readyset

Stop Throwing Hardware at the Problem

The Case for Smarter Scaling with Query Caching

MySQL Belgium Days
30/01/2025

Marcelo Altmann

- **Senior Software Engineer @ Readyset**
- **Senior Software Engineer @ Percona**
  - **Maintainer of Percona XtraBackup for MySQL**
- **Long time Community contributor**
- **Oracle ACE**
- **MySQL Code Contributor**
- **MySQL Rust Driver Contributor**

Gautam Gopinadhan

- **CEO @ Readyset**
- **Past Datarobot, Microsoft Azure, StorSimple, Cisco, Sun Microsystems.**
  - **Azure Blob Storage, Azure Edge Computing.**
  - **Distributed Systems, Filesystems, Storage caching, WAN Optimization, Security, Media Streaming**

Agenda

Agenda

- **Scaling solutions for an example query**
- **How Readyset compares**
- **Demo**
- **ProxySQL & Readyset**
- **Case Study**

Database Scaling

Database Scaling

- **Out of resource to serve queries**
- **Database = Application performance**
- **Cache complements your database infrastructure**
- **Suitable for applications requiring low latency and high throughput.**

Database Scaling

```
mysql> SELECT d.dept_name, COUNT(*) AS employee_count FROM dept_emp de JOIN departments d ON
de.dept_no = d.dept_no WHERE de.to_date = '9999-01-01' GROUP BY de.dept_no;
+--------------------+----------------+
| dept_name          | employee_count |
+--------------------+----------------+
| Development        |          61386 |
| Sales              |          37701 |
| Production         |          53304 |
| Human Resources    |          12898 |
| Research           |          15441 |
| Quality Management |          14546 |
| Marketing          |          14842 |
| Customer Service   |          17569 |
| Finance            |          12437 |
+--------------------+----------------+
9 rows in set (0.437 sec)
```

Database Scaling

```
mysql> EXPLAIN ANALYZE FORMAT=TREE SELECT d.dept_name. . .

| -> Table scan on <temporary>  (actual time=442..442 rows=9 loops=1)
    -> Aggregate using temporary table  (actual time=442..442 rows=9 loops=1)
        -> Nested loop inner join  (cost=44889 rows=33114) (actual time=0.605..346 rows=240124
loops=1)
            -> Filter: (de.to_date = DATE'9999-01-01')  (cost=33299 rows=33114) (actual
time=0.594..88.5 rows=240124 loops=1)
                -> Table scan on de  (cost=33299 rows=331143) (actual time=0.591..65.7
rows=331603 loops=1)
            -> Single-row index lookup on d using PRIMARY (dept_no=de.dept_no)  (cost=0.25
rows=1) (actual time=913e-6..938e-6 rows=1 loops=240124)
```

Database Scaling

```
mysql> mysql> CREATE INDEX idx_dept_emp_to_date ON dept_emp(to_date, dept_no);
Query OK, 0 rows affected (0.965 sec)

mysql> EXPLAIN ANALYZE FORMAT=TREE SELECT d.dept_name
| -> Table scan on <temporary>  (actual time=152..152 rows=9 loops=1)
    -> Aggregate using temporary table  (actual time=152..152 rows=9 loops=1)
        -> Nested loop inner join  (cost=19.1 rows=85.7) (actual time=0.0382..75.3 rows=240124
loops=1)
            -> Covering index scan on d using dept_name  (cost=1.15 rows=9) (actual
time=0.0163..0.0195 rows=9 loops=1)
            -> Covering index lookup on de using idx_dept_emp_to_date (to_date=DATE'9999-01-01',
dept_no=d.dept_no)  (cost=1.15 rows=9.53) (actual time=0.0069..6.96 rows=26680 loops=9)
```

