ClickHouse Practice and Contributions -- from academy to industry

郑天祺 博士, Amos Bird (Ph.D)，zhengtianqi@kuaishou.com

快手

About me

- Active ClickHouse Contributor
  - 200 + valid PRs
  - ~40 Stack Overflow Answers
  - Doing some code reviews occasionally

https://github.com/amosbird

- Graduated from ICT CAS with a Ph.D degree in database

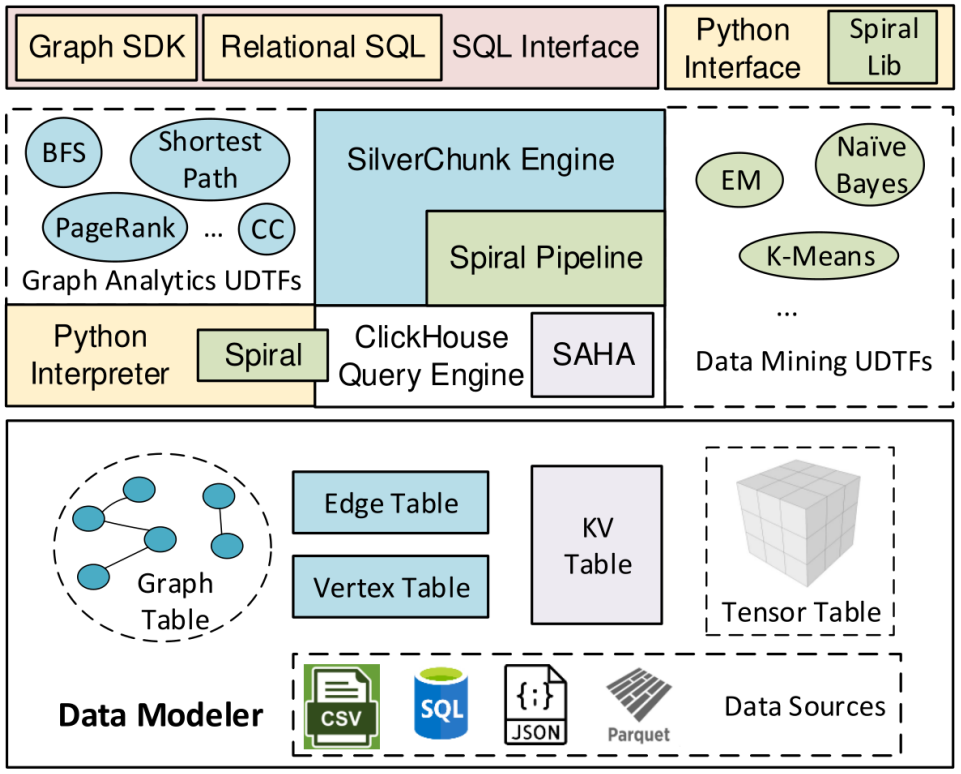
- Currently @kuaishou Data Platform Department

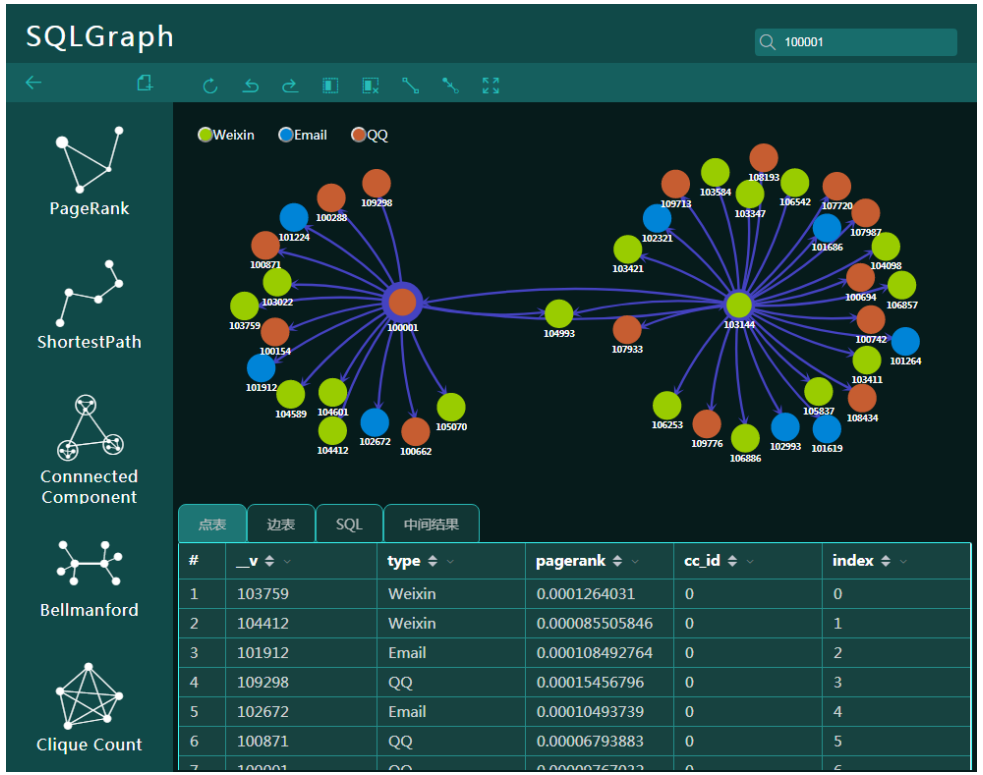
Outline

- Back in the school days – scenario exploration
- First entry into the industry – OLAP practice
- Consistent efforts in the open-source field – novel features
- Looking into the future – current working items

Back in the school days

- Try every possibilities out of ClickHouse





Back in the school days (Lessons learned)

- ClickHouse is very extensible and portable
  - A barebone computational framework (/programs)
  - A self-contained portable binary (even on Android)
  - A rich set of input/output interfaces and formats (omnipotent)

- ClickHouse is friendly to use, and easy to hack
  - There are 4 contributors in my lab, some are green hands
  - The system is understandable from top to bottom

- ClickHouse can solve real problems

Back in the school days (Lessons learned)

- ClickHouse is very extensible and portable
  - A barebone computational framework (/programs)
  - A self-contained portable binary (even on Android)
  - A rich set of [illegible]otent)
- ClickHouse is [illegible]
  - There are 4 [illegible]s
  - The system is understandable from top to bottom
- ClickHouse can solve real problems

"If some system works a little faster than ClickHouse on some degenerate query, this means that I have not yet optimized the code, and I will do it tomorrow." – Alexey Milovidov

