Rebalancing Shards in Clickhouse

for self-hosted OSS clusters

By
Shivji Kumar Jha & Pranav Mehta

Safe Harbour Statement

The views, opinions, and conclusions presented in these slides are solely my own and do not reflect the official stance, policies, or perspectives of my employer or any affiliated organization. Any statements made are based on my personal experiences and research and should not be interpreted as official guidance or endorsement.

ABOUT US

Shivji Kumar Jha
Staff Engineer/Lead
CPaaS Data Platform, Nutanix

- Interests: Databases, Streaming, Infra, App Backends
- Contributed code to MySQL, Pulsar, Clickhouse
- Excited about Open-Source Software & Communities
- Regular Speaker(29*), See tinyurl.com/shiv-slides
- Ping : linkedin.com/in/shivjijha/

Pranav Mehta
Software Engineer (MTS4)
CPaaS Team, Nutanix

- Passionate About Distributed Systems
- Excited about:
  - Clickhouse, NATS, P2P, WASM, Linux
- Ping: linkedin.com/in/pranavmehta94/

Table Of contents

- What is Data Rebalancing
- A survey from the neighborhood
- Rebalancing in Clickhouse
- Our experiments tailored to our needs
- Future Work / Wishlist

Love Clickhouse

Simple (single executable) and speaks SQL

Fast, Very fast (see clickbench). Written in C++ like DBs should be ☺

Huge adoption & a large community of users & contributors

Very popular (Github: 41.6K *, 7.4K forks)

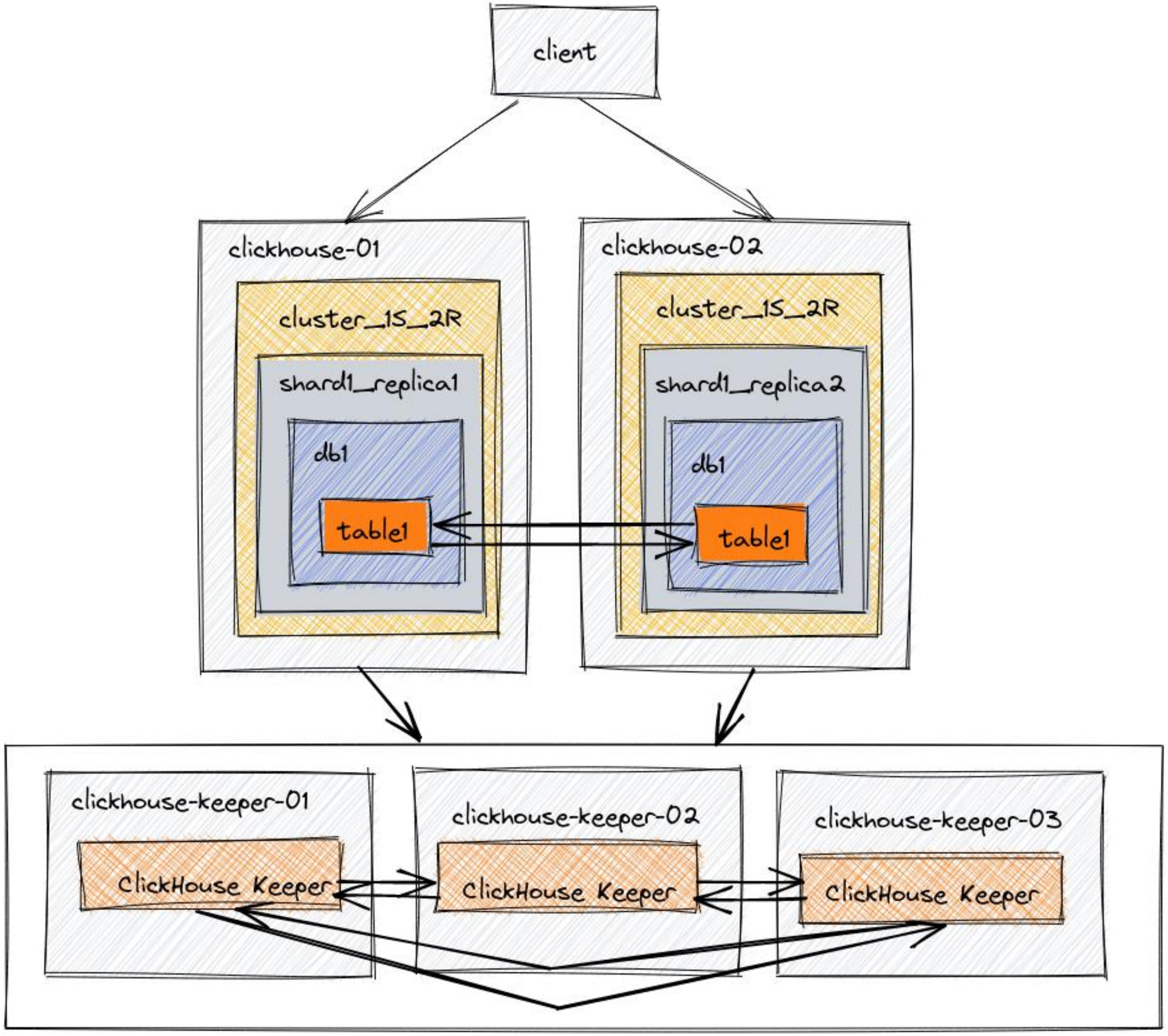
Flexible : Permissible, lots of configs, choices of deployments etc

Tons of integrations (Engines, views, I/O formats etc)

And.. Growing fast!

Rebalancing Shards

Clickhouse: Shards & Replicas



https://clickhouse.com/docs/architecture/replication

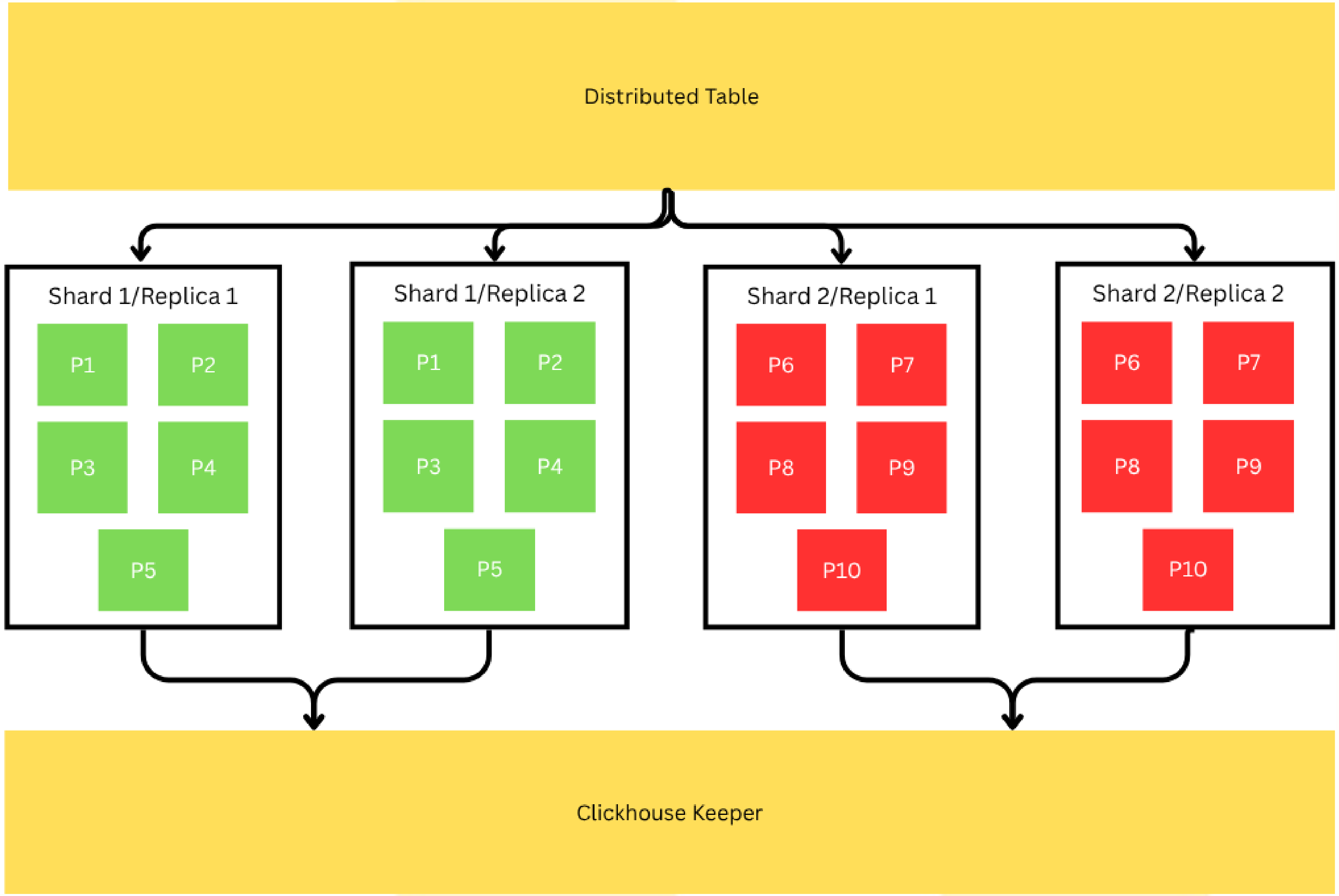
DATA
REBALANCING

Why?

