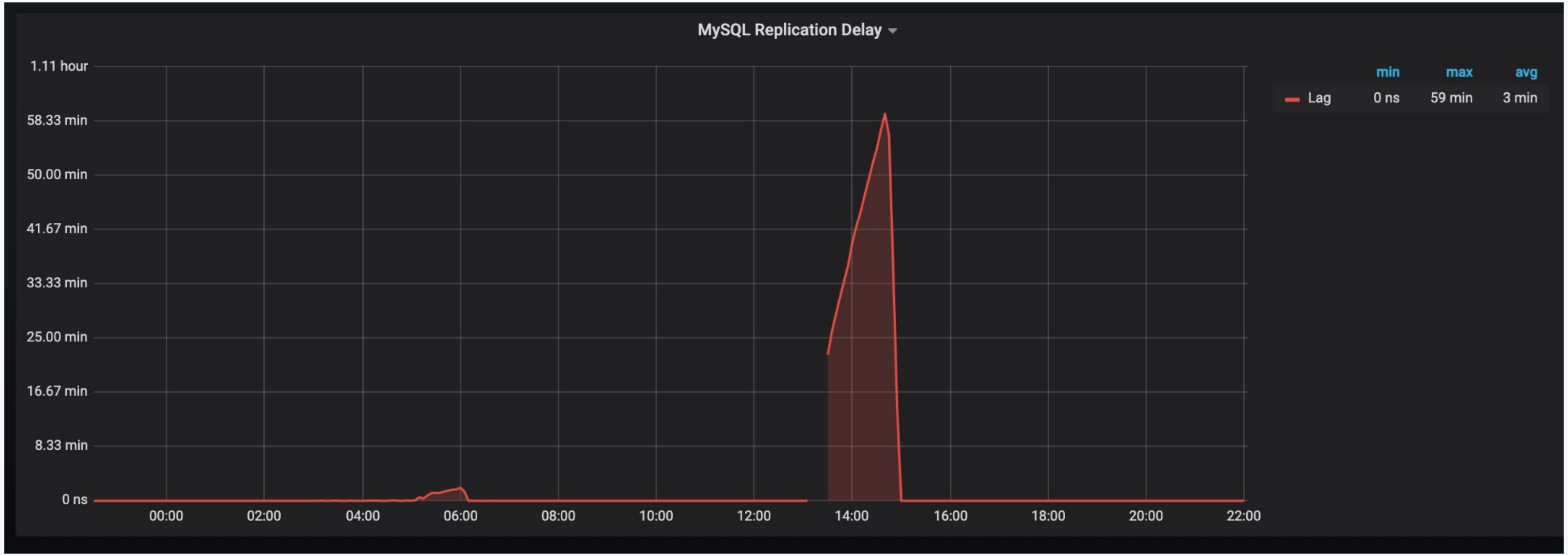
- Mixed mode replication
- Parallel replication requires a restart
    - Had to wait for failover/switchover to be in place
- No reads were offloaded to the replicas

Eliminate replication lag

- Mixed mode replication
- Group commit
  - How much delay are you able to add?
  - Offload queries to replicas
    - Replication lag issues
  - Don’t add delay to a replica without parallel replication enabled!