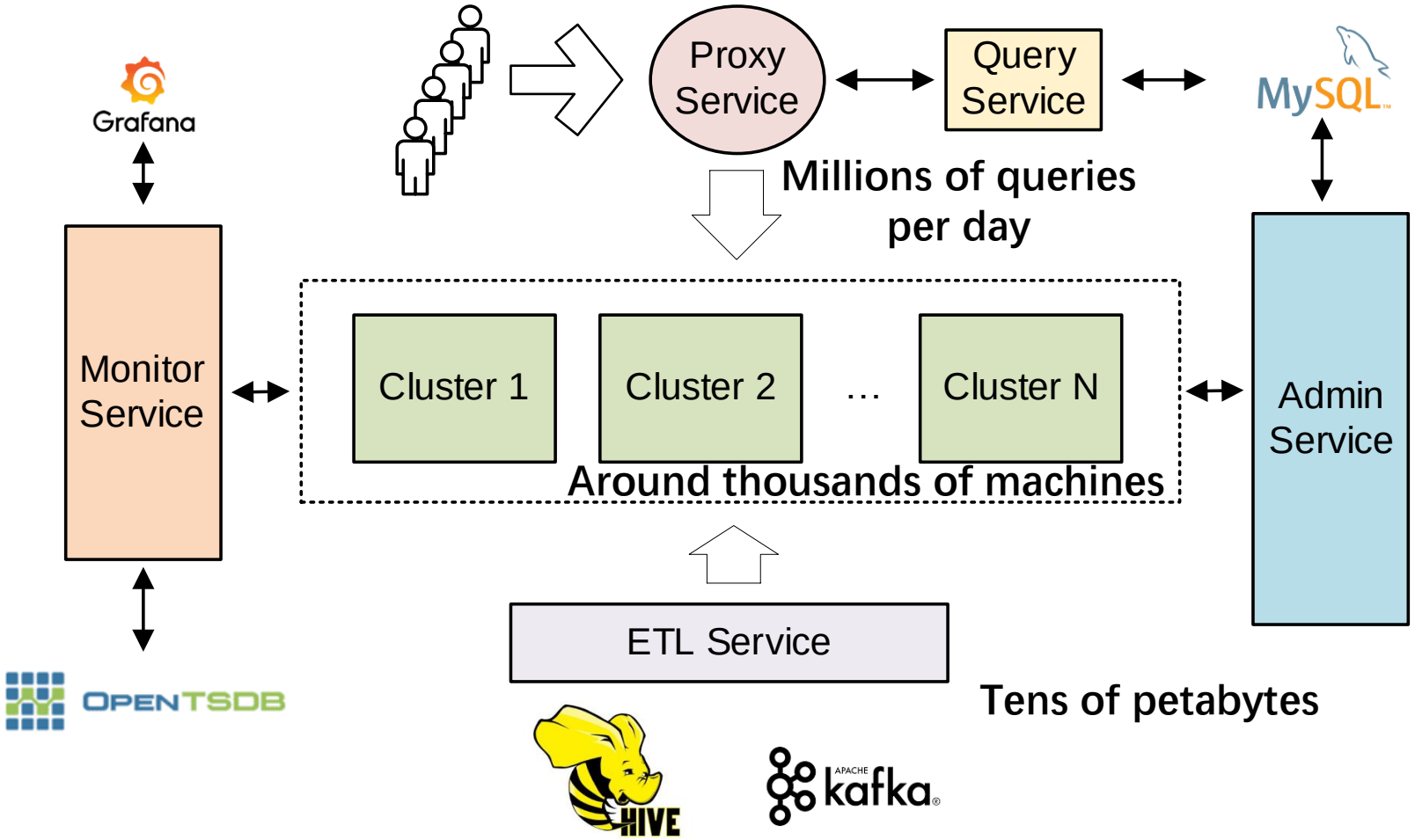
First entry into the industry

- @kuaishou for ~100 days now

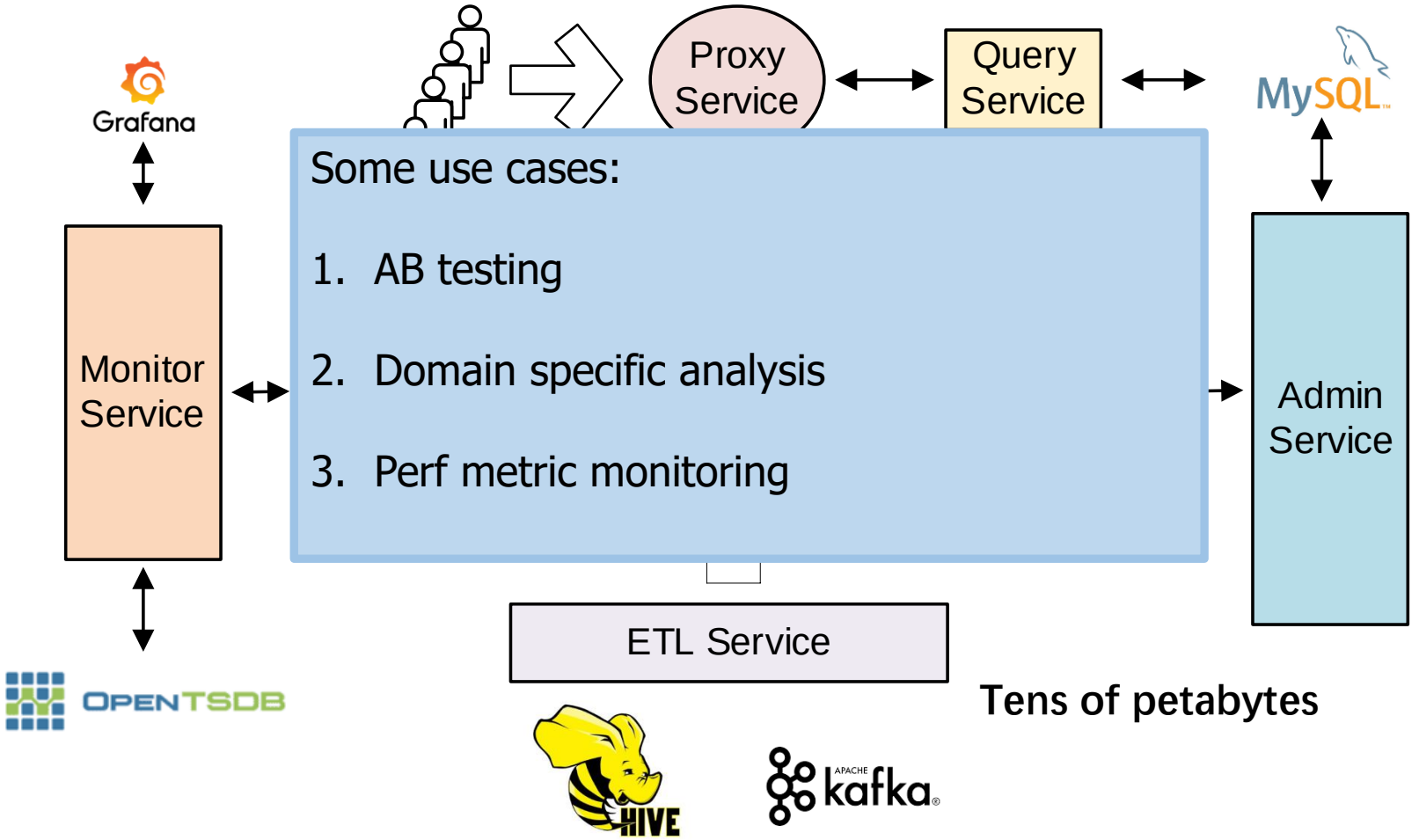
- Interesting setups
  - Multiple clusters with a proxy service portal
  - Frontend of ClickHouse management
  - ETL pipelines from Hive and Kafka
  - Two kinds of ClickHouse clusters

- Demanding issues
  - HW RAID crash and IO inefficiency
  - Query optimization (mostly indices)
  - Resource management

