How to cope with (unexpected) swelling 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 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
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
    - Counterintuitive: 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

<table>
<thead>
<tr>
<th>#</th>
<th>Notifications</th>
<th>Most time consuming Queries</th>
<th>Change</th>
<th>Count</th>
<th>Avg. Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>select 'id', 'status', 'st...'</td>
<td>8.05 % 18.36 min</td>
<td><strong>18317 % ▲ 18.25 min</strong></td>
<td>▲ 18.13 %</td>
<td>2.56 M</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.07 % 5.98 sec</td>
<td></td>
<td>▲ 0.14 %</td>
<td>18.82 K</td>
</tr>
<tr>
<td>6</td>
<td>select batch_id, user_id...</td>
<td>5.01 % 11.43 min</td>
<td><strong>14770 % ▲ 11.36 min</strong></td>
<td>▲ 5.42 %</td>
<td>1.37 M</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.06 % 4.61 sec</td>
<td></td>
<td>▲ 0.16 %</td>
<td>21.67 K</td>
</tr>
<tr>
<td>3</td>
<td>select ? from batch_id p...</td>
<td>8.28 % 18.9 min</td>
<td><strong>2644 % ▲ 18.21 min</strong></td>
<td>▲ 4.01 %</td>
<td>1.01 M</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.5 % 41.32 sec</td>
<td></td>
<td>▲ 0.18 %</td>
<td>24.43 K</td>
</tr>
<tr>
<td>9</td>
<td>select ? from message p...</td>
<td>3.77 % 8.6 min</td>
<td><strong>2247 % ▲ 8.24 min</strong></td>
<td>▲ 4.3 %</td>
<td>1.09 M</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>8.26 % 21.99 sec</td>
<td></td>
<td>▲ 0.24 %</td>
<td>32.16 K</td>
</tr>
</tbody>
</table>
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!
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

- 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
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

```sql
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
## Top 20 Queries by Total Time

<table>
<thead>
<tr>
<th>#</th>
<th>Notifications</th>
<th>Most time consuming Queries</th>
<th>Change</th>
<th>Count</th>
<th>Avg. Latency</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>663288</td>
<td>insert into <code>dashboard</code>...</td>
<td>2.3 %</td>
<td>10599</td>
<td>1.07 ms</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 day before</td>
<td>0.02 %</td>
<td>106.16</td>
<td>1.73 min</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5996</td>
<td>select * from <code>dashboard</code>...</td>
<td>10.49 %</td>
<td>7573</td>
<td>5.04 ms</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 day before</td>
<td>0.13 %</td>
<td>106.16</td>
<td>1.73 min</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>175.39 μs</td>
<td>set time_zone=?</td>
<td>0.94 %</td>
<td>4993</td>
<td>48.78 μs</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 day before</td>
<td>0.02 %</td>
<td>106.16</td>
<td>1.73 min</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>216.48 μs</td>
<td>set names utf8mb4 collate...</td>
<td>1.01 %</td>
<td>4738</td>
<td>52.81 μs</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 day before</td>
<td>0.02 %</td>
<td>106.16</td>
<td>1.73 min</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>557.7 μs</td>
<td>protocol comm_change_u...</td>
<td>2.55 %</td>
<td>4531</td>
<td>66.42 μs</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 day before</td>
<td>0.05 %</td>
<td>106.16</td>
<td>1.73 min</td>
<td></td>
</tr>
</tbody>
</table>
### The result [2 of 2] (replica)

<table>
<thead>
<tr>
<th>ID</th>
<th>Most time consuming Queries</th>
<th>Change</th>
<th>Count</th>
<th>Avg. Latency</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td><code>select * from 'dashba...</code></td>
<td>1.55%</td>
<td>86.09%</td>
<td>2.95%</td>
<td>0.95%</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.59%</td>
<td>0.69%</td>
<td>1.16</td>
<td>0.03%</td>
</tr>
<tr>
<td>17</td>
<td><code>select count(*) from 'da...</code></td>
<td>1.38%</td>
<td>84.32%</td>
<td>2.95%</td>
<td>0.95%</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.53%</td>
<td>0.67%</td>
<td>1.16</td>
<td>0.03%</td>
</tr>
<tr>
<td>20</td>
<td><code>select 'status', 'status...</code></td>
<td>0.67%</td>
<td>74.55%</td>
<td>1.27%</td>
<td>0.95%</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.27%</td>
<td>0.51%</td>
<td>0.80</td>
<td>0.05%</td>
</tr>
<tr>
<td>19</td>
<td><code>select count(batch_id) a...</code></td>
<td>1%</td>
<td>69.19%</td>
<td>0.95%</td>
<td>1.93%</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.42%</td>
<td>0.74%</td>
<td>1.83</td>
<td>0.04%</td>
</tr>
<tr>
<td>13</td>
<td><code>select batches_found.* f...</code></td>
<td>1.74%</td>
<td>59.36%</td>
<td>0.85%</td>
<td>1.25%</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.77%</td>
<td>1.01%</td>
<td>2.19</td>
<td>0.04%</td>
</tr>
<tr>
<td>11</td>
<td><code>select * from 'dashba...</code></td>
<td>2.15%</td>
<td>3.99%</td>
<td>2.34%</td>
<td>0.95%</td>
</tr>
<tr>
<td></td>
<td>1 day before</td>
<td>0.65%</td>
<td>0.96%</td>
<td>22.23 sec</td>
<td></td>
</tr>
</tbody>
</table>
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?
(represented 5% of the time)
Thank you for attending!
Questions?