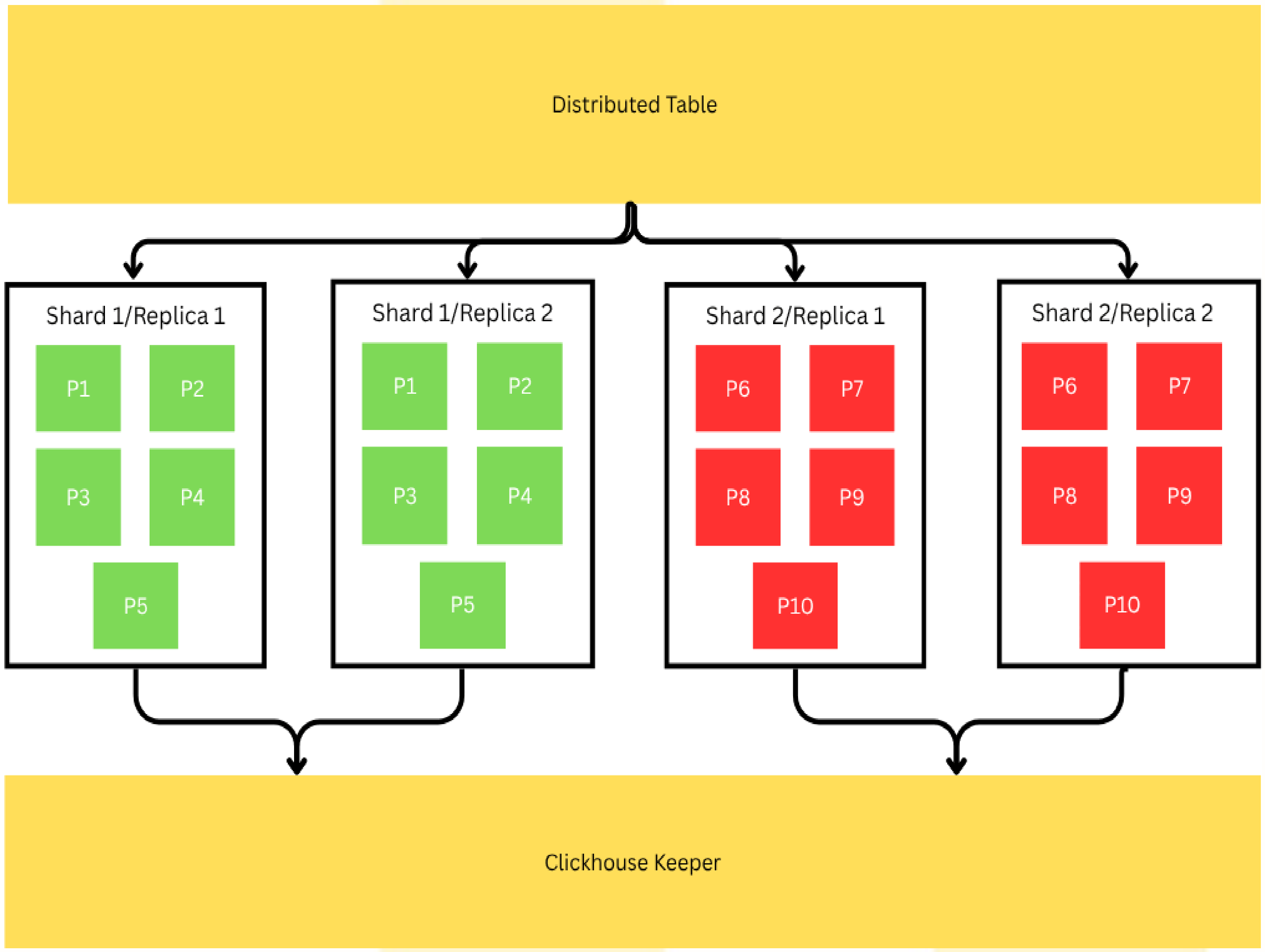
- **Avoid hotspots** - Prevent some nodes from becoming overloaded while others sit idle.
- **Optimal use of resources** – evenly spread disk usage, faster querying
- **Maintain performance** – Ensure consistent query speed and throughput.
- **Enable scaling** – Distribute data evenly when adding or removing nodes

Rebalancing Shards in Clickhouse

Clickhouse Deployment



- 4 node cluster
  - 2 shard
  - 2 replica
- ReplicatedMergeTree
- Clickhouse Keeper



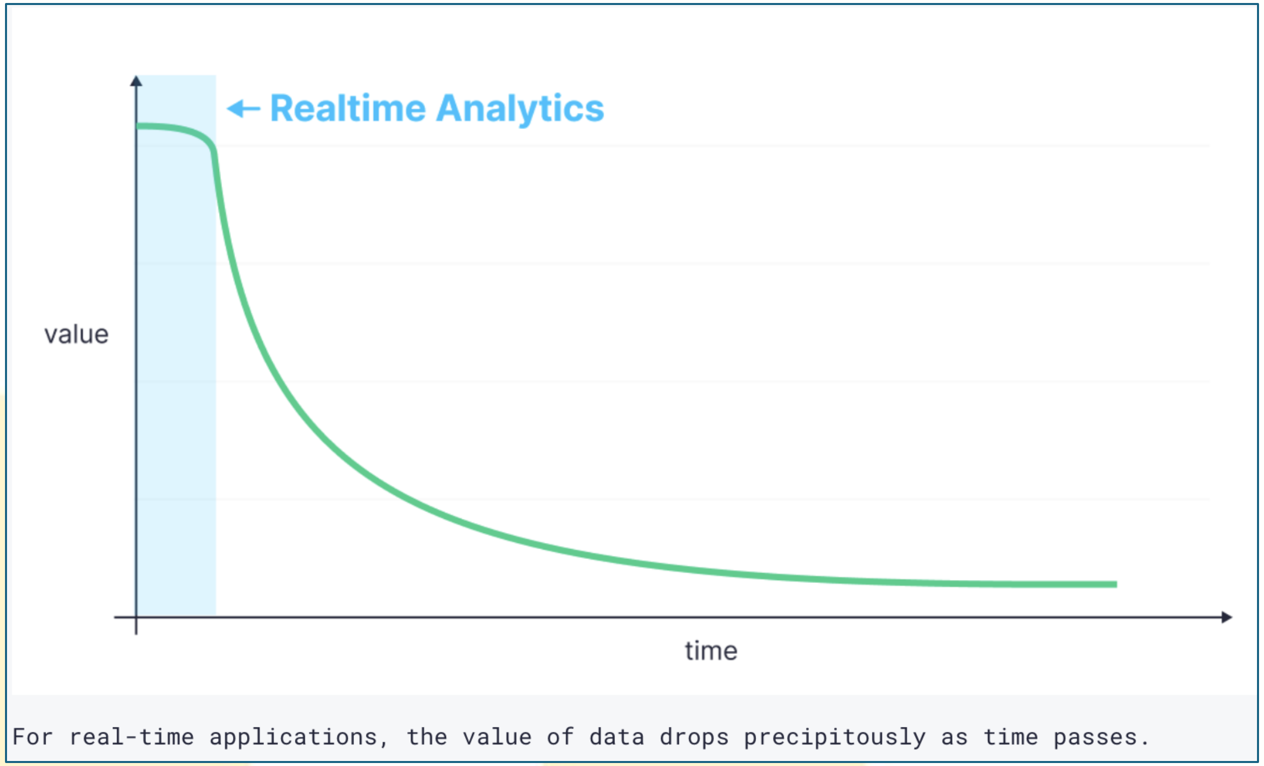
Current Options

Option 0 : Use clickhouse cloud

- All data on S3 or object storage of choice
- Compute-compute segregation
- If you use the self-hosted version?
  - You can get quite far without needing any of this!
  - there are some other options

Option 1: Let TTL do its magic over time

- Do Nothing!
- Realtime data looses value over time.
  - Eg: supply chain tracking, fraud detection, live call centre feedback etc
- If you have a time series:
  - Old data expires & rebalances over time.
  - Old data is rolled up into summary (MVs)