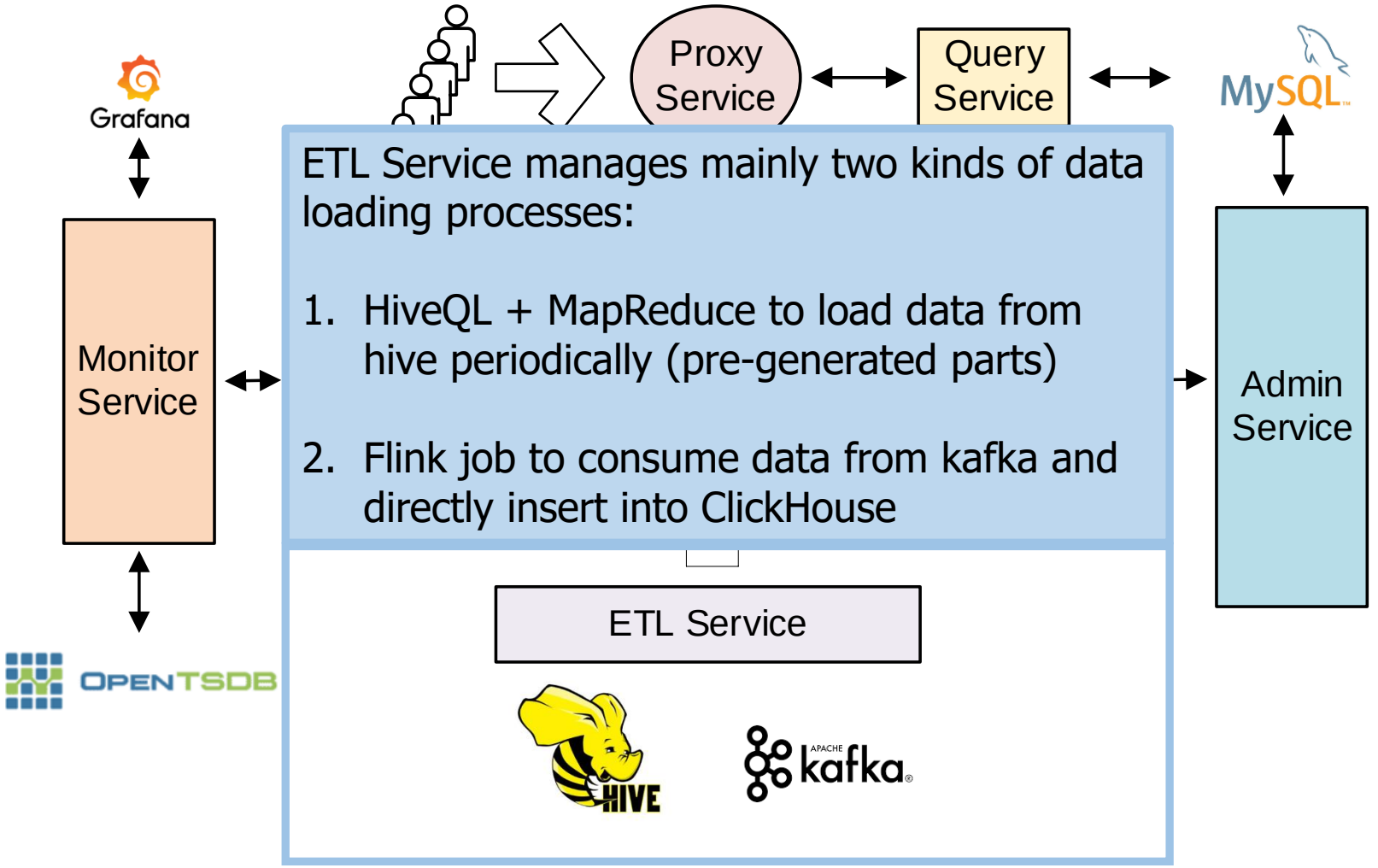
Full picture of our ClickHouse service



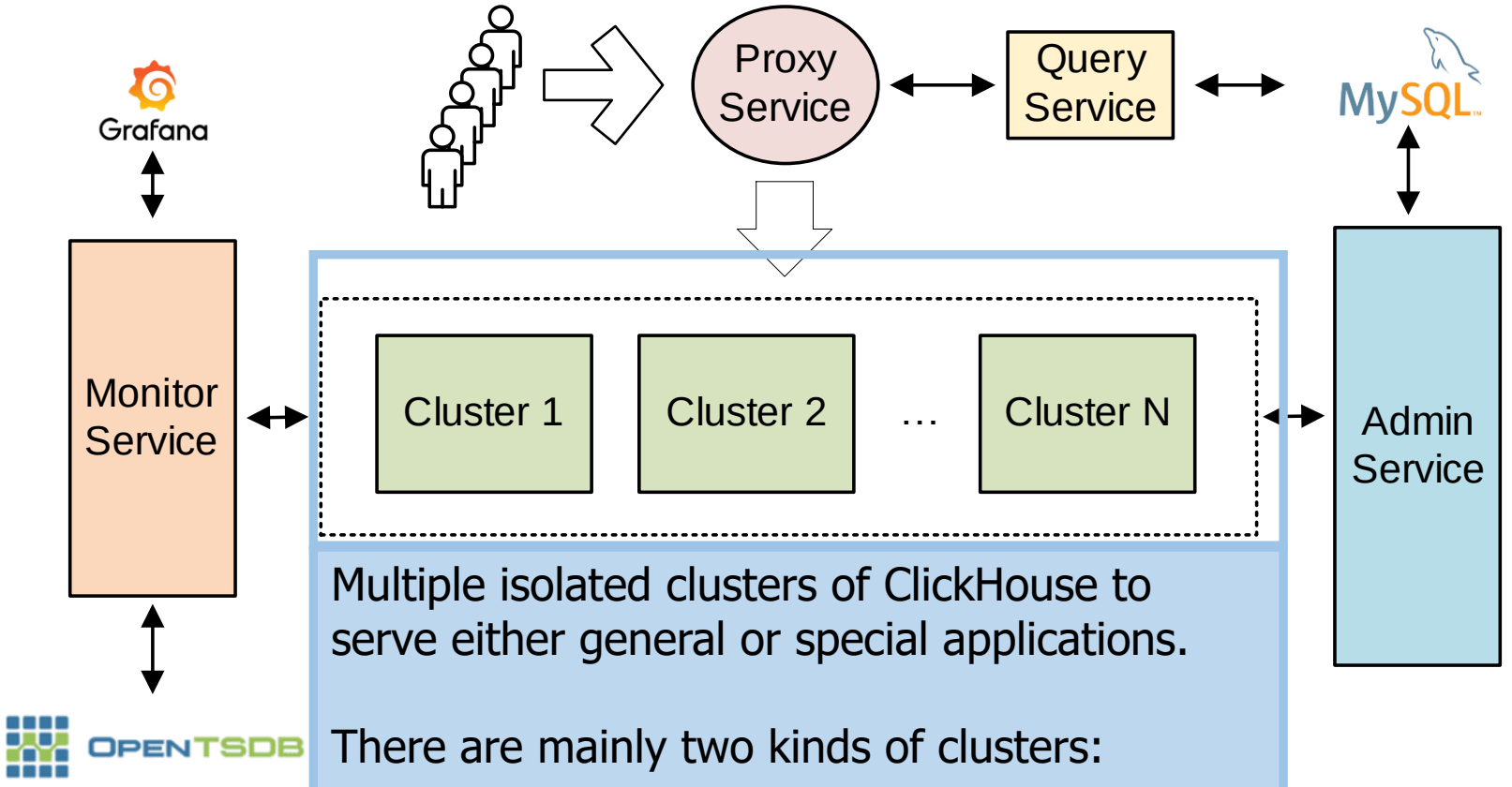
Full picture of our ClickHouse service



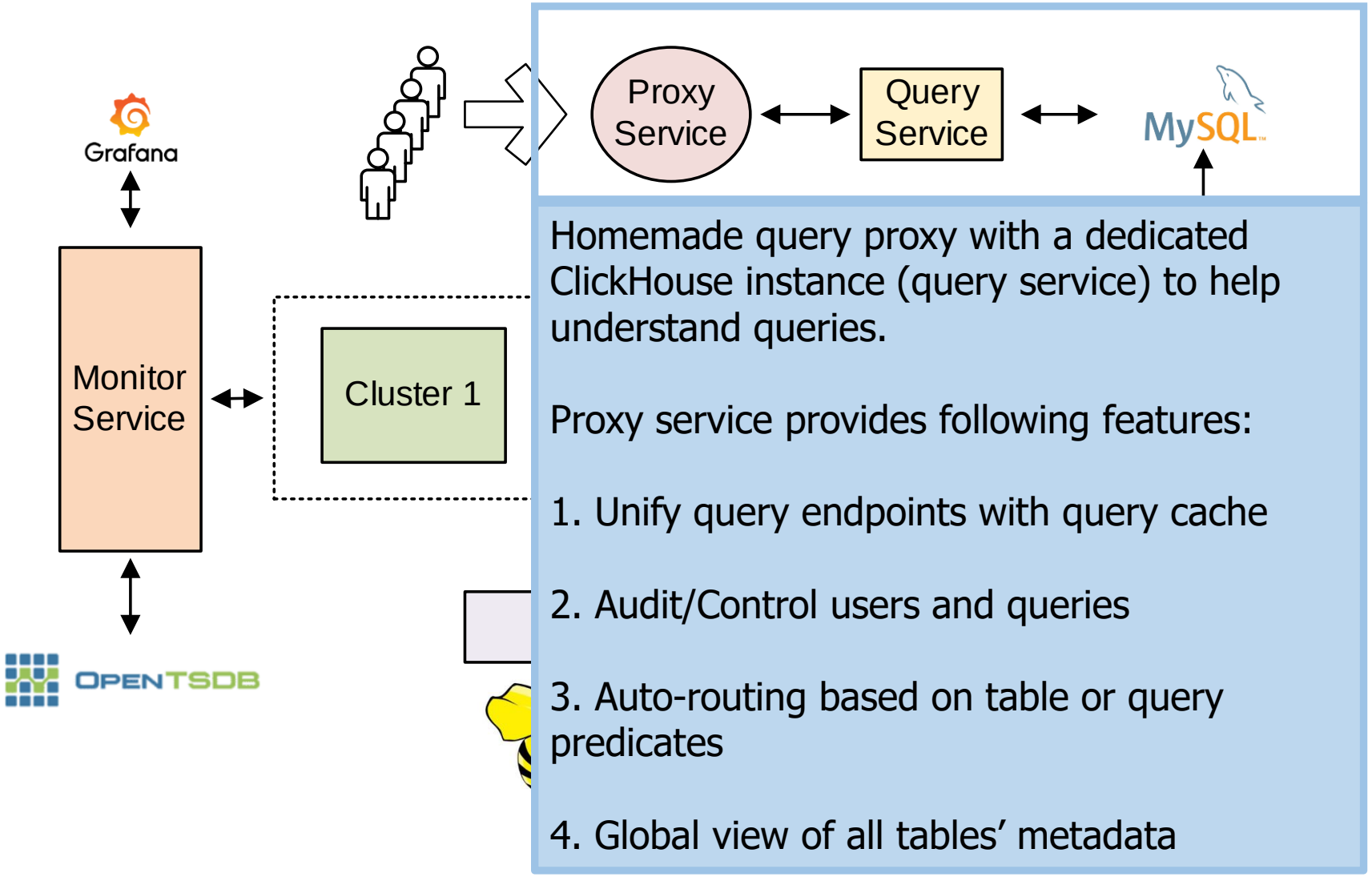
