Opensource Column Store Databases: MariaDB ColumnStore vs. ClickHouse

Alexander Rubin
VirtualHealth
About me

● Working with MySQL for 10-15 years
  ○ Started at MySQL AB 2006
    - Sun Microsystems, Oracle (MySQL Consulting)
    - Percona since 2014
  ○ Recently joined Virtual Health (medical records startup)
Caution:

1. This talk is **not about specifics of implementation**
   - A number of presentations about Clickhouse and MariaDB @ Percona Live 2019

2. This is all about:
   - **What?** -- what is the problem
   - **Why?** -- why queries are slow
   - **How?** -- how to solve

3. Examples are real-world example, medical insurance records
   - (but no actual PII data shown)
Simple query - top 10 - clients who visited doctors most often (data from 2017-2019)

```
mysql> SELECT
    ->   client_id,
    ->   min(date) as first_visit,
    ->   max(date) as last_visit,
    ->   count(distinct date) as days_visited,
    ->   count(cv.id) as visits,
    ->   count(distinct cv.service_location_name) as locations
    -> FROM client_visit cv
    -> GROUP BY client_id
    -> ORDER by visits desc
    -> LIMIT 10;

+-----------+-------------+------------+--------------+--------+-----------+
<table>
<thead>
<tr>
<th>client_id</th>
<th>first_visit</th>
<th>last_visit</th>
<th>days_visited</th>
<th>visits</th>
<th>locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>...........</td>
<td>2017-08-07</td>
<td>2019-05-24</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
+-----------+-------------+------------+--------------+--------+-----------+

10 rows in set (46.24 sec)
```
What exactly is slow?

Is 47 seconds slow?
... depends on expectations
- Data Science world it is blazing fast
- Realtime report/dashboard - extremely slow

... Web based queries - users tends to reload page if it is slow
... Leaving MySQL with tons of queries doing the same thing
What to do?

Some ideas:

1. Use index Luke!
2. Table per report
3. Pre-aggregate - table per group of reports
4. Something else
Use index

**But, it is already using index:**

```
  id: 1
  select_type: SIMPLE
  table: cv
  partitions: NULL
  type: index
  possible_keys: FK_client_visit
    key: FK_client_visit
    key_len: 5
    ref: NULL
    rows: 10483873
    filtered: 100.00
    Extra: Using temporary; Using filesort
1 row in set, 1 warning (0.00 sec)

PRIMARY KEY (`id`),
KEY `FK_client_visit_author_id` (`client_id`)
```
Ok, better index: covered index

mysql> alter table client_visit add key comb(client_id, date, service_location_name);
Query OK, 0 rows affected (38.48 sec)
Records: 0  Duplicates: 0  Warnings: 0

    table: cv
  partitions: NULL
     type: index
possible_keys: FK_client_id,comb
     key: comb
  key_len: 776
    ref: NULL
    rows: 10483873
filtered: 100.00
   Extra: Using index; Using temporary; Using filesort

10 rows in set (12.18 sec)
Ok, how large is the table?

```
mysql> show table status like 'client_visit' \G
*************************** 1. row ***************************
    Name: client_visit
    Engine: InnoDB
    Version: 10
   Row_format: Dynamic
    Rows: 10483873
Avg_row_length: 233
  Data_length: 2 452 602 880
Index_length: 1 644 773 376
```

~= 4G on disk, that is it!

24 columns, including `notes` text `description` text etc...
Ok, other options in MySQL?

Create table per each report

Problems

1. Too many tables
2. Hard to maintain
Ok, other options in MySQL?

Pre-aggregate in a table:

- group by client_id + avg, sum, …
- group by date + avg, sum

Final report will do another aggregation if needed

Problems:

1. Some aggregates can’t be re-aggregated
2. Still too many tables
3. Hard to maintain
And it was only the beginning... now this:

```sql
SELECT
cv.client_id as client_id,
min(date) as first_visit,
max(date) as last_visit,
count(distinct date) as days_visited,
count(distinct cv.id) as visits,
count(distinct cp.cpt_code) as procedures,
count(distinct cv.service_location_name) as locations,
sum(billed_amount) as total_billed,
max(billed_amount) as max_price,
avg(billed_amount) as avg_price
FROM
  client_visit cv
join client_procedure cp on cp.encounter_id = cv.encounter_id
join client_procedure_claim cpc on cp.id = cpc.client_procedure_id
join client_claim cc on cc.id = cpc.client_claim_id
GROUP BY client_id
ORDER BY total_billed desc
LIMIT 10
```
4 table JOIN, all large tables