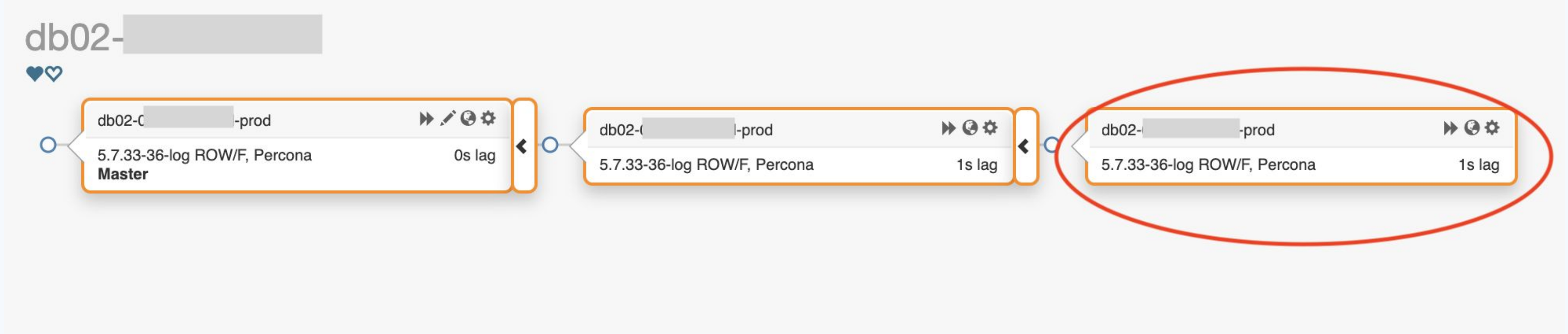
Eliminate replication lag

- **Don’t add delay to a replica without parallel replication enabled!**



Eliminate replication lag

- Don’t add delay to a replica without parallel replication enabled!



db02-[redacted]