Full picture of our ClickHouse service



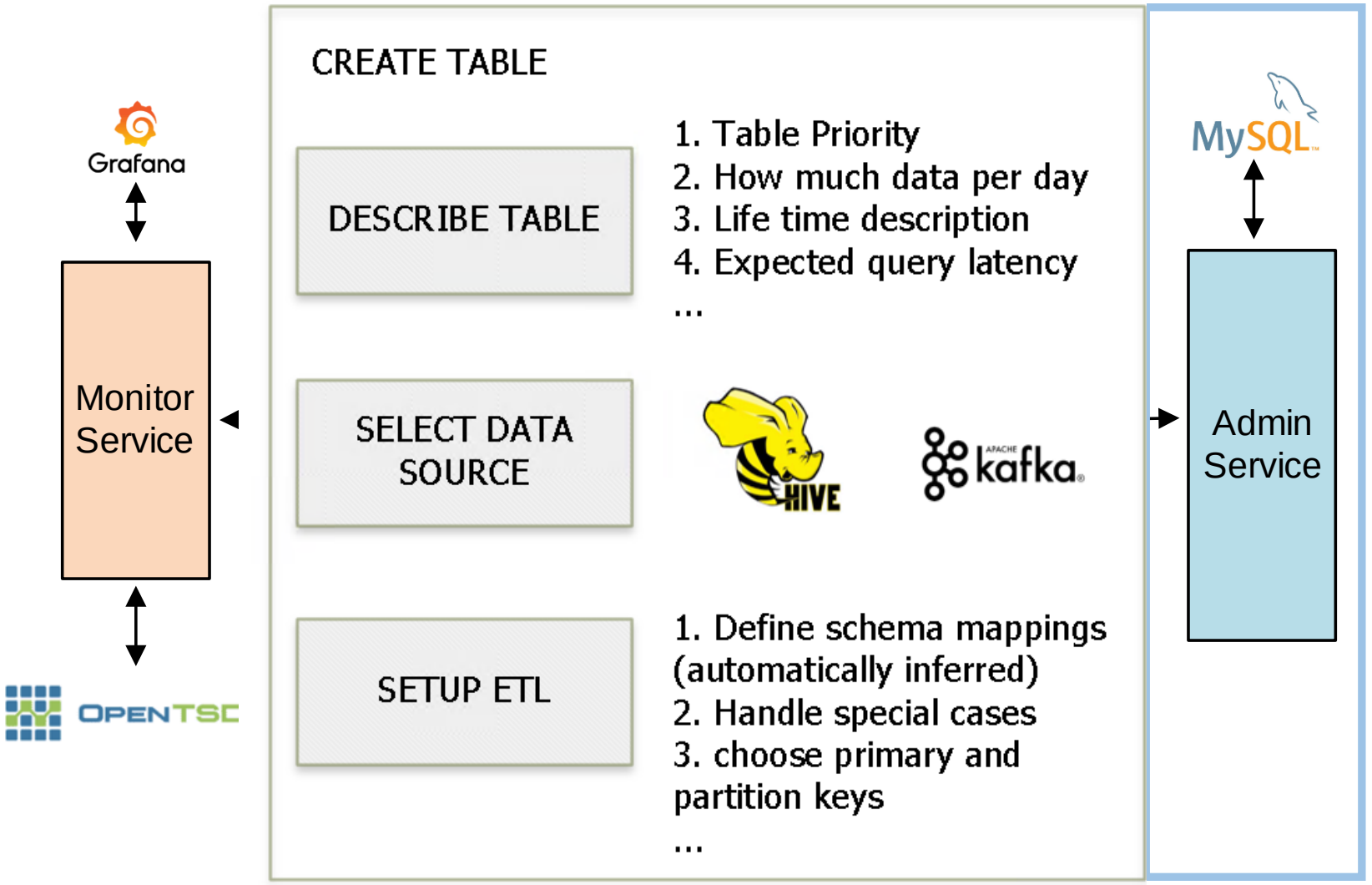
Full picture of our ClickHouse service



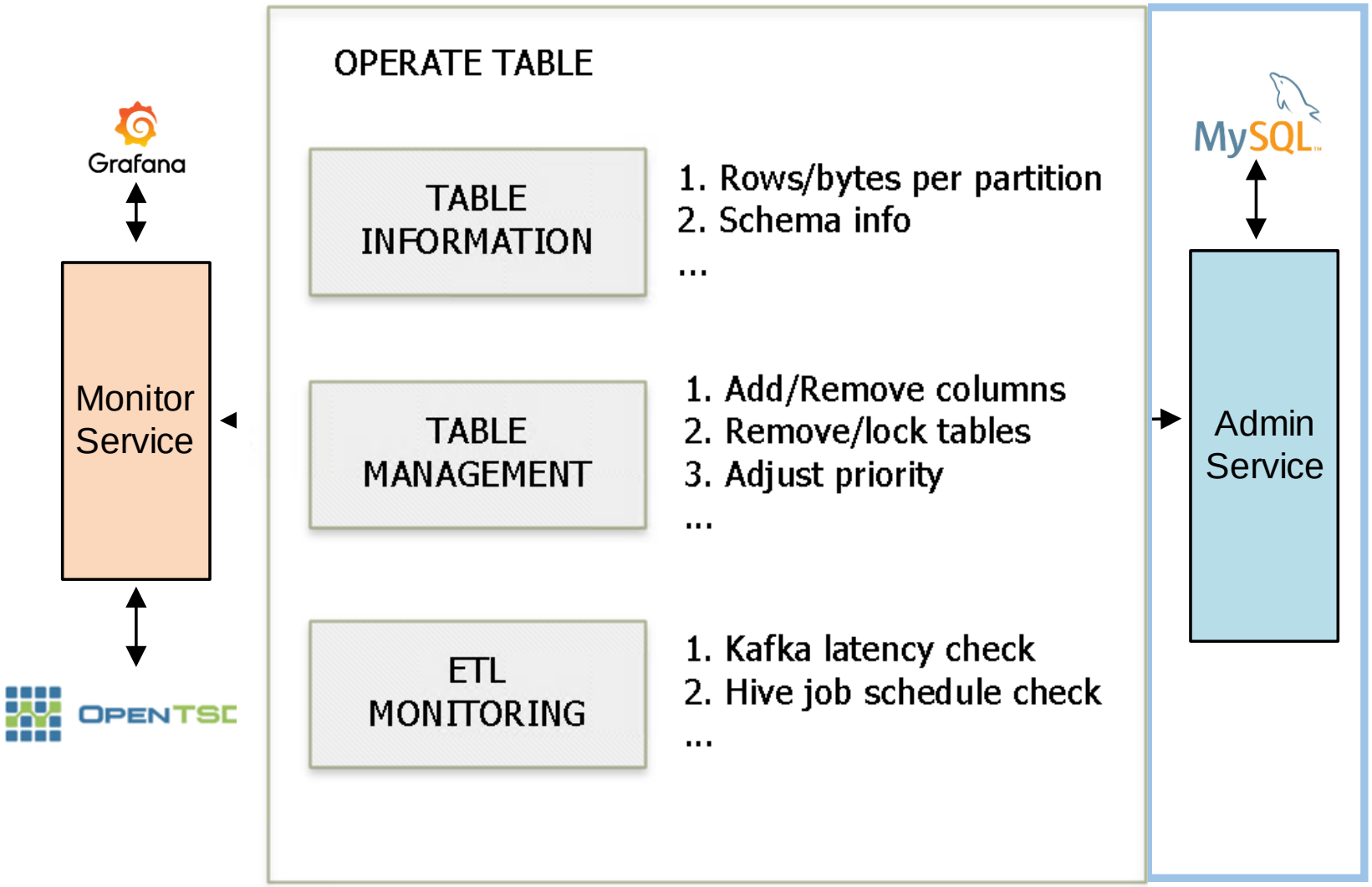