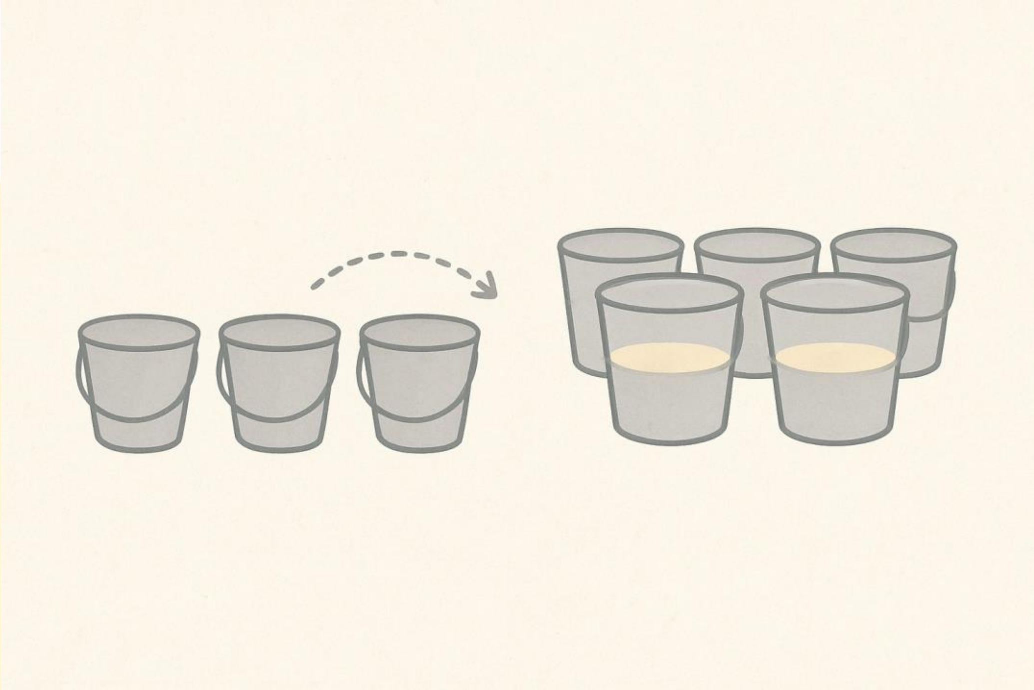


For real-time applications, the value of data drops precipitously as time passes.

https://www.tinybird.co/blog-posts/definition-of-real-time-data

Option 2: Move data to new fat cluster

- **Export data** using INSERT FROM SELECT into new cluster
  - Assuming you have extra resources for 2 clusters
  - Pros: Works even without partitioning.
  - Cons:
    - Poor performance on large datasets;
    - High IO and network overhead



Option 3: Write to new shard temporarily

- Modify the cluster to write exclusively to the new shard until balance is restored
  - Pros: Easy
  - Cons:
    - Causes potential load imbalance;
    - Doesn’t rebalance existing data. Queries may still be skewed

Option 4: Skew ingest to new shards

- Adjust the distributed table's shard **weights** to bias new writes to a new shard.
    - Pros: Easy; no need to change write targets.
    - Cons:
        - Causes potential load imbalance
        - Doesn't rebalance existing data. Queries still skewed

Option 5: Move Partitions Manually

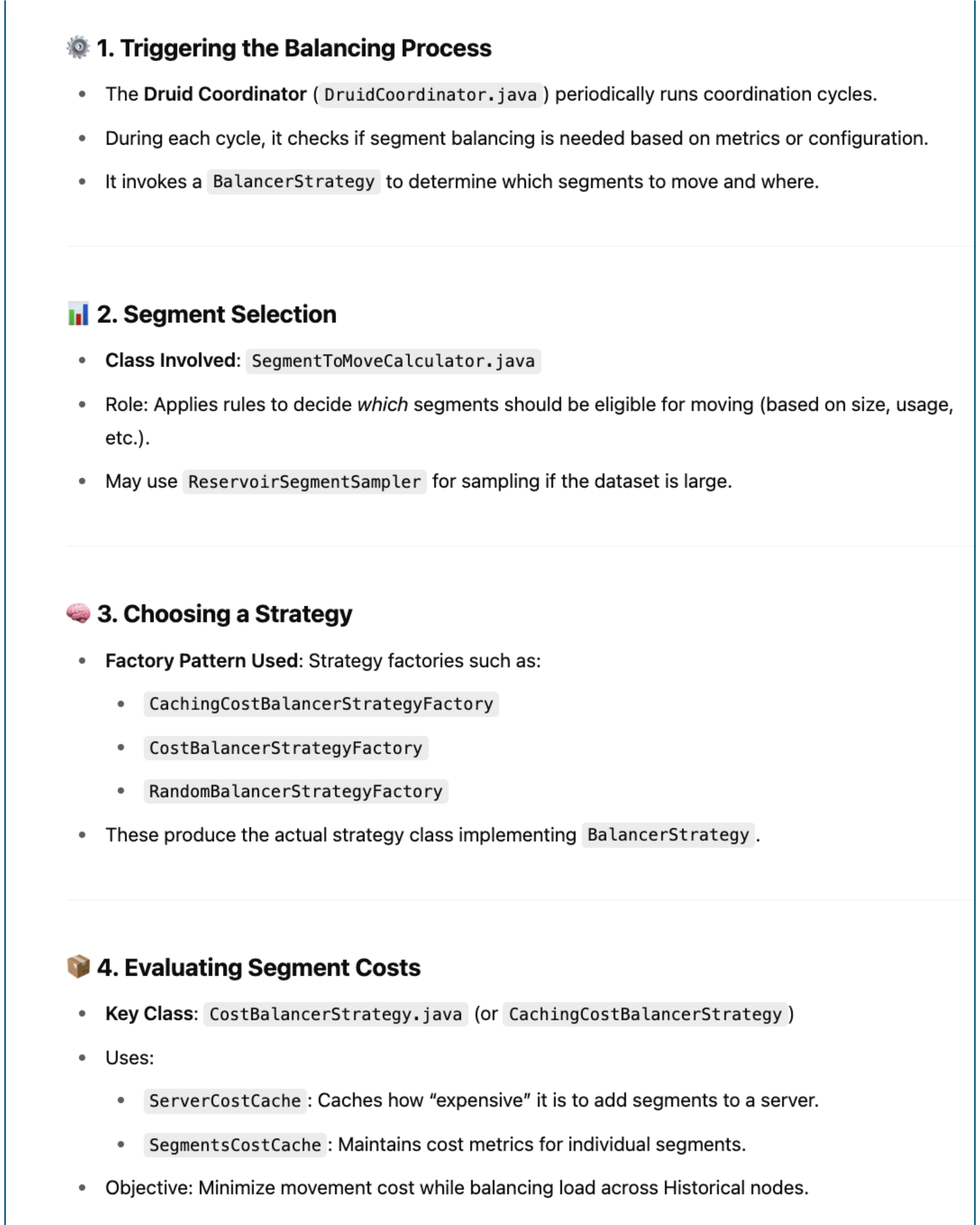
- For existing partitioned data,
  - **Detach partition** from loaded node
  - **Move** them to the new node, and
  - **Reattach** to the less loaded destination
- Pros:
  - More precise; avoids rewriting all data.
- Cons:
  - Manual and operationally intensive.
  - Inevitable maintenance window with partial responses / downtime.

https://clickhouse.com/docs/sql-reference/statements/alter/partition

Inspirations from the neighborhood

1. Rebalancing with Objects Backup

Doc: https://druid.apache.org/docs/latest/design/coordinator/#balancing-segments-in-a-tier
Code: https://github.com/apache/druid/blob/master/server/src/main/java/org/apache/druid/server/coordinator/balancer



⚙️ 1. Triggering the Balancing Process

- The **Druid Coordinator** (`DruidCoordinator.java`) periodically runs coordination cycles.
- During each cycle, it checks if segment balancing is needed based on metrics or configuration.
- It invokes a `BalancerStrategy` to determine which segments to move and where.

