# ClickHouse Anti-Patterns. Learning from Users' Mistakes

Mikhail Filimonov

# Anti-patterns?

Patterns - something we do often

Anti-patterns - something we do often (but regret later). :)

# Who can guess what is the problem?

- "when I run some mutation clickhouse loses the zookeeper connection"
- "duplicates are removed very slowly by ReplacingMergeTree"
- "backup of a relatively small table takes dozens of minutes"
- "zookeeper often breaks the connection (due to zxid overflow)"
- "no free inodes on the filesystem"
- "clickhouse collects a lot of inactive parts, and can't remove them later."
- "OPTIMIZE FINAL is very slow on relatively small table."
- "replication issues (high zookeeper traffic, delays)"
- "too many parts"
- "inserts go slower when more data collected"
- "clickhouse-server starting up takes more than 30 minutes."
- "Too many partitions for single INSERT block (more than 100)"

# So many problems...

Because of ...

**bad PARTITION BY!**

# And the anti-pattern is …

- PARTITION BY (tenantid)
- PARTITION BY (tenantid, toStartOf...(timestamp))

# Where is the problem?

- Partition is physical data separation!
- Every partition is stored in a separate directory on the filesystem, and needs to be processed separately from others - in filesystem / in zookeeper / by ALTER commands, etc.

# Where is the problem: Inserts

- How many wallets do you need to have to store $5000 in cash?
- If you get $10 extra - how many wallets you will put that in?

Insert fan-out effect:

- You can put $100 into 100 wallets once a day.
- But what if you're working at a cash desk and need to accept $1 and distribute it to 100 wallets every second?...

In other words - if you have real-time inserts the best option is when inserted data lands in one (or few) partitions.

So you can't have partitioning by the tenant and real-time inserts simultaneously.

# Where is the problem: selects

Once you need to take 1% from every wallet - how many operations you will need to do?

# “But I have only X tenants”

Be an optimist: you will grow :)

Partition by tenantid is not easy to scale.

# How to do it better?

Remove the tenant_id from the partition key.

Put it at the beginning of ORDER BY!

This way you will be able to scale your system linearly!

# How to do it worse?

"I can just create a separate table/database per tenant!" (c) unknown user.

Picking between those 2: "partition by" is the lesser evil

# But... Does it mean 'absolutely no'? Never ever?

Sometimes we are forced to use anti-patterns.

Sometimes partition by tenant_id is the only acceptable option. (usually because of some contract requirements like data privacy).

In that case, you can use it, but

- test your solution with more tenants to know your capacity
- be aware of the potential problems you can hit, and set up a good monitoring
- have plan B (what will you do if you will grow beyond the capacity

# Picking the partitioning

Do you have less than several million rows in the table (or several gigabytes)?

- You don't need partitioning!

Do you have some time-series data?

- Use yearly / monthly / weekly / daily / hourly partitioning, depending on how much data you get and for how long you need to store it.
- Your single partition size should be from a few gigabytes to a few hundreds of gigabytes.
- Sometimes you can partition by something like tenant_category, or by tenant_retention_policy additionally to the timestamp-based partitioning.

Do you use Aggregating/Collapsing/Summing/Replacing and want to make collapsing happen faster?

- your single partition size should be from 100 Mb to 30 Gb. Sometimes you can use something like modulo (userid % 100).

# Don't do this:

- PARTITION BY (timestamp) -- how many timestamps do you have?
- PARTITION BY (field / 10) -- are you sure that floating-point number in the partition key is a good idea?
- PARTITION BY (src_ip, target_ip, toYYYYMM(timesamp)) -- how many IP combinations do you have?
- PARTITION BY (userid, transaction_type)
- PARTITION BY (unique_id) etc.

# Thank you!

Mikhail Filimonov.
Altinity 2022