Full picture of our ClickHouse service



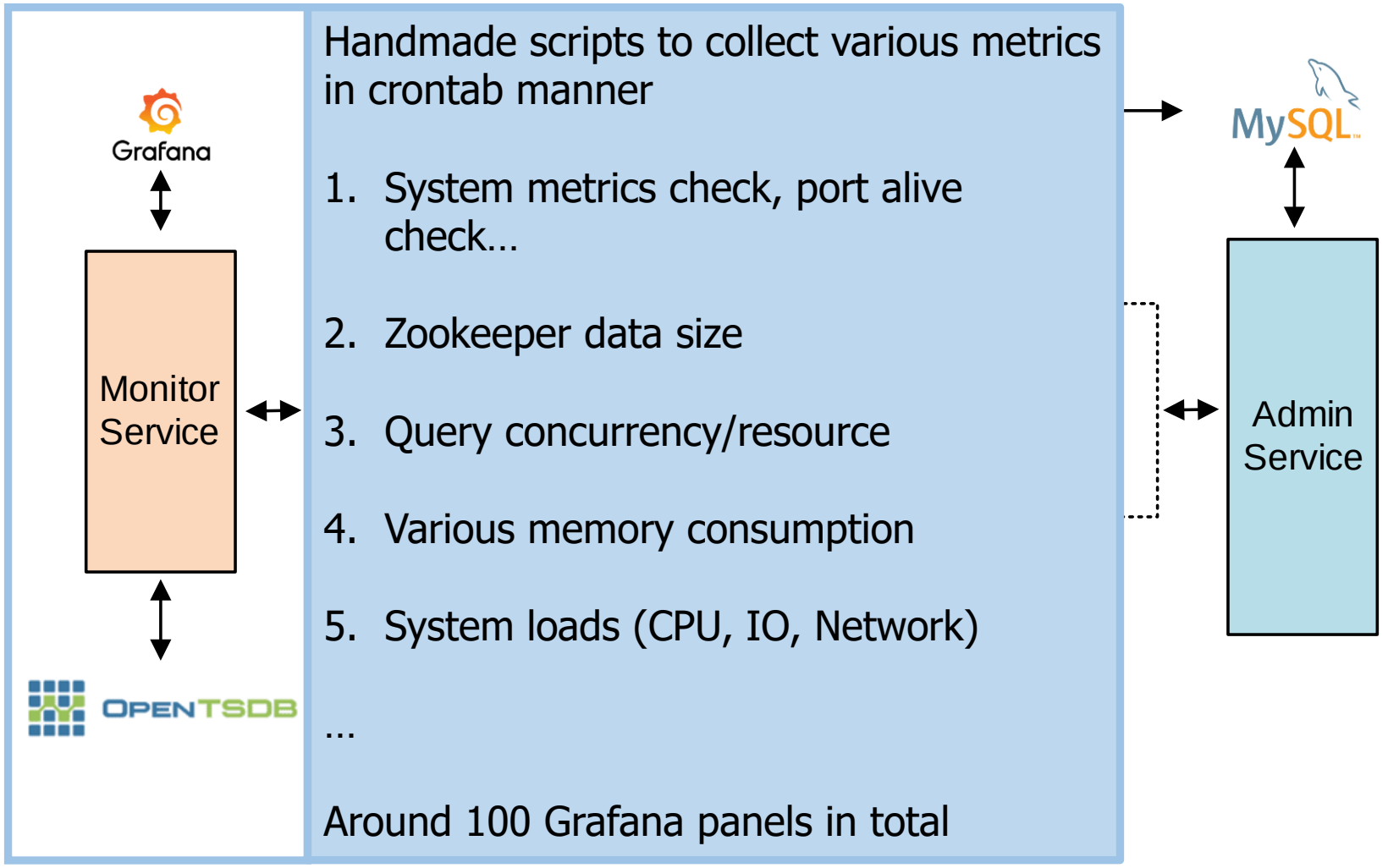
Full picture of our ClickHouse service



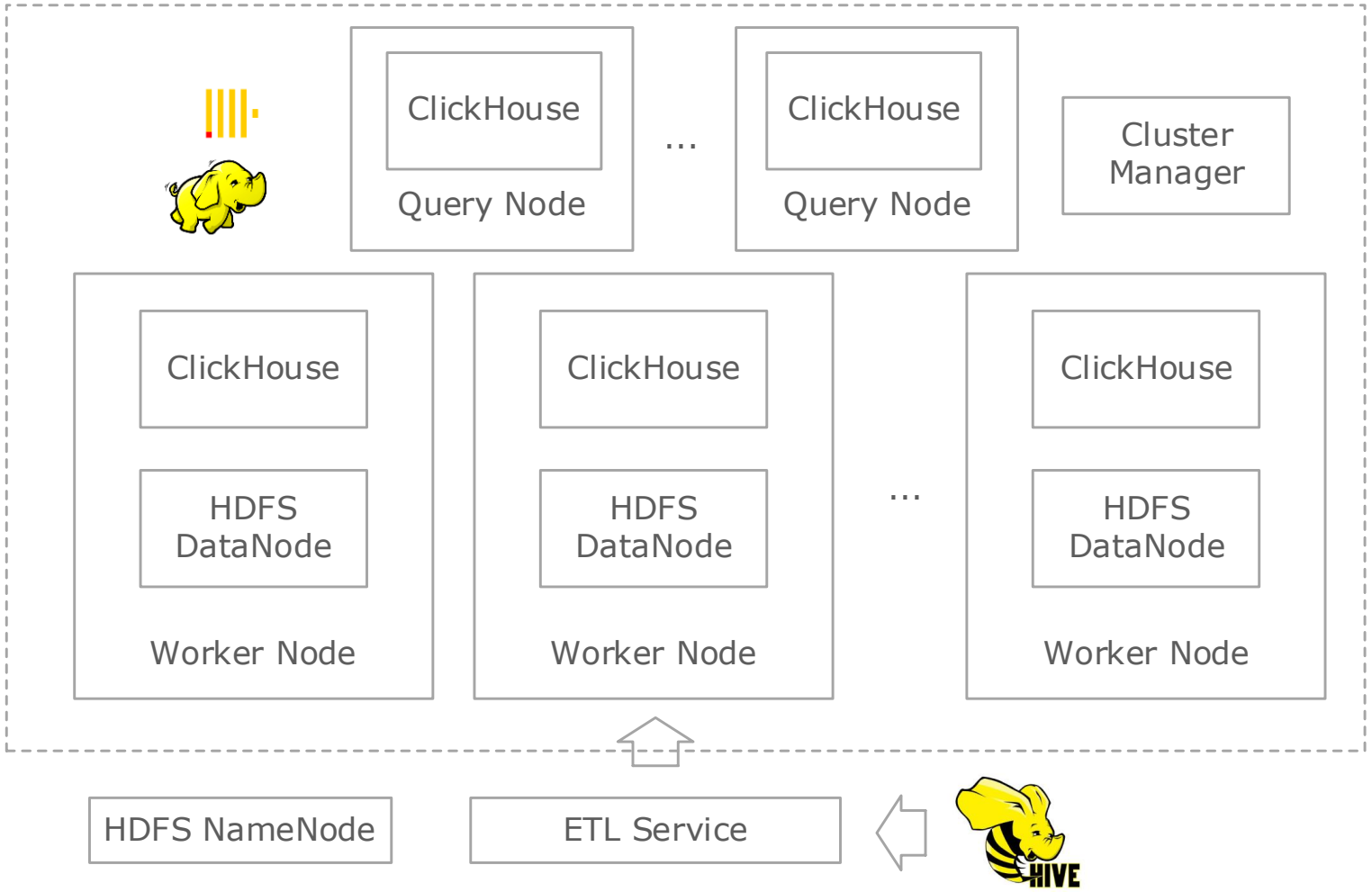
Full picture of our ClickHouse service