📊 2. Segment Selection

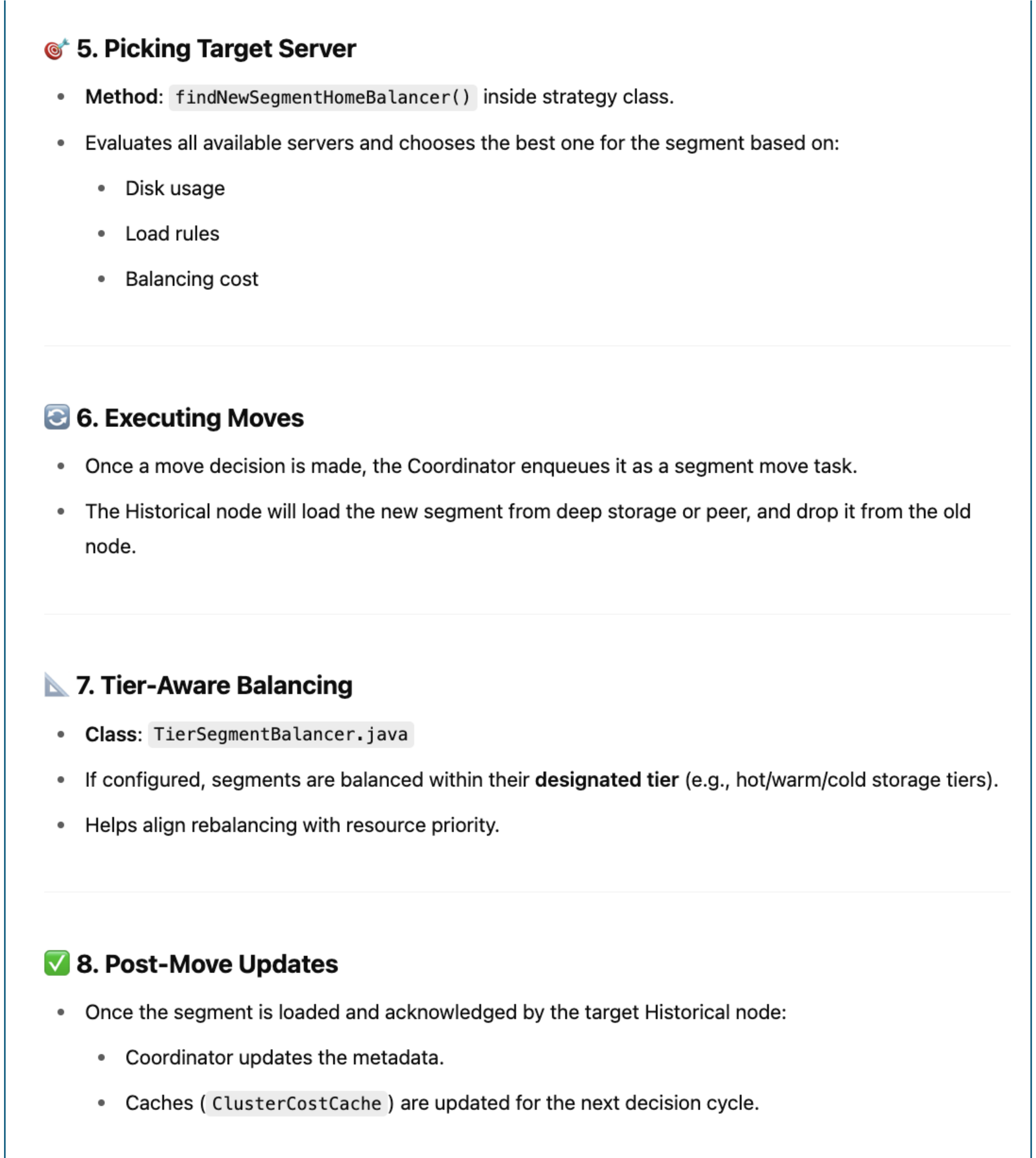
- **Class Involved**: `SegmentToMoveCalculator.java`
- Role: Applies rules to decide *which* segments should be eligible for moving (based on size, usage, etc.).
- May use `ReservoirSegmentSampler` for sampling if the dataset is large.

🧠 3. Choosing a Strategy

- **Factory Pattern Used**: Strategy factories such as:
  - `CachingCostBalancerStrategyFactory`
  - `CostBalancerStrategyFactory`
  - `RandomBalancerStrategyFactory`
- These produce the actual strategy class implementing `BalancerStrategy` .

📦 4. Evaluating Segment Costs

- **Key Class**: `CostBalancerStrategy.java` (or `CachingCostBalancerStrategy`)
- Uses:
  - `ServerCostCache` : Caches how "expensive" it is to add segments to a server.
  - `SegmentsCostCache` : Maintains cost metrics for individual segments.
- Objective: Minimize movement cost while balancing load across Historical nodes.



🎯 5. Picking Target Server

- **Method**: `findNewSegmentHomeBalancer()` inside strategy class.
- Evaluates all available servers and chooses the best one for the segment based on:
  - Disk usage
  - Load rules
  - Balancing cost

🔄 6. Executing Moves

- Once a move decision is made, the Coordinator enqueues it as a segment move task.
- The Historical node will load the new segment from deep storage or peer, and drop it from the old node.

📐 7. Tier-Aware Balancing

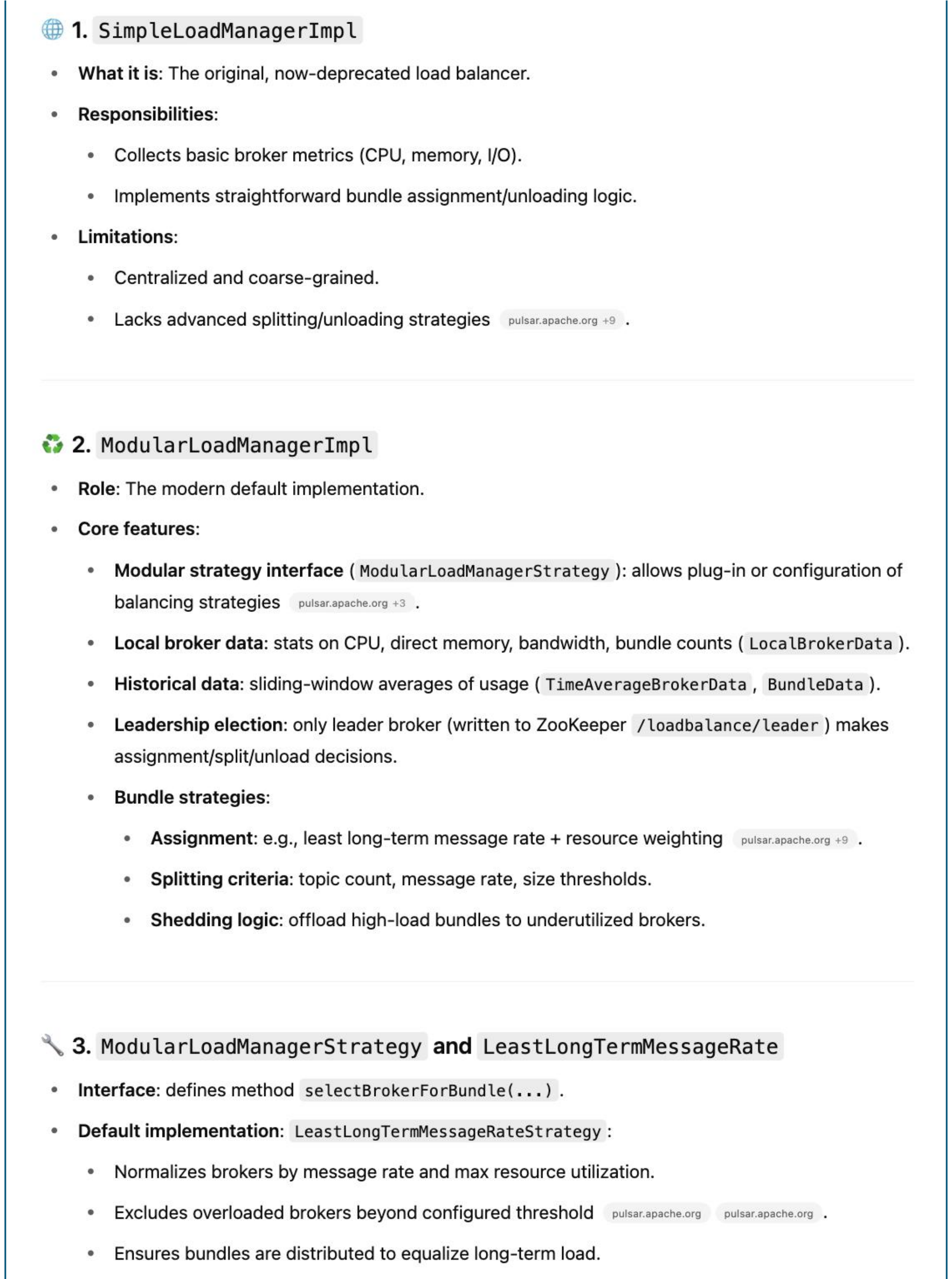
- **Class**: `TierSegmentBalancer.java`
- If configured, segments are balanced within their **designated tier** (e.g., hot/warm/cold storage tiers).
- Helps align rebalancing with resource priority.

✅ 8. Post-Move Updates

- Once the segment is loaded and acknowledged by the target Historical node:
  - Coordinator updates the metadata.
  - Caches (`ClusterCostCache`) are updated for the next decision cycle.