db02-[redacted]-prod
5.7.33-36-log ROW/F, Percona 0s lag
Master

db02-[redacted]-prod
5.7.33-36-log ROW/F, Percona 1s lag

db02-[redacted]-prod
5.7.33-36-log ROW/F, Percona 1s lag

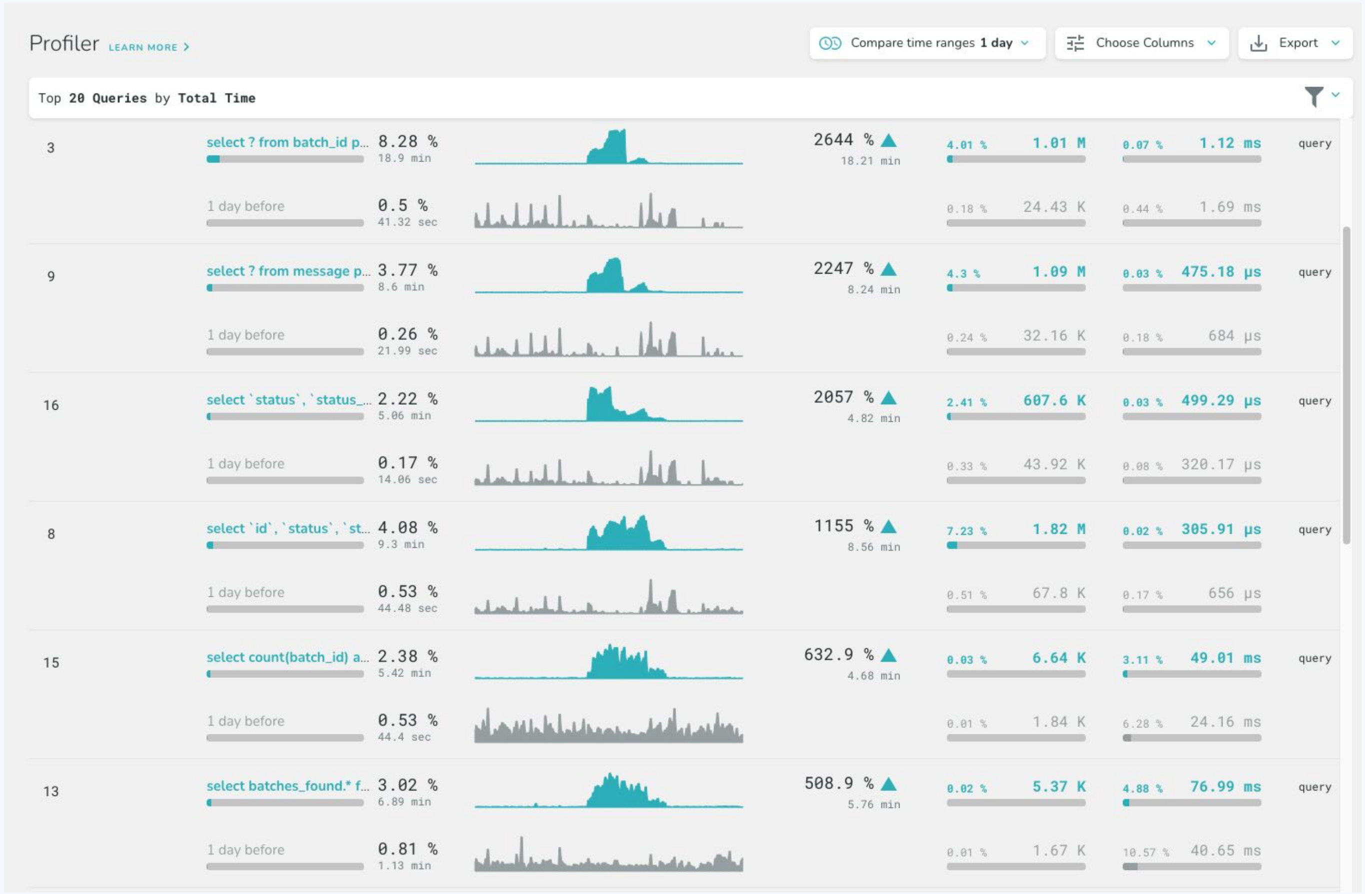
Eliminate replication lag

- Parallel replication
  - How many threads do we need during millupling?
  - How effective are threads during off hours?
  - How much delay do we need to add?
- Tuning session(s)
  - Add (non-sending) workload
  - 8 threads works fine during high workloads

Other restrictions

What else did we encounter?

What other behaviour did we observe during the influx



Profiler LEARN MORE ›

Compare time ranges **1 day** | Choose Columns | Export

Top **20 Queries** by **Total Time**

3	select ? from batch_id p...	8.28 % 18.9 min		2644 % ▲ 18.21 min	4.01 %	1.01 M	0.07 %	1.12 ms	query
	1 day before	0.5 % 41.32 sec			0.18 %	24.43 K	0.44 %	1.69 ms	
9	select ? from message p...	3.77 % 8.6 min		2247 % ▲ 8.24 min	4.3 %	1.09 M	0.03 %	475.18 µs	query
	1 day before	0.26 % 21.99 sec			0.24 %	32.16 K	0.18 %	684 µs	
16	select \`status\`, \`status_...	2.22 % 5.06 min		2057 % ▲ 4.82 min	2.41 %	607.6 K	0.03 %	499.29 µs	query
	1 day before	0.17 % 14.06 sec			0.33 %	43.92 K	0.08 %	320.17 µs	
8	select \`id\`, \`status\`, \`st...	4.08 % 9.3 min		1155 % ▲ 8.56 min	7.23 %	1.82 M	0.02 %	305.91 µs	query
	1 day before	0.53 % 44.48 sec			0.51 %	67.8 K	0.17 %	656 µs	
15	select count(batch_id) a...	2.38 % 5.42 min		632.9 % ▲ 4.68 min	0.03 %	6.64 K	3.11 %	49.01 ms	query
	1 day before	0.53 % 44.4 sec			0.01 %	1.84 K	6.28 %	24.16 ms	
13	select batches_found.* f...	3.02 % 6.89 min		508.9 % ▲ 5.76 min	0.02 %	5.37 K	4.88 %	76.99 ms	query
	1 day before	0.81 % 1.13 min			0.01 %	1.67 K	10.57 %	40.65 ms	

What other issues were observed?

- Strange query behaviour
  - Schema designed around 1-n relations
  - Many indexes
  - UUIDs pushing indexes out of memory

UUIDs pushing indexes out of memory - Example

```
CREATE TABLE message (
id bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT,
...
message_id VARCHAR(36) NOT NULL,
...
created_at DATETIME NOT NULL,
PRIMARY_KEY(id, created_at),
UNIQUE message_id_idx (message_id),
KEY customer_created_idx (customer, created_at),
...
)
PARTITION BY RANGE (MONTH(created_at)) (
    PARTITION p1 VALUES LESS THAN (2),
    PARTITION p2 VALUES LESS THAN (3),
    PARTITION p3 VALUES LESS THAN (4),
...
);
```