<table>
<thead>
<tr>
<th>client_id</th>
<th>first_visit</th>
<th>last_visit</th>
<th>days_visited</th>
<th>visits</th>
<th>procedures</th>
<th>locations</th>
<th>total_billed</th>
<th>max_price</th>
<th>avg_price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2018-02-14</td>
<td>2019-04-22</td>
<td>64</td>
<td>64</td>
<td>..</td>
<td>..</td>
<td>200K</td>
<td>11K</td>
<td>449.34</td>
</tr>
</tbody>
</table>

10 rows in set (5 min 18.16 sec)
Why MySQL is slow for such queries?

1. Row oriented - even if we retrieve 2 fields it will need to read a row
2. InnoDB organize table by pages (16k page) - will need to read more
3. MySQL will use 1 cpu core per query, not utilizing all cores
Why MySQL is slow for such queries?

Row-oriented DBMS (MySQL)

Column-oriented DBMS

https://clickhouse.yandex/docs/en/
Column Store Databases

MariaDB Columnstore


Tips and Tricks with MariaDB ColumnStore

Wednesday 5:10 PM - 5:35 PM
Column Store Databases

Yandex Clickhouse

https://clickhouse.yandex/

Low Cost Transactional and Analytics With MySQL and Clickhouse, Have Your Cake and Eat It Too!

Wednesday 5:40 PM - 6:05 PM

Clickhouse Features to Blow your Mind

Thursday 11:55 AM - 12:45 PM
Column-store tests

Testing box 1:
- AWS ec2 instance, c5d.4xlarge
- RAM: 32.0 GiB
- vCPU: 16
- Disk: NVMe SSD + EBS

Testing box 2:
- AWS ec2 instance, c5d.18xlarge
- RAM: 144.0 GiB
- vCPU: 72
- Disk: NVMe SSD + EBS
Is it worth using column store: Q1

<table>
<thead>
<tr>
<th></th>
<th>MySQL</th>
<th>Clickhouse</th>
<th>MariaDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time (sec)</td>
<td>46.24</td>
<td>0.754</td>
<td>11.43</td>
</tr>
<tr>
<td>Speed increase</td>
<td></td>
<td>62x</td>
<td>4x</td>
</tr>
<tr>
<td>compared to MySQL</td>
<td></td>
<td>6248%</td>
<td>404%</td>
</tr>
<tr>
<td>(times, %)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

AWS ec2 instance,
c5d.4xlarge
<table>
<thead>
<tr>
<th></th>
<th>MySQL</th>
<th>Clickhouse</th>
<th>MariaDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time (sec)</td>
<td>5 min 18.16 sec</td>
<td>33.83 sec</td>
<td>1 min 2.16 sec</td>
</tr>
<tr>
<td>Speed increase</td>
<td>9x</td>
<td>940%</td>
<td>5x</td>
</tr>
<tr>
<td>compared to MySQL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(times, %)</td>
<td>940%</td>
<td></td>
<td>511%</td>
</tr>
</tbody>
</table>

AWS ec2 instance,
c5d.4xlarge
## Table sizes on disk

<table>
<thead>
<tr>
<th></th>
<th>MySQL</th>
<th>Clickhouse</th>
<th>ColumnStore</th>
</tr>
</thead>
<tbody>
<tr>
<td>client_visit</td>
<td>5,876,219,904</td>
<td>793,976,832</td>
<td>3,606,462,464</td>
</tr>
<tr>
<td>client_procedure</td>
<td>13,841,203,200</td>
<td>2,253,180,928</td>
<td>9,562,865,664</td>
</tr>
<tr>
<td>client_procedure_claim</td>
<td>2,466,250,752</td>
<td>292,007,936</td>
<td>335,683,584</td>
</tr>
<tr>
<td>client_claim</td>
<td>11,710,496,768</td>
<td>2,400,182,272</td>
<td>6,720,749,568</td>
</tr>
<tr>
<td>Total</td>
<td>33,894,170,624</td>
<td>5,739,347,968</td>
<td>20,225,761,280</td>
</tr>
<tr>
<td>Smaller compared to MySQL size (x)</td>
<td></td>
<td>5.91</td>
<td>1.68</td>
</tr>
</tbody>
</table>