2. Rebalancing without Objects

Doc: https://pulsar.apache.org/docs/next/develop-load-manager/
Proposal: https://github.com/apache/pulsar/issues/16691
https://github.com/apache/pulsar/tree/master/pulsar-broker/src/main/java/org/apache/pulsar/broker/loadbalance



🌐 1. `SimpleLoadManagerImpl`

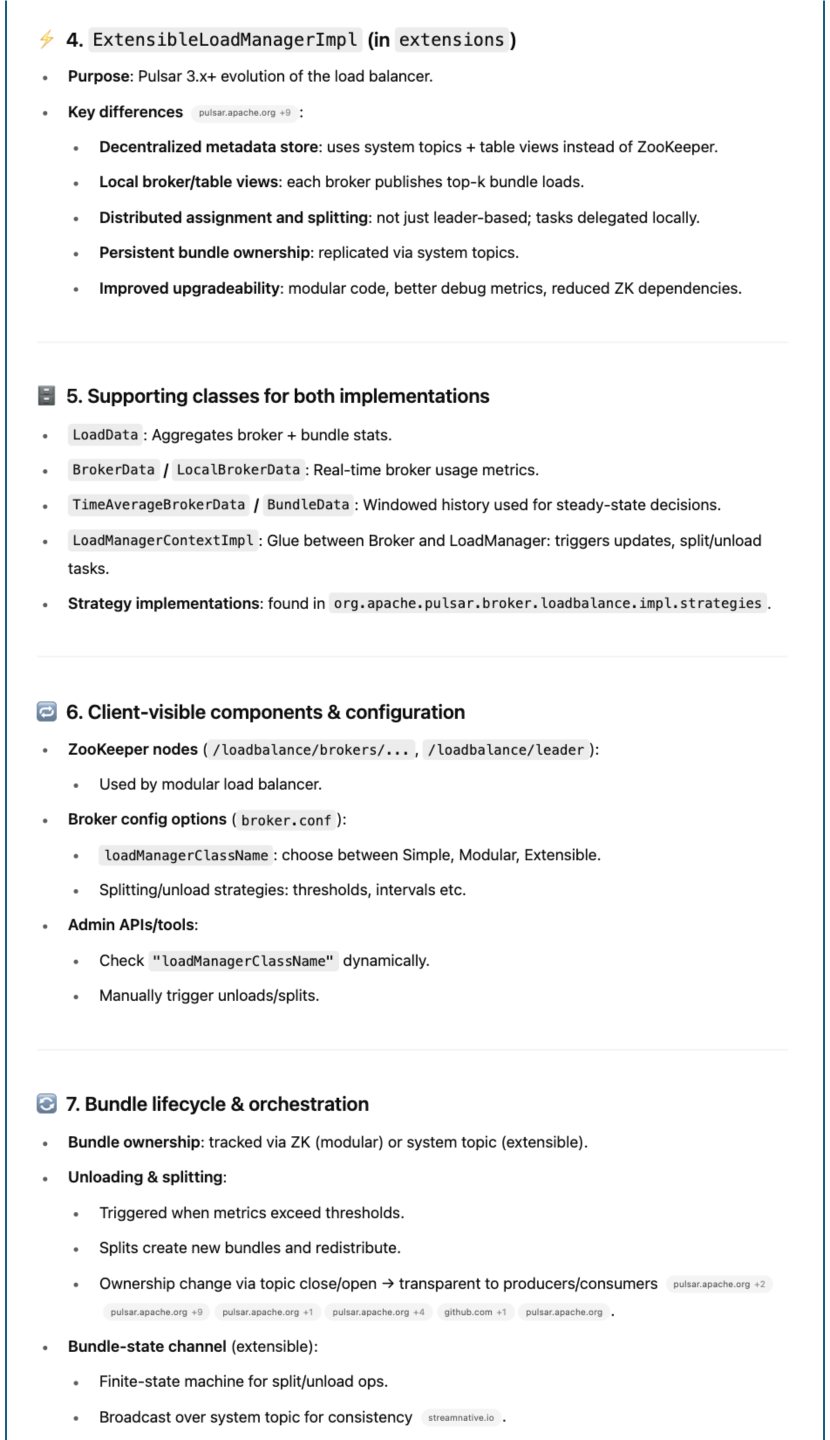
- **What it is**: The original, now-deprecated load balancer.
- **Responsibilities**:
  - Collects basic broker metrics (CPU, memory, I/O).
  - Implements straightforward bundle assignment/unloading logic.
- **Limitations**:
  - Centralized and coarse-grained.
  - Lacks advanced splitting/unloading strategies pulsar.apache.org +9 .

♻️ 2. `ModularLoadManagerImpl`

- **Role**: The modern default implementation.
- **Core features**:
  - **Modular strategy interface** (`ModularLoadManagerStrategy`): allows plug-in or configuration of balancing strategies pulsar.apache.org +3 .
  - **Local broker data**: stats on CPU, direct memory, bandwidth, bundle counts (`LocalBrokerData`).
  - **Historical data**: sliding-window averages of usage (`TimeAverageBrokerData` , `BundleData`).
  - **Leadership election**: only leader broker (written to ZooKeeper `/loadbalance/leader`) makes assignment/split/unload decisions.
  - **Bundle strategies**:
    - **Assignment**: e.g., least long-term message rate + resource weighting pulsar.apache.org +9 .
    - **Splitting criteria**: topic count, message rate, size thresholds.
    - **Shedding logic**: offload high-load bundles to underutilized brokers.

🔧 3. `ModularLoadManagerStrategy` and `LeastLongTermMessageRate`

- **Interface**: defines method `selectBrokerForBundle(...)` .
- **Default implementation**: `LeastLongTermMessageRateStrategy` :
  - Normalizes brokers by message rate and max resource utilization.
  - Excludes overloaded brokers beyond configured threshold pulsar.apache.org pulsar.apache.org .
  - Ensures bundles are distributed to equalize long-term load.



⚡ 4. `ExtensibleLoadManagerImpl` (in `extensions`)

- **Purpose**: Pulsar 3.x+ evolution of the load balancer.
- **Key differences** pulsar.apache.org +9 :
  - **Decentralized metadata store**: uses system topics + table views instead of ZooKeeper.
  - **Local broker/table views**: each broker publishes top-k bundle loads.
  - **Distributed assignment and splitting**: not just leader-based; tasks delegated locally.
  - **Persistent bundle ownership**: replicated via system topics.
  - **Improved upgradeability**: modular code, better debug metrics, reduced ZK dependencies.

🗄️ 5. Supporting classes for both implementations

- `LoadData` : Aggregates broker + bundle stats.
- `BrokerData` **/** `LocalBrokerData` : Real-time broker usage metrics.
- `TimeAverageBrokerData` **/** `BundleData` : Windowed history used for steady-state decisions.
- `LoadManagerContextImpl` : Glue between Broker and LoadManager: triggers updates, split/unload tasks.
- **Strategy implementations**: found in `org.apache.pulsar.broker.loadbalance.impl.strategies` .

🔁 6. Client-visible components & configuration

- **ZooKeeper nodes** (`/loadbalance/brokers/...` , `/loadbalance/leader`):
  - Used by modular load balancer.
- **Broker config options** (`broker.conf`):
  - `loadManagerClassName` : choose between Simple, Modular, Extensible.
  - Splitting/unload strategies: thresholds, intervals etc.
- **Admin APIs/tools**:
  - Check `"loadManagerClassName"` dynamically.
  - Manually trigger unloads/splits.

🔄 7. Bundle lifecycle & orchestration

- **Bundle ownership**: tracked via ZK (modular) or system topic (extensible).
- **Unloading & splitting**:
  - Triggered when metrics exceed thresholds.
  - Splits create new bundles and redistribute.
  - Ownership change via topic close/open → transparent to producers/consumers pulsar.apache.org +2 pulsar.apache.org +9 pulsar.apache.org +1 pulsar.apache.org +4 github.com +1 pulsar.apache.org .
- **Bundle-state channel** (extensible):
  - Finite-state machine for split/unload ops.
  - Broadcast over system topic for consistency streamnative.io .

The (ideal) rebalancing algo:

- Keeps data spread in a way that help parallelize queries
  - If two parts are more likely to be picked by a query, keep apart.
- Evaluates cost based on:
  - Server capacity (% disk, cpu, ram, network i/o etc in use)
  - Demand for data : read/write throughput etc
  - Recent movements so we don’t keep moving same part over and over.
  - Historical averaged load data
- An assortment of algos to choose from!
  - And maybe an easy plug-and-play to add custom rebalance logic?