UUIDs pushing indexes out of memory

- What do we see?
  - Schema with `id` and `created_at` as PK (auto increment)
  - Unique UUID (`message_id`)
  - Datetime used for sorting range queries on various columns
- INSERTS
  - New rows will be inserted sequentially
  - Every row has a unique UUID, which is “random”
  - Random inserts means more parts of the `message_id_idx` index need to be loaded
  - Less memory available to satisfy range queries (sorting on datetime)
- SELECTS
  - Read-offloading of message status requires lookup by `message_id`
  - Also pushes indexes out of memory
- Solution:
  - Increase memory

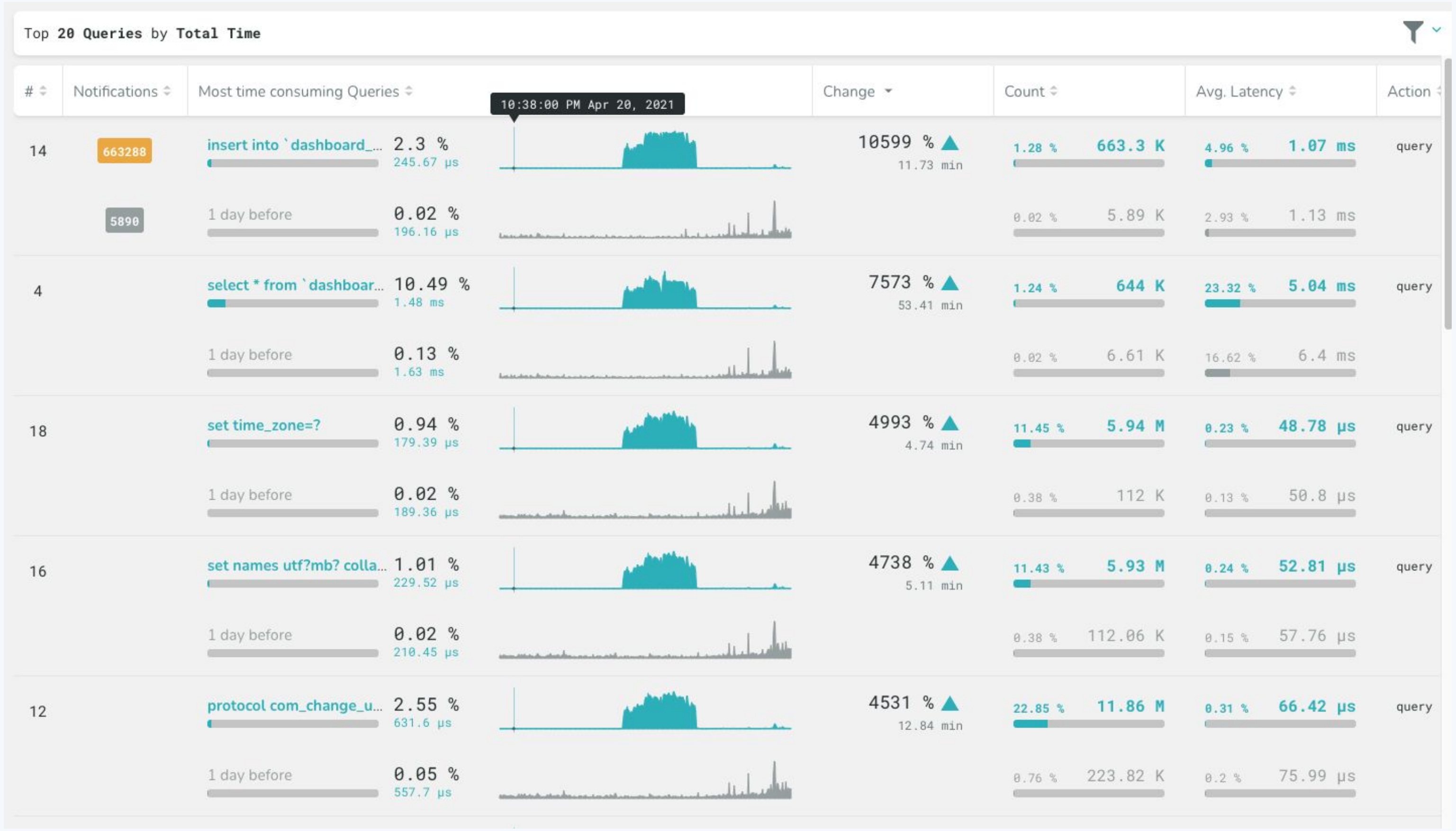
Conclusion

What did we do in the end?