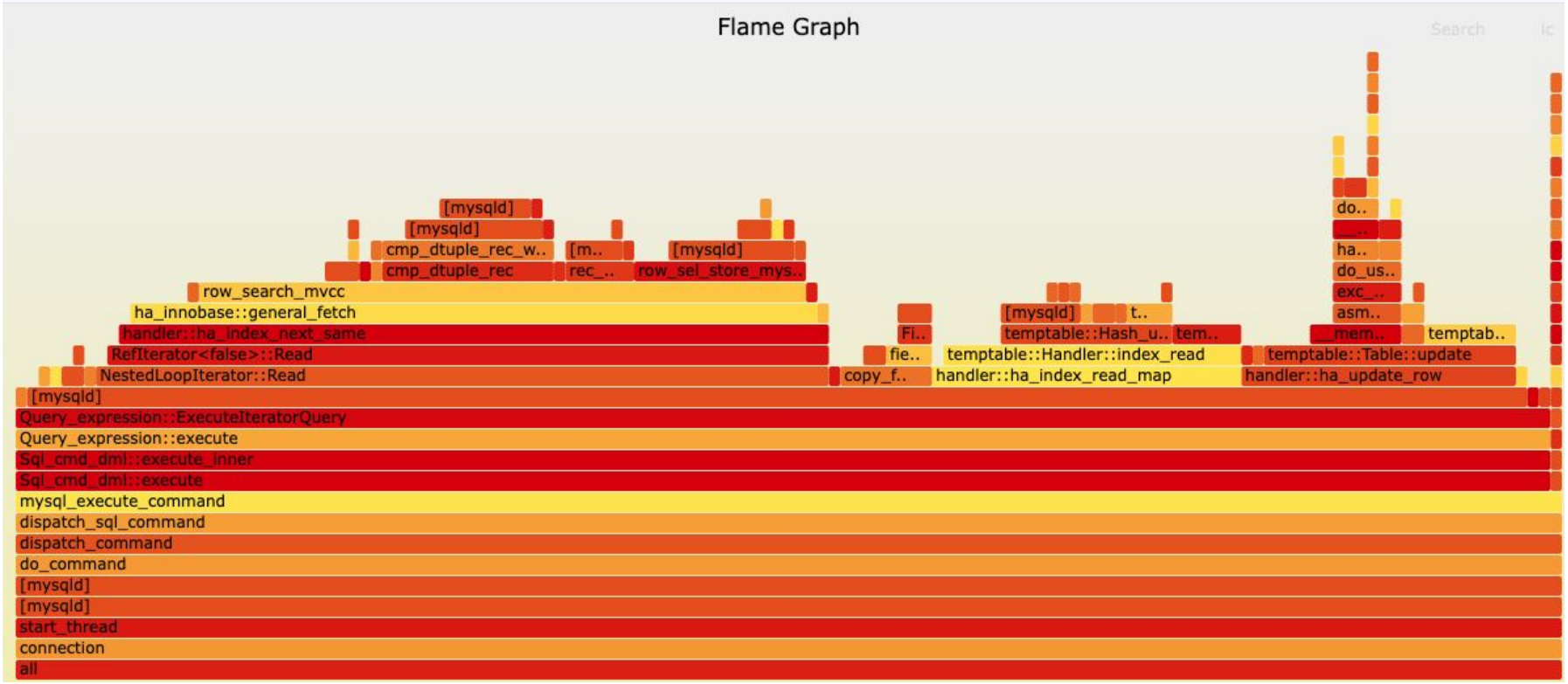
Database Scaling

```
mysql> SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_read%';
. . .
| Innodb_buffer_pool_read_requests      | 4910464 |
| Innodb_buffer_pool_reads              | 3144    |

mysql> SELECT d.dept_name, COUNT(*) AS employee_count FROM dept_emp de JOIN departments d ON
de.dept_no = d.dept_no WHERE de.to_date = '9999-01-01' GROUP BY de.dept_no;
. .
9 rows in set (0.138 sec)

mysql> SHOW GLOBAL STATUS LIKE 'Innodb_buffer_pool_read%';
. . .
| Innodb_buffer_pool_read_requests      | 5112188 |
| Innodb_buffer_pool_reads              | 3144    |
```

Database Scaling



Database Scaling

```
[ec2-user@ip-10-0-16-172 fosdem]$ sudo perf stat -p $(pidof mysqld) -- mysql -e "SELECT . . .

 Performance counter stats for process id '5981':

            136.34 msec task-clock                #    0.954 CPUs utilized
                33      context-switches          #  242.050 /sec
                 3      cpu-migrations            #   22.005 /sec
              5795      page-faults               #   42.506 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       0.142859932 seconds time elapsed
```