Clickhouse- ongoing work

Self-balancing architecture

Proposal:

https://github.com/ClickHouse/ClickHouse/issues/13574

Introduces live part movement

ALTER TABLE table_name MOVE PARTITION|PART partition_expr TO SHARD 'shard_id’

Self-balancing architecture

- Mark part to be moved with a unique identifier
- Copy data to the destination shard
- Remove data from source shard
- When reading data, use the unique identifier to discard duplicate part
- Open Issues #49574 , #40814, #40189, #40188
- Disabled!
- nvartolomei.com/rebalancing-data-shared-nothing

https://github.com/ClickHouse/ClickHouse/pull/17871

Self-balancing architecture – Challenges

- Race condition leading to duplicate data in SELECT - #49574
  - proposal made #50777
- Tasks Cleanup- #40814
- Data loss along with zero copy replication- #40189, #40188
- Assigning UUID to existing parts
- Disabled!

https://nvartolomei.com/rebalancing-data-shared-nothing/

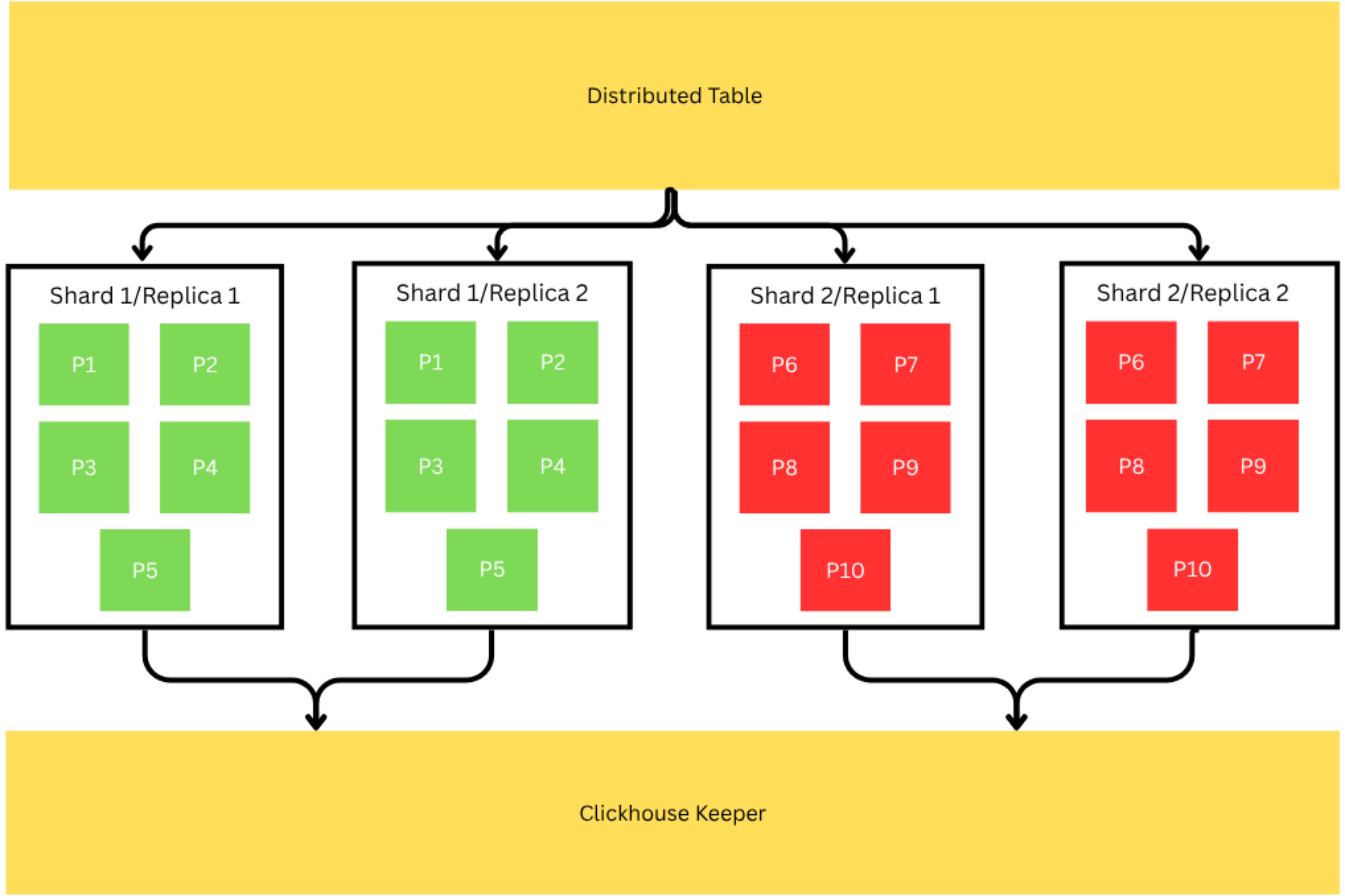
Self-balancing architecture – Our efforts

- https://github.com/ClickHouse/ClickHouse/issues/65633
- https://github.com/ClickHouse/ClickHouse/pull/68020