*Compression*
Exporting from MySQL

Usually 3 options

1. ETL to Star Schema
2. ETL to flat de-normalized tables
3. Copy / replicate realtime (as is)

I will talk about option 3.
Yandex Clickhouse

```
sudo apt-get install clickhouse-client
clickhouse-server
```
Clickhouse: export from mysql (schema)

https://github.com/Altinity/clickhouse-mysql-data-reader

1. Schema import

```
$ clickhouse-mysql --create-table-sql \ 
    --src-host=mysql-replica-host \ 
    --src-user=export \ 
    --src-password=xxxxxx \ 
    --src-schemas=main \ 
    --src-tables=client_condition,client_procedure,client_visit
```

It will choose partition key and sort key, i.e.

```
ENGINE = MergeTree() PARTITION BY toYYYYMM(created_date) ORDER BY client_id
```

_Beware: enum is not supported (bug)_
Clickhouse: export from MySQL (data)

1. Use clickhouse-mysql-data-reader (slower)
2. Use native Clickhouse MySQL connection:

   INSERT INTO client_procedure_claim SELECT *
   FROM
   mysql('host', 'db', 'client_procedure_claim', 'export', 'xxxxx')

   Ok.

Clickhouse: connect using MySQL client

https://github.com/sysown/proxysql/wiki/ClickHouse-Support

$ wget https://github.com/sysown/proxysql/releases/download/v2.0.4/proxysql_2.0.4-ubuntu18_amd64.deb

$ dpkg -i proxysql_2.0.4-clickhouse-ubuntu18_amd64.deb

$ proxysql --clickhouse-server
Clickhouse: connect using MySQL client

$ mysql -h 127.0.0.1 -P 6032 ...
Admin> SELECT * FROM clickhouse_users;
Empty set (0.00 sec)

Admin> INSERT INTO clickhouse_users VALUES ('clicku','clickp',1,100);
Query OK, 1 row affected (0.00 sec)

Admin> LOAD CLICKHOUSE USERS TO RUNTIME;
Query OK, 0 rows affected (0.00 sec)

Admin> SAVE CLICKHOUSE USERS TO DISK;
Query OK, 0 rows affected (0.01 sec)

https://github.com/sysown/proxysql/wiki/ClickHouse-Support
Clickhouse: connect using MySQL client

mysql -h 127.0.0.1 -P 6090 -uclicku -pclickp
...
Server version: 5.5.30 (ProxySQL ClickHouse Module)

mysql> select client_id, count(cv.id) as visits,
       count(distinct cv.service_location_name) as locations
from client_visit cv
group by client_id order by visits desc limit 10;
+-------------------------+----------+-----------+
| client_id | visits | locations |
+-------------------------+----------+-----------+
... 10 rows in set (0.53 sec)
ProxySQL to Clickhouse - experimental

Some bugs exists:

```sql
select min(date) as first_visit, max(date) as last_visit from client_visit;
ERROR 2013 (HY000): Lost connection to MySQL server during query
```
Clickhouse - joining MySQL source

CREATE VIEW clients_mysql as select * from mysql('mysql_host', 'db', 'client', 'export', 'xxxx');

select count(*) from clients_mysql;

SELECT count(*)
FROM clients_mysql

<table>
<thead>
<tr>
<th>count()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1035284</td>
</tr>
</tbody>
</table>
SELECT cv.client_id as client_id ...
FROM ...
INNER JOIN (SELECT user_id, is_active FROM clients_mysql ) AS c ON cv.client_id = c.user_id 
WHERE c.is_active = 1 ...

10 rows in set. Elapsed: 36.653 sec. Processed 122.58 million rows, 2.30 GB (3.34 million rows/s., 62.72 MB/s.)

CREATE VIEW clients_mysql as select * from mysql('mysql_host', 'db', 'client', 'export', 'xxxx');
MariaDB Column Store Architecture

Application

BI Tool

SQL Client

Custom Big Data App

MariaDB SQL Front End (User Module)

Distributed Query Engine (Performance Module)

Data Storage

Columnar Distributed Data Storage
Local Storage | SAN/NAS | EBS | GlusterFS | CEPH
MariaDB ColumnStore: export from mysql

Schema import - need to create custom schema

`mysqldump --no-data`

… change engine=InnoDB to engine=Columnstore
MariaDB ColumnStore: export from mysql

Schema import - need to create custom schema

`mysqldump --no-data` does not work out of the box

```
$ mcsmysql test < client_visit.sql
ERROR 1069 (42000) at line 25: Too many keys specified; max 0 keys allowed

$ mcsmysql test < client_visit.sql
ERROR 1075 (42000) at line 25: Incorrect table definition; there can be only one auto column and it must be defined as a key
```
MariaDB ColumnStore: export from mysql (data)