Database Scaling - QPS

$$\text{Queries Per Second} = \frac{\text{Total CPU Capacity}}{\text{Task Clock Time}}$$

$$\text{Queries Per Second} = \frac{16}{0.13634}$$

$$\text{Queries Per Second} = 117.35\,\text{queries/second}$$

Readyset

Readyset

- **Based on MIT PHD Thesis - Partial State in Dataflow-Based Materialized Views (Noria) by Jon Gjengset**
- **https://github.com/readysettech/readyset**
- **Core / Cloud**
- **No application code changes / Wire compatible with MySQL & PostgreSQL**
- **Automatic cache maintenance / No TTL**
- **https://www.youtube.com/watch?v=ch0utkJqJZQ**

Database Scaling - Readyset

```
readyset> SELECT d.dept_name, COUNT(*) AS employee_count FROM dept_emp de JOIN departments d ON
de.dept_no = d.dept_no WHERE de.to_date = '9999-01-01' GROUP BY de.dept_no;
+--------------------+----------------+
| dept_name          | employee_count |
+--------------------+----------------+
| Customer Service   |          17569 |
| Development        |          61386 |
| Finance            |          12437 |
| Human Resources    |          12898 |
| Marketing          |          14842 |
| Production         |          53304 |
| Quality Management |          14546 |
| Research           |          15441 |
| Sales              |          37701 |
+--------------------+----------------+
9 rows in set (0.000 sec) ← FR https://bugs.mysql.com/bug.php?id=117270
```

Database Scaling - Readyset

```
[ec2-user@ip-10-0-16-172 ~]$ sudo perf stat -p $(pidof readyset) -- mysqlr -e "SELECT ...

 Performance counter stats for process id '58344':

              0.65 msec task-clock                       #    0.005 CPUs utilized
                 7      context-switches                 #   10.847 K/sec
                 0      cpu-migrations                   #    0.000 /sec
                 0      page-faults                      #    0.000 /sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       0.142028280 seconds time elapsed
```

Database Scaling - QPS

$$\text{Queries Per Second} = \frac{16}{0.00065}$$

$$\text{Queries Per Second} = 24,615.38 \, \text{queries/second}$$

Database Scaling - Scaling Down

```
# MySQL on c6.4xlarge (16 vCPUs) - $0.612 / Hour
MySQL delivers each query in 136.34ms and a max capacity of 117.35 QPS

# Readyset on c6a.large (2 vCPUs) - $0.0765 / Hour
Readyset delivers each query in 0.65ms and a total of 24,615 QPS (16 vCPUS) - 3,076.92 QPS (~26x)
```

 # MySQL: $5,360.88/year
 # ReadySet: $670.14/year
 # Improvement: ~8x cheaper for ReadySet while delivering 26x improvement in QPS and 209x improvement in latency.