Shout out: Tarun Annapareddy

Clickhouse Proposal

https://github.com/ClickHouse/ClickHouse/issues/45766

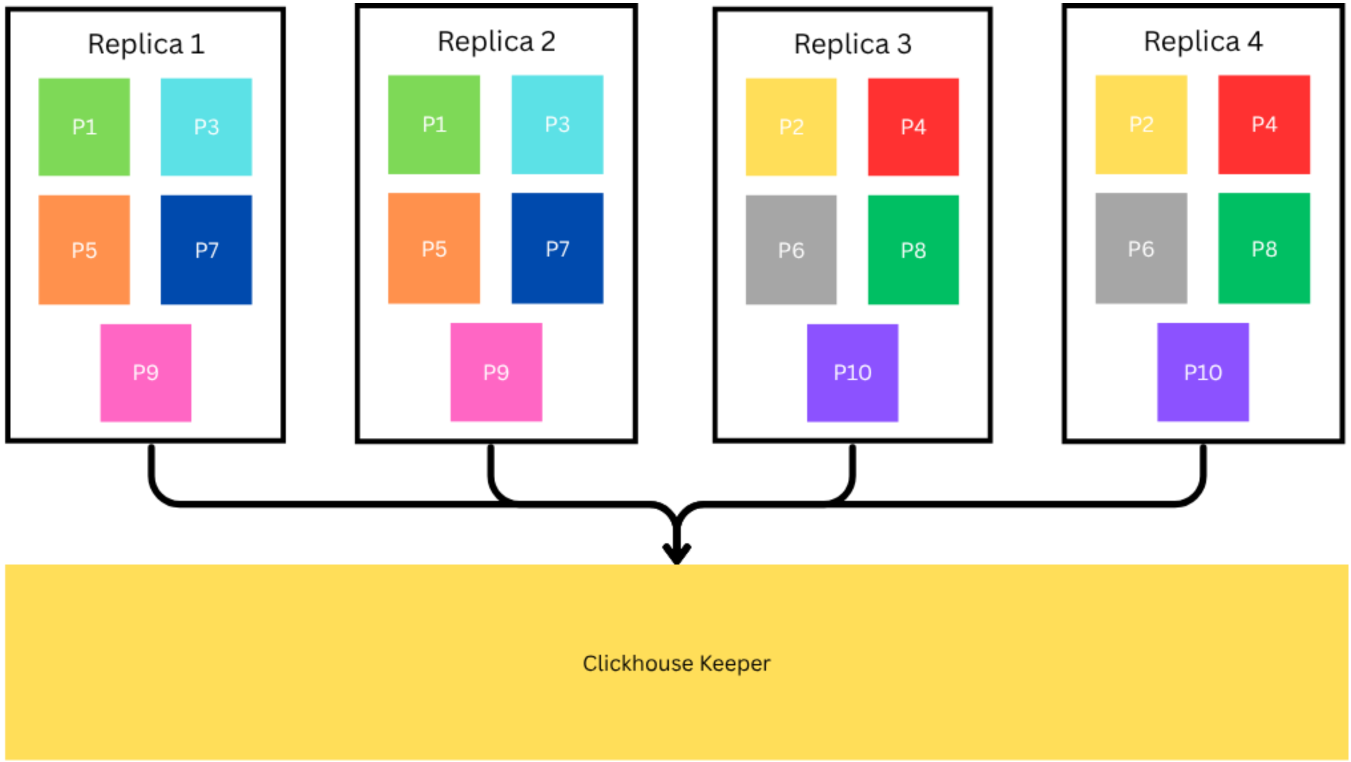


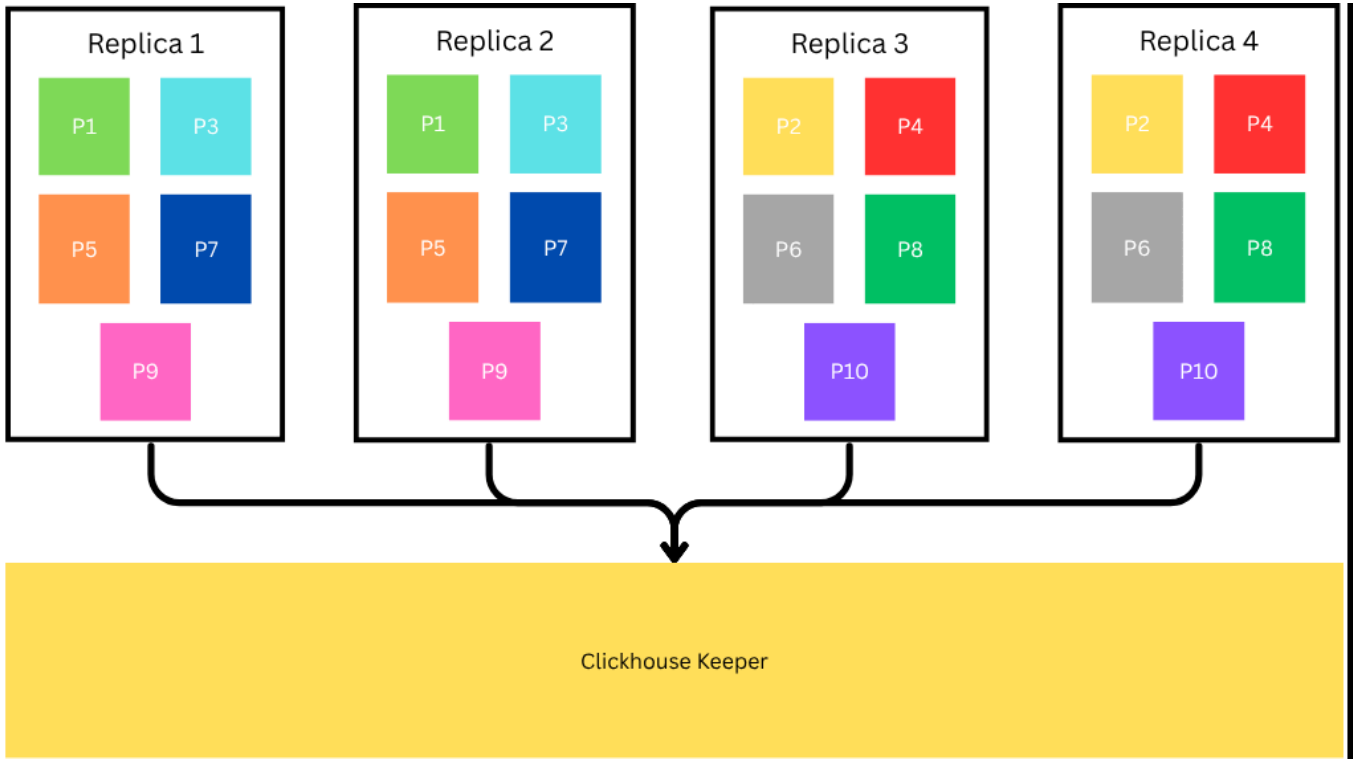
ReplicatedMergeTree

- 2 shards
- 2 replicas

ReplicatedMergeTree Extended

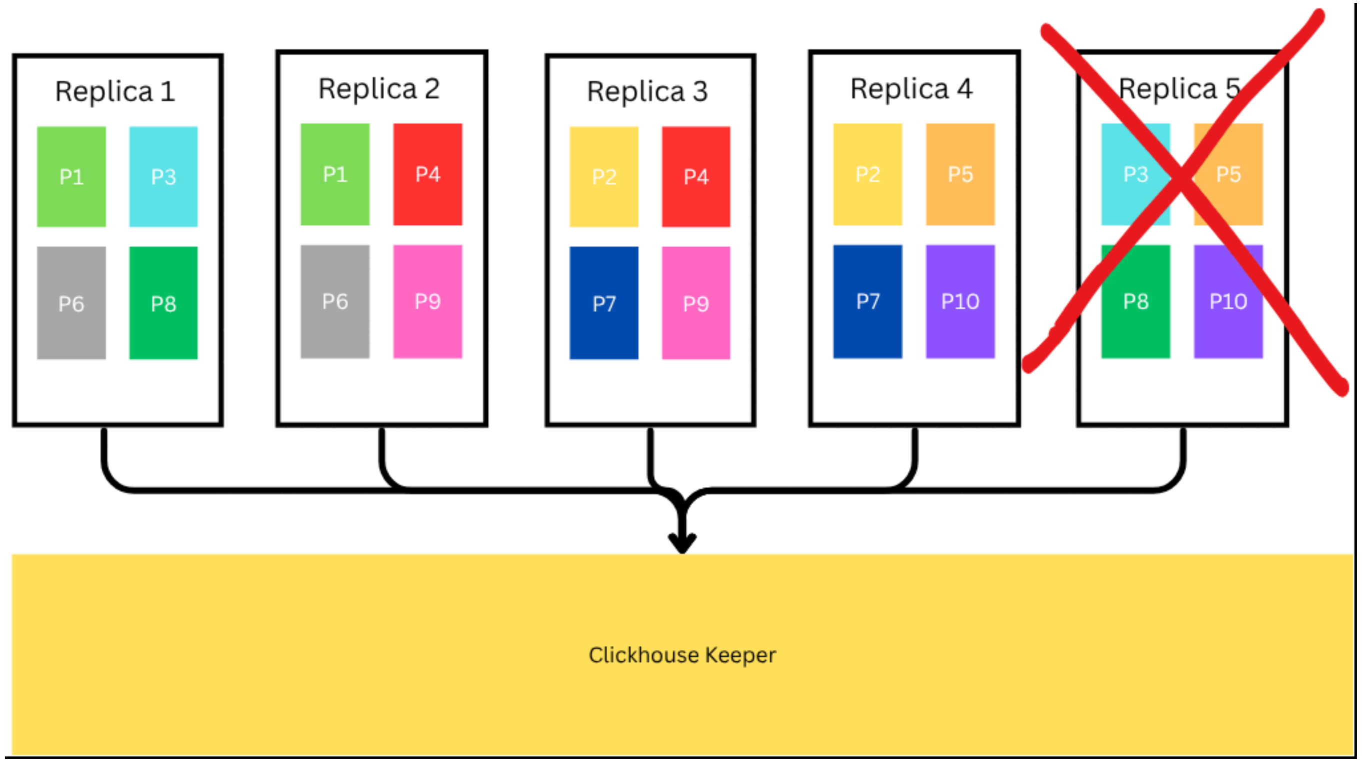
- Replication Factor: 2





ReplicatedMergeTree Extended

- Replication Factor: 2
- Rebalancing after adding a Node



ReplicatedMergeTree Extended

- Replication Factor: 2
- Rebalancing after removing a Node

Open PR

- Basic SELECT/INSERT support
- Basic re-sharding support with a naïve algorithm
- Table Settings
  - Cluster
  - cluster_replicaton_factor
- Query Settings
  - cluster_query_shards
- Queries
  - SYSTEM SYNC REPLICA CLUSTER
  - SYSTEM DROP CLUSTER REPLICA
- System Table
  - system.cluster_partitions

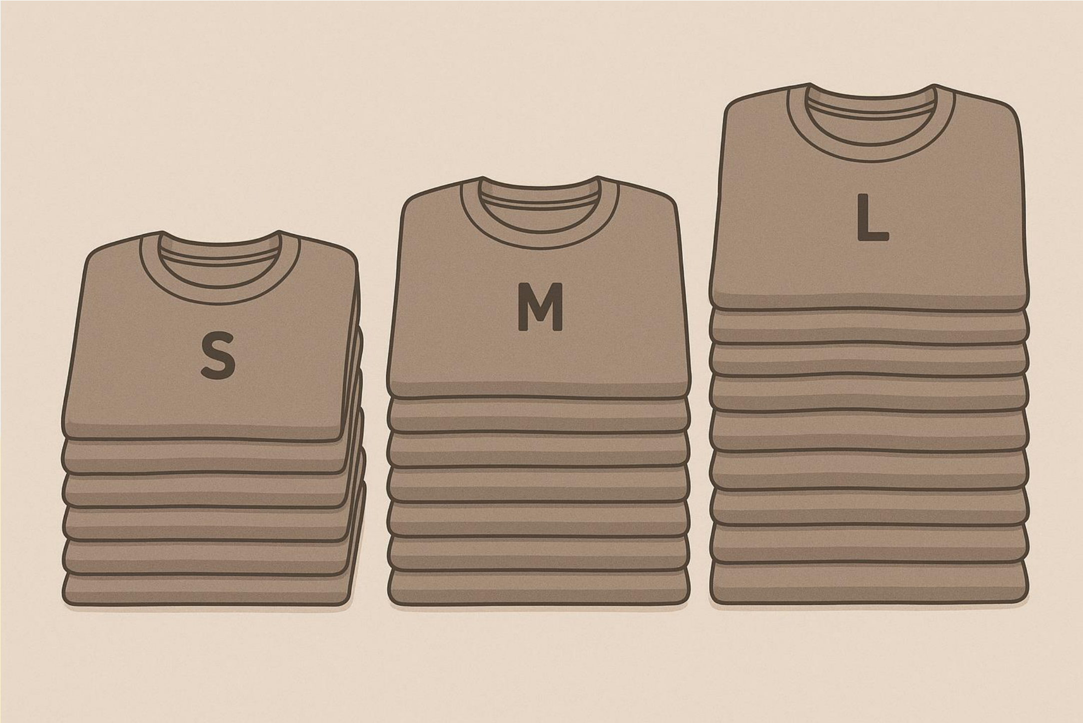
https://github.com/ClickHouse/ClickHouse/pull/58132