What did we do?

- Scale down
    - Slow down concurrency
        - Queueing
- Offload queries to replicas
    - Address replication lag issues: enable parallel replication
- Enable parallel replication
    - Change binary logging mode
    - Tune for intensive workload
- Scale up
    - Increase memory to keep (more) indexes in memory

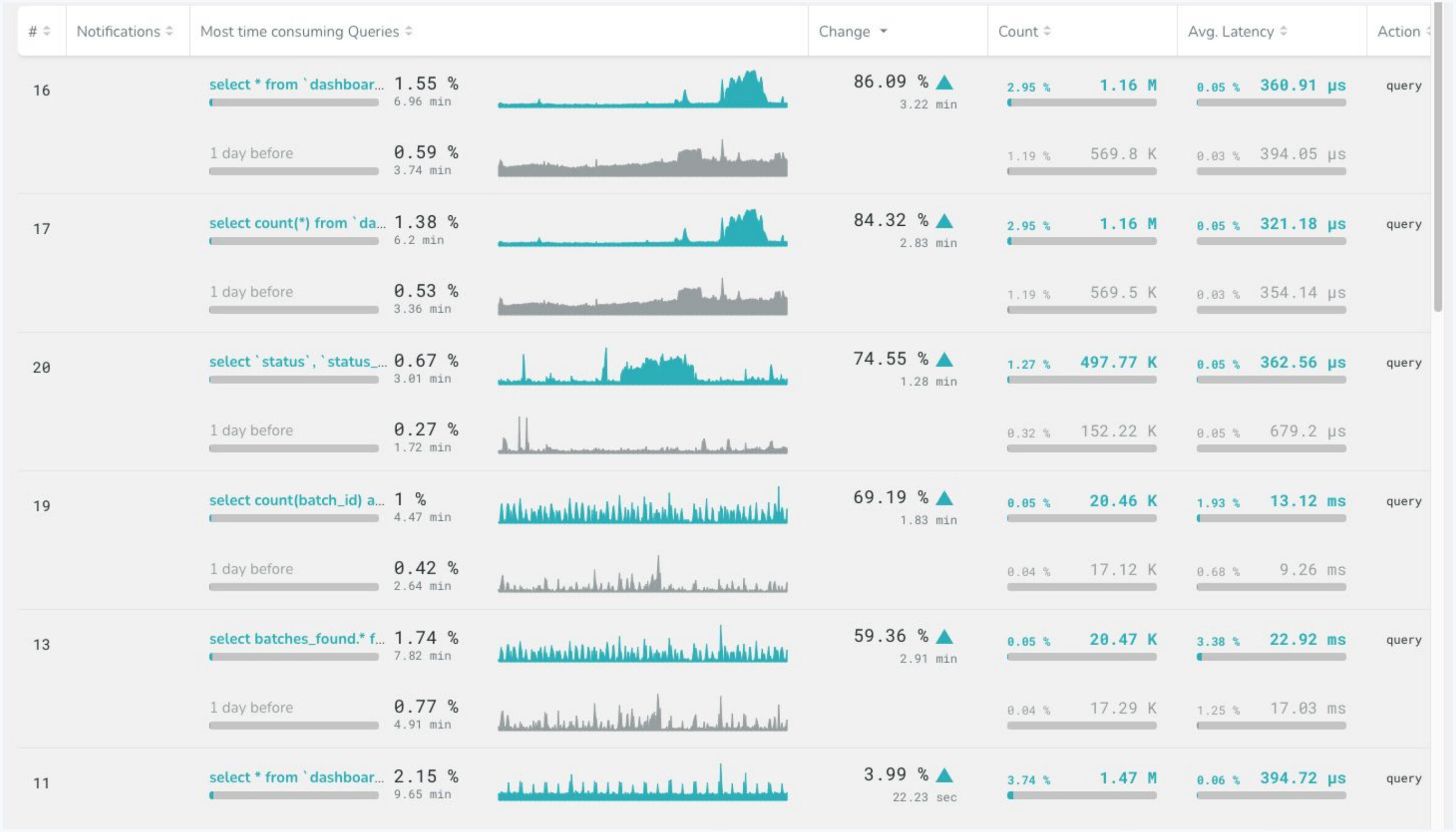
The result [1 of 2] (primary)