Readyset Demo

Big Idea

Generalized solution != Common Scenarios

Cost of Generalization is high

- Query parsing, execution complexity
- Replication units are complex

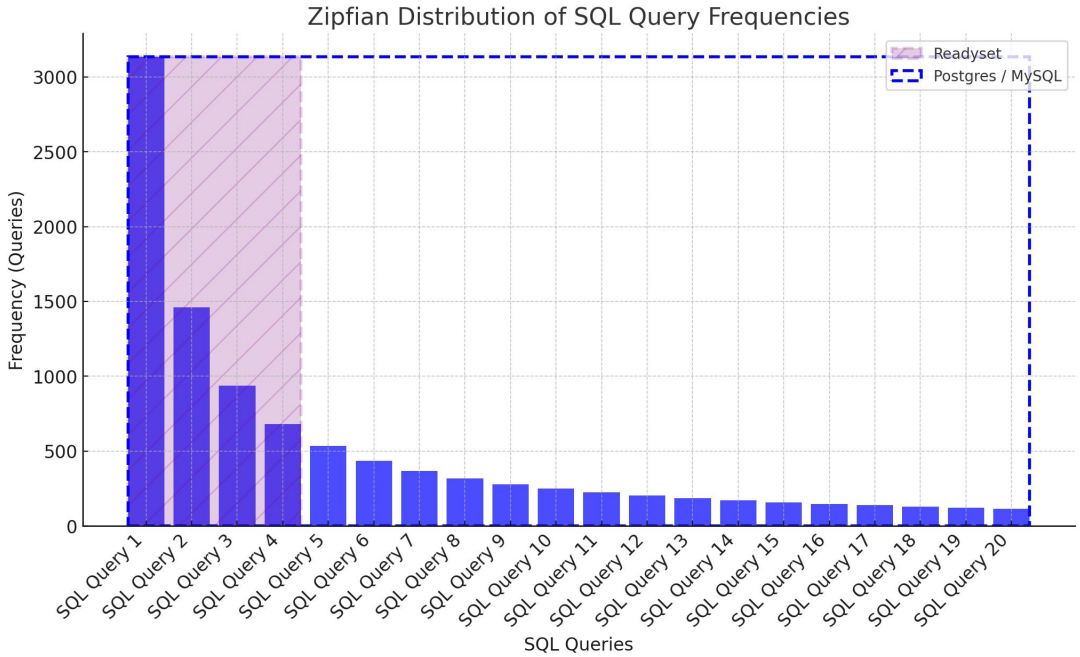
Augment with a smart Query Cache for Cost and Performance benefits.

Examples:

- Deepseek vs Big LLM
- DuckDB vs Big Data Distributed Systems

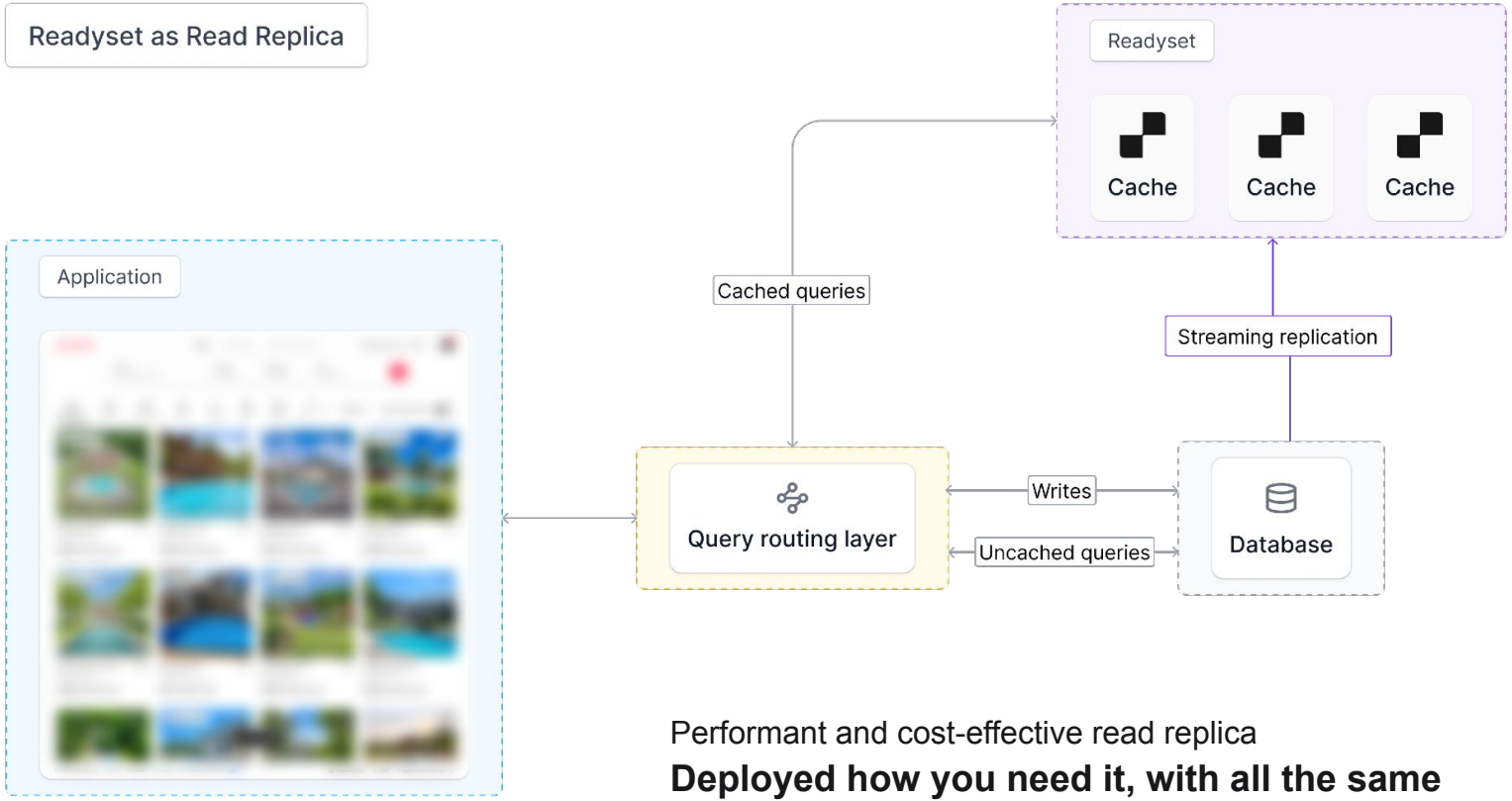
Big Idea: Key observations about Web Application Workloads