Fastest way -
1. mysql > select into outfile …
2. $ cpimport …

Easiest way:
1. Import into InnoDB locally (columnstore includes MySQL server)
2. Run “insert into columnstore_table select * from innodb_table”
MariaDB ColumnStore: Joining MySQL source

1. Export using InnoDB storage engine
2. JOIN across engines

SELECT ...
FROM
  client_visit cv
  JOIN client_procedure cp ON cp.encounter_id = cv.encounter_id
  JOIN client_procedure_claim cpc ON cp.id = cpc.client_procedure_id
  JOIN client_claim cc ON cc.id = cpc.client_claim_id
  JOIN client_innodb c ON cv.client_id = c.user_id
WHERE c.is_active = 1
GROUP BY client_id
ORDER BY total_billed DESC
LIMIT 10;
Clickhouse: replication from MySQL

clickhouse-mysql --src-server-id=1 --src-resume --src-wait
--nice-pause=1 --src-host=10.0.0.142 --src-user=chreader
--src-password=pass --src-tables=wiki.pageviews
--dst-host=127.0.0.1 --pump-data --csvpool
ColumnStore: replication from MySQL

Coming soon

https://jira.mariadb.org/browse/MCOL-498
https://jira.mariadb.org/browse/MCOL-593
Update / delete - MariaDB ColumnStore

DMLs are usually the slowest in Columnar Stores

Single row:

MariaDB [vh]> update client set is_active = 0 where user_id = 3216031;
Query OK, 1 row affected (0.156 sec)
Rows matched: 0  Changed: 0  Warnings: 0
Update / delete - Clickhouse

Implemented as “Alter table update” (mutations)
Asynchronous


Single row:

:) ALTER TABLE client
    UPDATE is_active = 0 WHERE user_id = 3216031

Ok.

0 rows in set. Elapsed: 0.006 sec.
When NOT to use Column Store db (1)

Single row (full row) by id: select * from client_claim where id = <num>;

MySQL: 1 row in set (0.00 sec)
Columnstore: 1 row in set (0.039 sec)
Clickhouse: 1 rows in set. Elapsed: 0.023 sec.
When NOT to use Column Store db (2)

When we will be using index in MySQL + limit

    SELECT *
    FROM client_claim
    WHERE place_of_service = 'OFFICE'
    ORDER BY received_date DESC
    LIMIT 10

MySQL: 10 rows in set (0.00 sec)
Columnstore: 10 rows in set (20.838 sec)
Clickhouse: 10 rows in set. Elapsed: 1.148 sec

ERROR 1815 (HY000): Internal error: IDB-2015: Sorting length exceeded. Session variable max_length_for_sort_data needs to be set higher.
When NOT to use Column Store db (2)

mysql> explain SELECT * FROM client_claim WHERE place_of_service = 'OFFICE' ORDER BY received_date DESC LIMIT 10\G

*************************** 1. row ***************************
id: 1
select_type: SIMPLE
table: client_claim
partitions: NULL
type: ref
possible_keys: place_of_service_received_date
key: place_of_service_received_date
key_len: 768
ref: const
rows: 13072594
filtered: 100.00
Extra: Using where
Other idea: daily online snapshots

- Size is columnar store DB is significantly smaller
- We can load daily snapshots
  - i.e. store 30 days of data that can be queried (without restore from backup)
Summary: Clickhouse

Advantages
- Fastest queries
- Very efficient storage

Disadvantages
- For JOINs needs RAM to store all JOINed tables
- No native MySQL protocol (only via ProxySQL)
- No standard sql support (data types, etc)
Summary: MariaDB ColumnStore

Advantages
● Native MySQL protocol - easier to integrate
● Native shared nothing cluster

Disadvantages
● Slower queries
Conclusion (1)

Clickhouse and MariaDB ColumnStore can help optimizing slow reporting queries

a. Those queries can be slow by design in MySQL
b. MySQL is good for lots of small queries rather than some smaller ones
c. MySQL will not be able to utilize all hardware resources
Conclusion (2)

We can see 5x-10x to 60x performance increase for some queries

- From 5 minutes for 30 seconds in Clickhouse
Conclusion (3)

Both systems can be used together with MySQL

a. Sync data quickly
b. Replicate realtime
Thank you!

Alexander Rubin

https://www.linkedin.com/in/alexanderrubin