Our Experiments

Tailored to our needs

Our Usecase

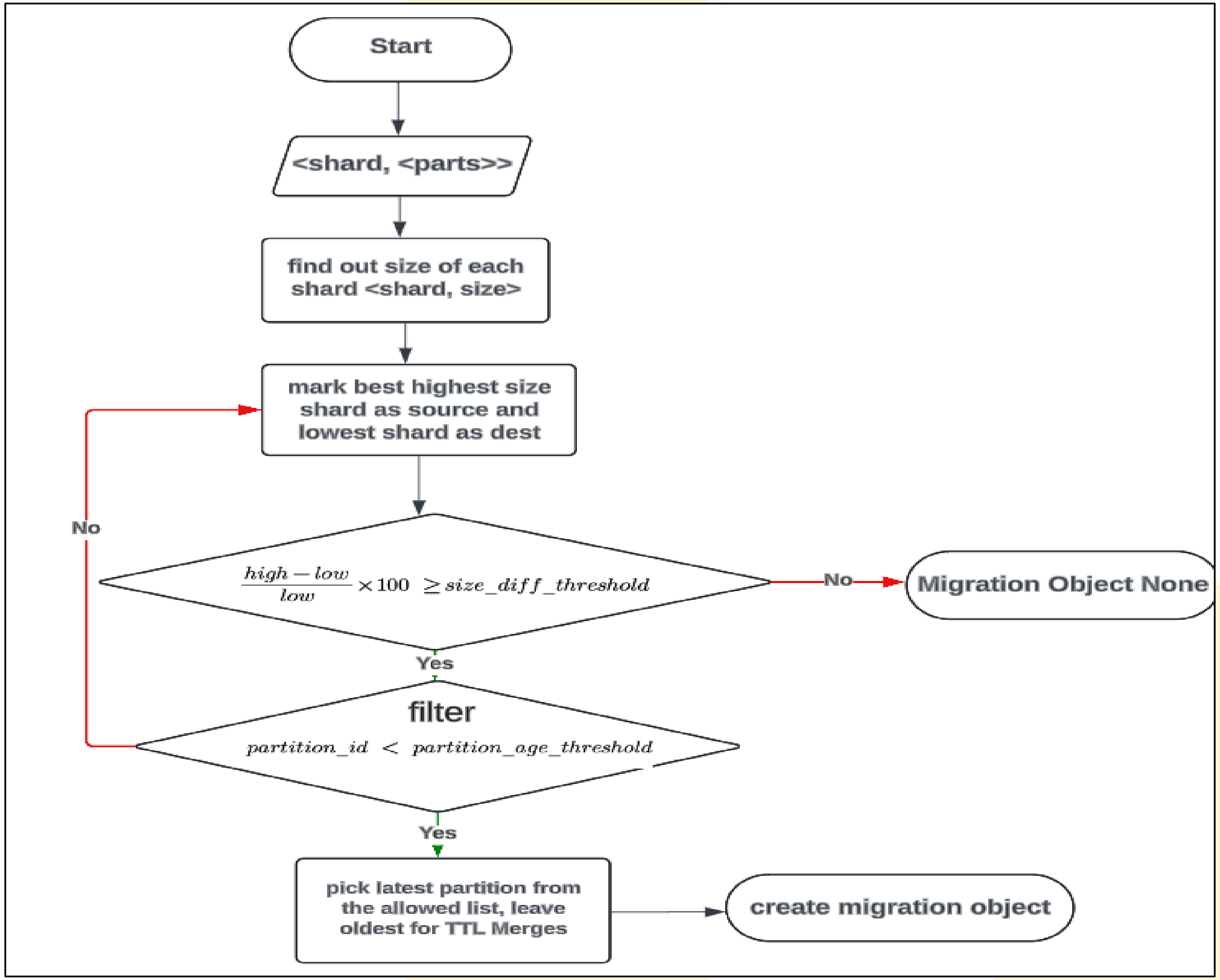
- Lots of on-prem deployments
- limited access
- Zero touch upgrades
- T-shirt size deployments
- Each size has fixed number of shards
- Bigger size means more clickhouse shards
- Re-balance during size upgrade



An external utility

1. Select source/destination shards for migration
2. Select partition least likely to be updated from source shard
3. Fetch parts from source shard
4. Attach partition to destination shard
5. Drop partition on source shard

Partition Selection



Start

<shard, <parts>>

find out size of each shard <shard, size>

mark best highest size shard as source and lowest shard as dest

$\frac{high - low}{low} \times 100 \geq size\_diff\_threshold$

No

Migration Object None

Yes

filter

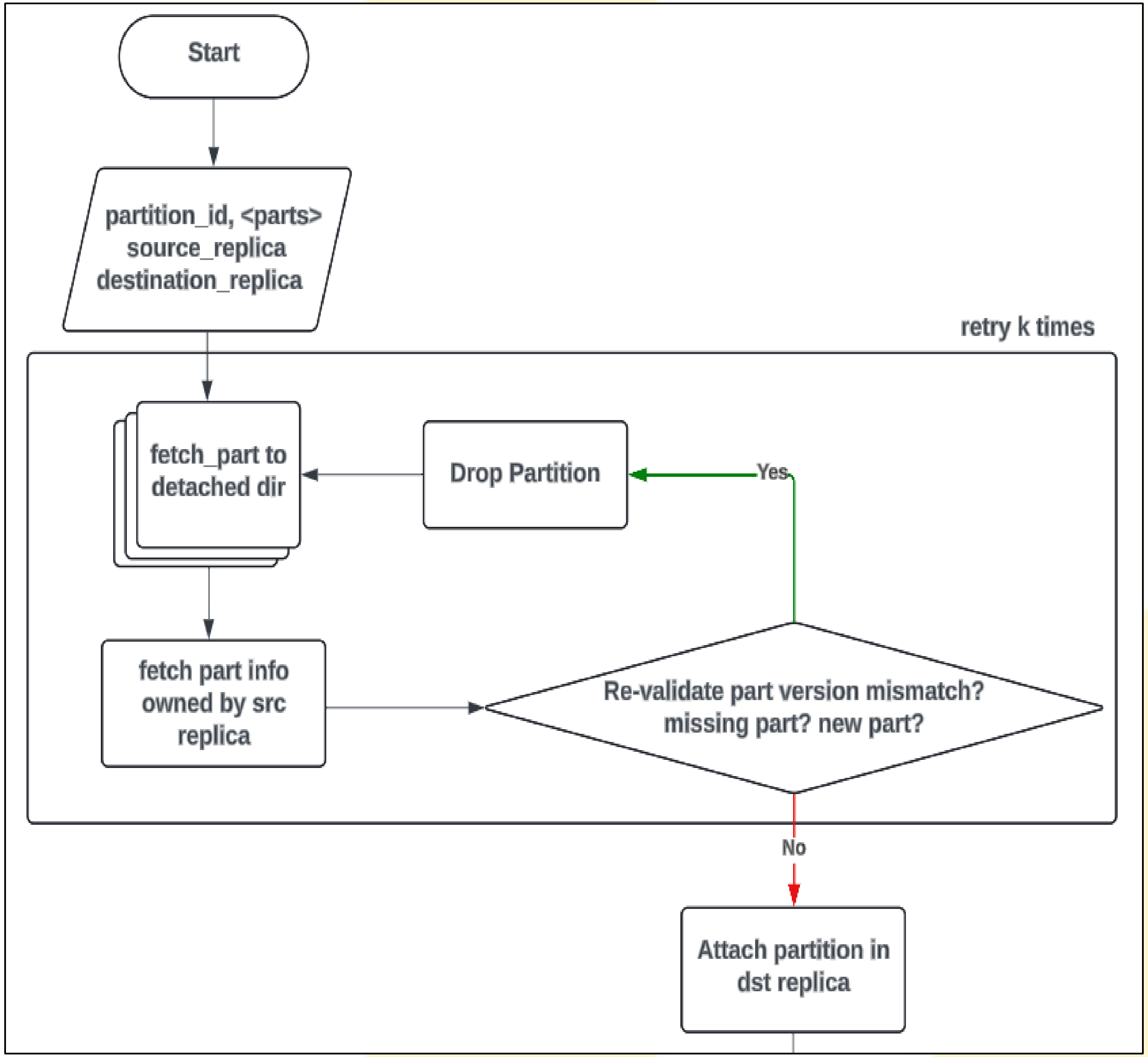
$partition\_id < partition\_age\_threshold$

No

Yes

pick latest partition from the allowed list, leave oldest for TTL Merges

create migration object


Start
partition_id, <parts>
source_replica
destination_replica
retry k times
fetch_part to detached dir
Drop Partition
Yes
fetch part info owned by src replica
Re-validate part version mismatch? missing part? new part?
No
Attach partition in dst replica

Optimistic Migration

Challenges

- Expensive Migration Retries
  - https://github.com/ClickHouse/ClickHouse/issues/66408
- Duplicates in SELECT
  - Can be solved with MOVE PARTS TO SHARD feature
- Works with limited partition key expression

Future Work

- For script-based rebalancing
  - Extend SYSTEM STOP MERGES to stop merges at PART|PARTITION level
  - Harden MOVE TO SHARD feature
- For fully automatic rebalancing
  - Implement Clickhouse rebalancing proposal
  - Extend K8s operator to rebalance data
  - Lets Collaborate!

QUESTIONS ?

Pranav Mehta

linkedin.com/in/pranavmehta94/

Shivji Kumar Jha

linkedin.com/in/shivjijha/

tinyurl.com/shiv-talks

tinyurl.com/shiv-slides

Thank You!