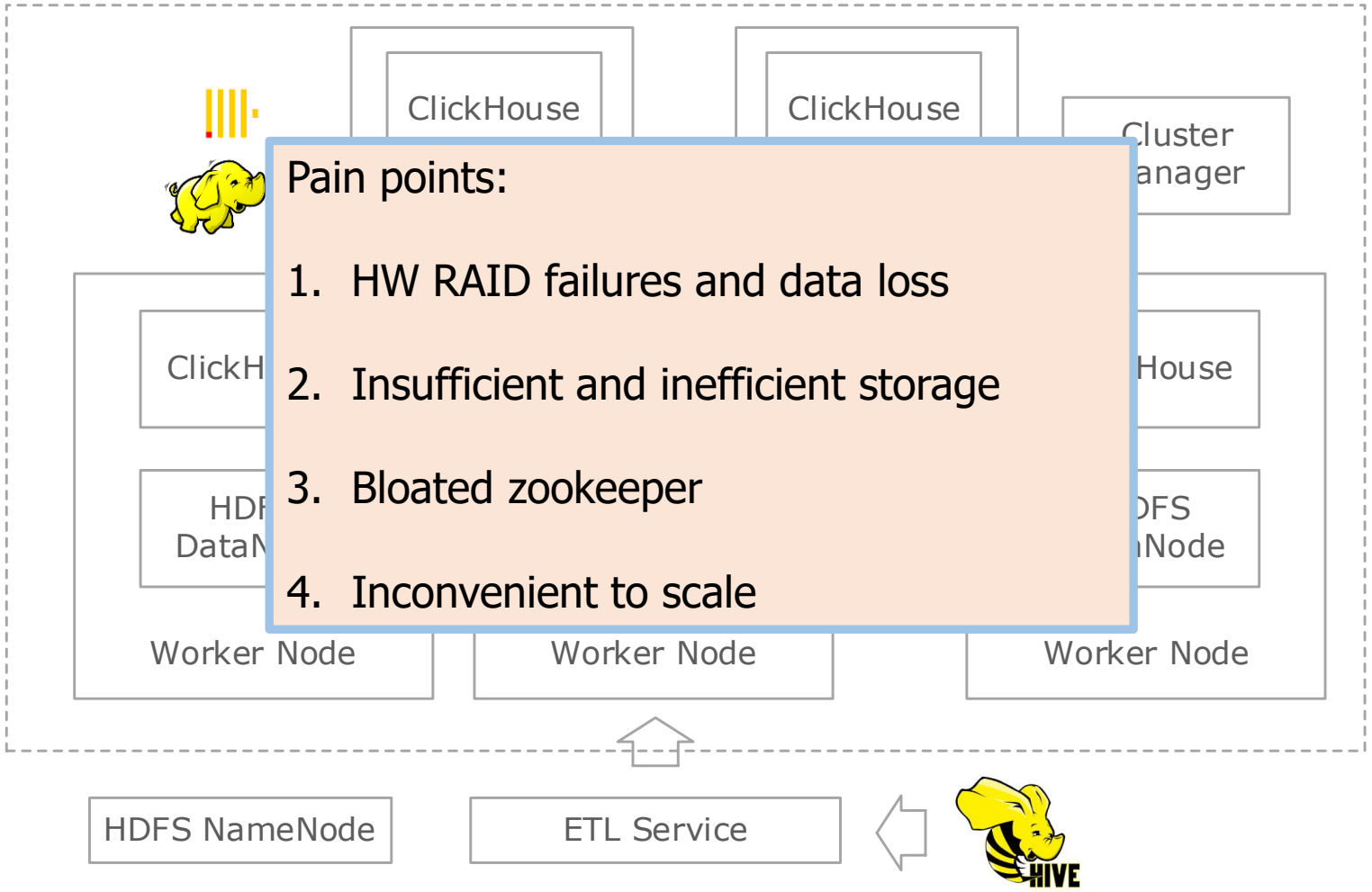
Full picture of our ClickHouse service



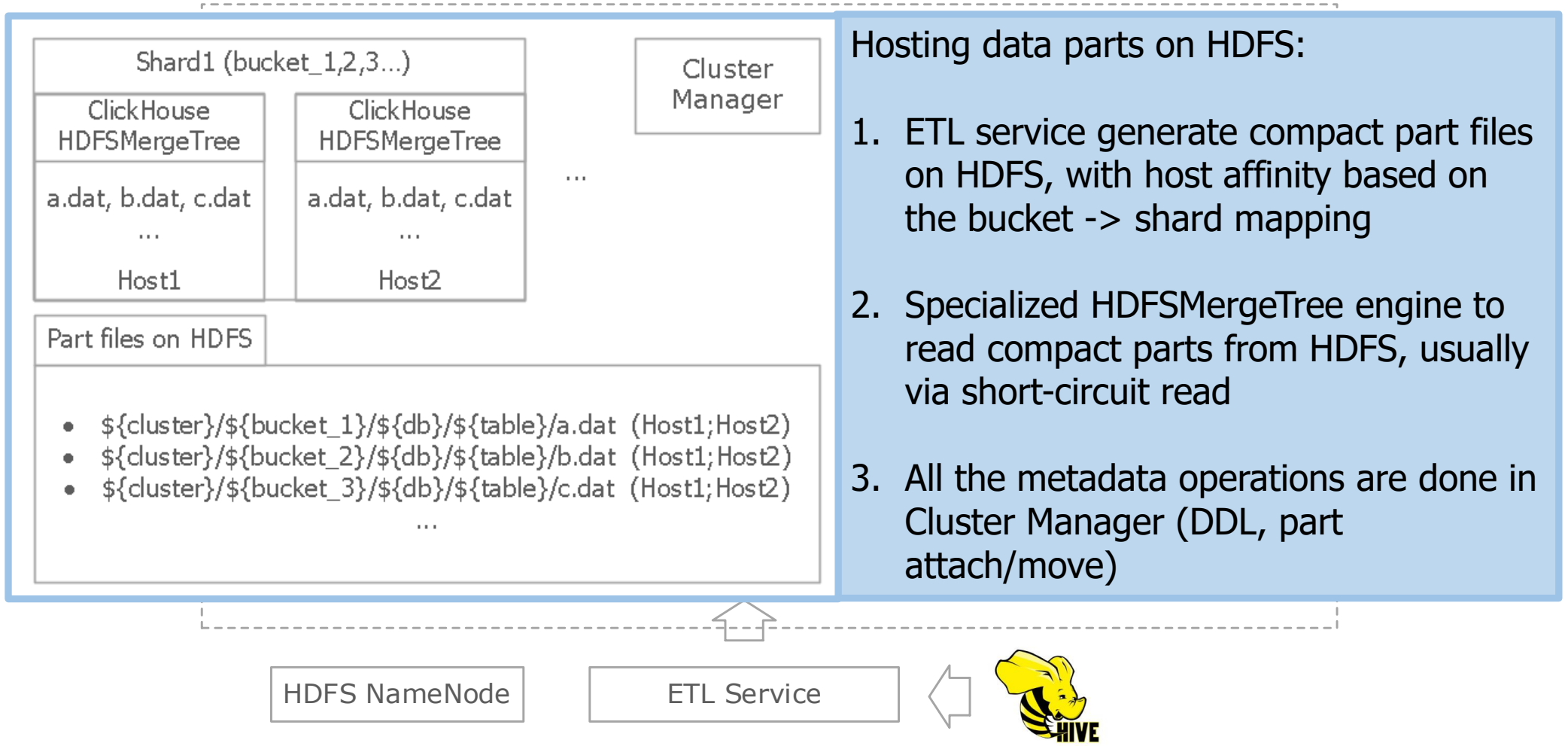
ClickHouse on HDFS



ClickHouse on HDFS



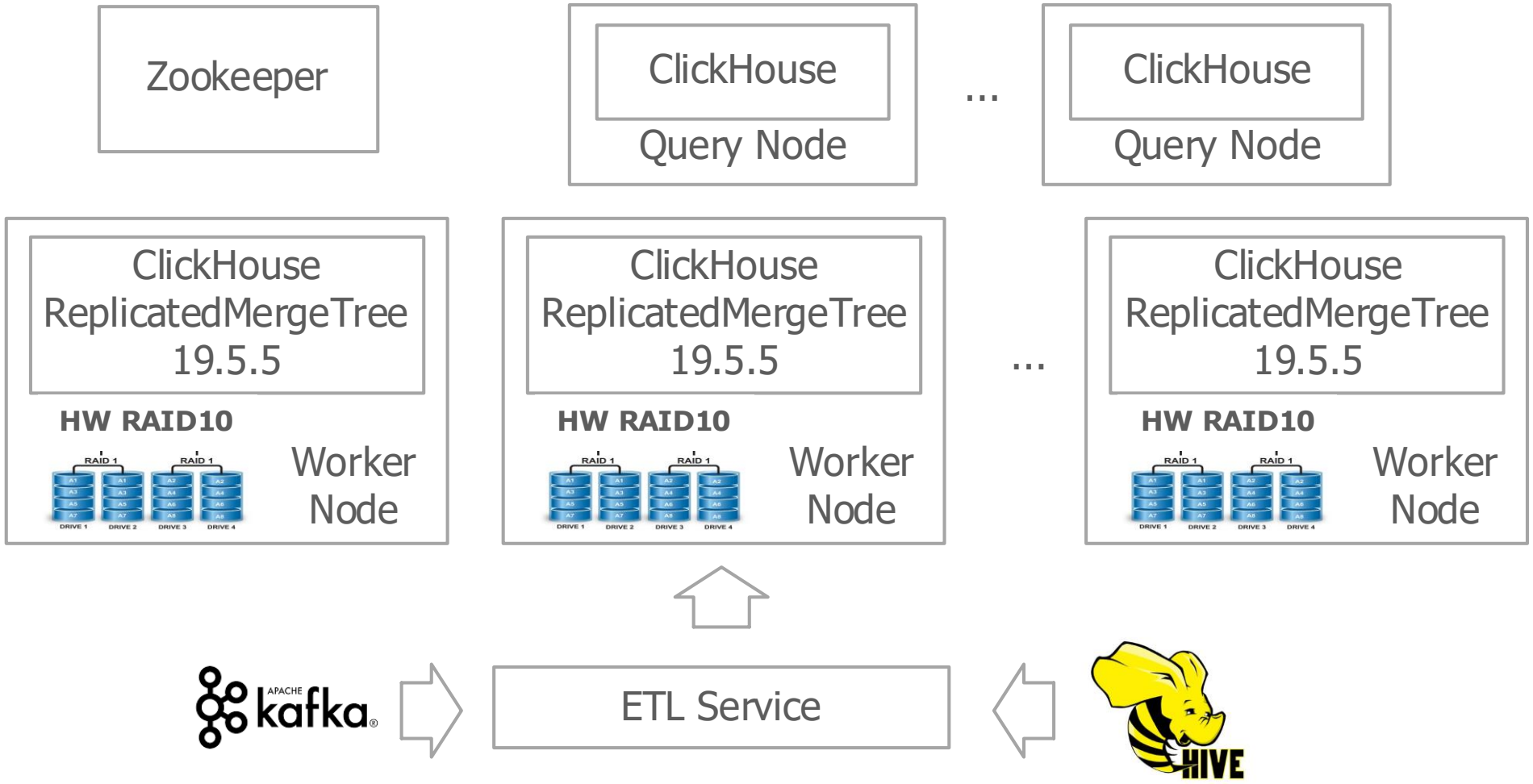