Top **20 Queries** by **Total Time**

#	Notifications	Most time consuming Queries		Change	Count		Avg. Latency		Action
14	663288	insert into `dashboard_...	2.3 % 245.67 µs	10599 % ▲ 11.73 min	1.28 %	663.3 K	4.96 %	1.07 ms	query
	5890	1 day before	0.02 % 196.16 µs		0.02 %	5.89 K	2.93 %	1.13 ms	
4		select * from `dashboar...	10.49 % 1.48 ms	7573 % ▲ 53.41 min	1.24 %	644 K	23.32 %	5.04 ms	query
		1 day before	0.13 % 1.63 ms		0.02 %	6.61 K	16.62 %	6.4 ms	
18		set time_zone=?	0.94 % 179.39 µs	4993 % ▲ 4.74 min	11.45 %	5.94 M	0.23 %	48.78 µs	query
		1 day before	0.02 % 189.36 µs		0.38 %	112 K	0.13 %	50.8 µs	
16		set names utf?mb? colla...	1.01 % 229.52 µs	4738 % ▲ 5.11 min	11.43 %	5.93 M	0.24 %	52.81 µs	query
		1 day before	0.02 % 210.45 µs		0.38 %	112.06 K	0.15 %	57.76 µs	
12		protocol com_change_u...	2.55 % 631.6 µs	4531 % ▲ 12.84 min	22.85 %	11.86 M	0.31 %	66.42 µs	query
		1 day before	0.05 % 557.7 µs		0.76 %	223.82 K	0.2 %	75.99 µs	

The result [2 of 2] (replica)



#	Notifications	Most time consuming Queries			Change	Count		Avg. Latency		Action
16		select * from `dashboar...	1.55 %	6.96 min	86.09 % ▲ 3.22 min	2.95 %	1.16 M	0.05 %	360.91 µs	query
		1 day before	0.59 %	3.74 min		1.19 %	569.8 K	0.03 %	394.05 µs	
17		select count(*) from `da...	1.38 %	6.2 min	84.32 % ▲ 2.83 min	2.95 %	1.16 M	0.05 %	321.18 µs	query
		1 day before	0.53 %	3.36 min		1.19 %	569.5 K	0.03 %	354.14 µs	
20		select `status`, `status_...	0.67 %	3.01 min	74.55 % ▲ 1.28 min	1.27 %	497.77 K	0.05 %	362.56 µs	query
		1 day before	0.27 %	1.72 min		0.32 %	152.22 K	0.05 %	679.2 µs	
19		select count(batch_id) a...	1 %	4.47 min	69.19 % ▲ 1.83 min	0.05 %	20.46 K	1.93 %	13.12 ms	query
		1 day before	0.42 %	2.64 min		0.04 %	17.12 K	0.68 %	9.26 ms	
13		select batches_found.* f...	1.74 %	7.82 min	59.36 % ▲ 2.91 min	0.05 %	20.47 K	3.38 %	22.92 ms	query
		1 day before	0.77 %	4.91 min		0.04 %	17.29 K	1.25 %	17.03 ms	
11		select * from `dashboar...	2.15 %	9.65 min	3.99 % ▲ 22.23 sec	3.74 %	1.47 M	0.06 %	394.72 µs	query

What can we still improve?

- Scaling
  - Improve slowest components (scale up)
  - Offload more queries to replicas
  - Increase number of replicas
- Schema restrictions
  - Design new schema that is able to cope with 1-to-1 relations
  - Diversify indexes
    - Difficult with failover/switchover candidates
- Move data out of cluster (6 months currently)
  - Archive database
  - Requires application overhaul
- Sharding
  - Spread writes over many hosts
  - Requires schema and application overhaul

The dilemma

Do we wish/need to optimize more for high traffic?

(represents 5% of the time)

Thank you for attending!

Questions?