- Queries often follow a Zipfian distribution
- Read heavy workloads. Infrequent changes. Eventual consistency is okay.



ProxySQL & Readyset

Readyset as Read Replica



Performant and cost-effective read replica

Deployed how you need it, with all the same performance benefits, no replication lag, and substantial cost improvements.

ProxySQL – The Intelligent MySQL Proxy

A powerful, open-source proxy layer between your applications and MySQL servers.

Acts as an intermediary, optimizing query routing, enhancing performance, and improving database efficiency.

Increased Performance: Faster query execution and reduced latency.

Improved Scalability: Handles growing traffic demands effectively.

Enhanced Reliability: Minimizes downtime and ensures business continuity.

Simplified Management: Streamlines database administration tasks.

ProxySQL & Readyset

Open Source

- **Scheduler - https://github.com/readysettech/proxysql_scheduler**
- **Automatic Query Caching**
- **Cache Warm-up**

ProxySQL + Readyset On the Cloud

- **Announcing ProxySQL + Readyset on Readyset.cloud**

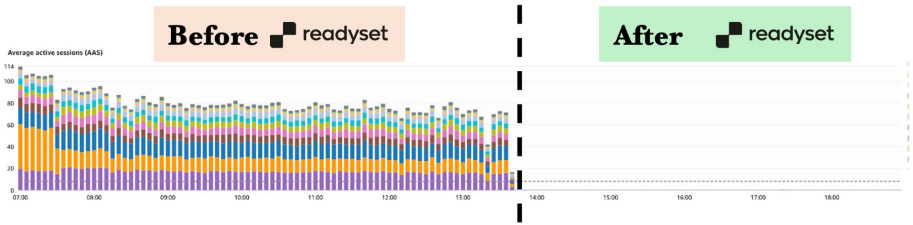
Case Study

Case Study

- **Leading Marketing Platform**
  - Number of read replicas reduced from 8 to 2 resulting in thousands of dollars in cost savings per month
  - In-production since August 2024
  - "*40,000 queries disappeared from the [primary database] server and they were literally nano-second response times with Readyset*" - Head of Platform

- **B2B Telecommunication Solution**
  - Customer using Readyset Private in a local setup
  - Managed to reduce workload on a 96 core server at 80% capacity to 18% with a single 12 core, 10GB Readyset server
  - *"I tried to turn Readyset off. I could not!"* - Sr. Software Engineer

Database Load in production on AWS Aurora



Primary AWS Aurora instance before and after Readyset - Source

Summary

Happy Hour

Hosted by Readyset

Saturday 1st Feb - 6PM

@Au Bassin

RSVP - https://bit.ly/ReadysetRSVP

Thank You!