ClickHouse on HDFS



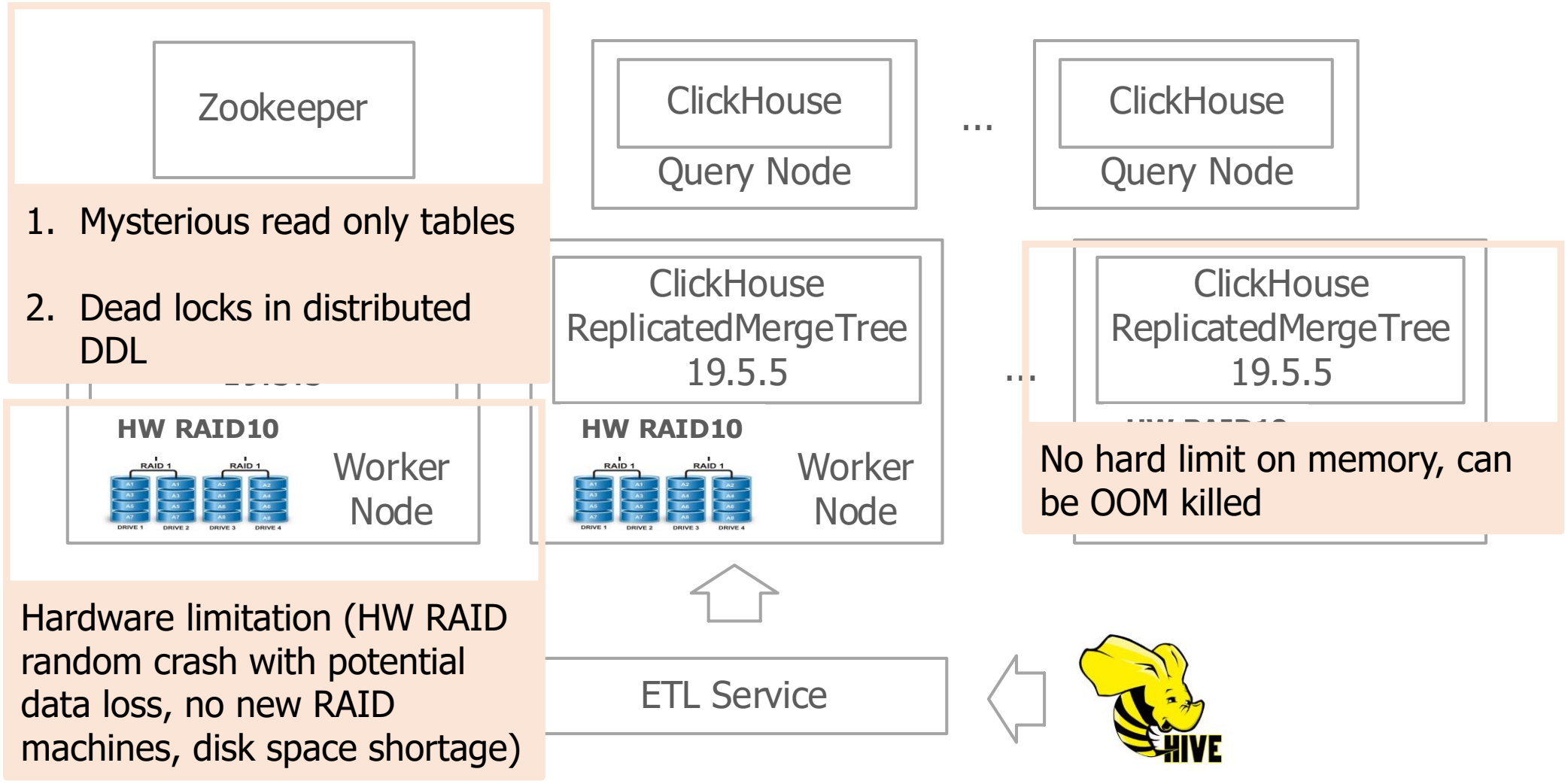
Hosting data parts on HDFS:

1. ETL service generate compact part files on HDFS, with host affinity based on the bucket -> shard mapping
2. Specialized HDFSMergeTree engine to read compact parts from HDFS, usually via short-circuit read
3. All the metadata operations are done in Cluster Manager (DDL, part attach/move)

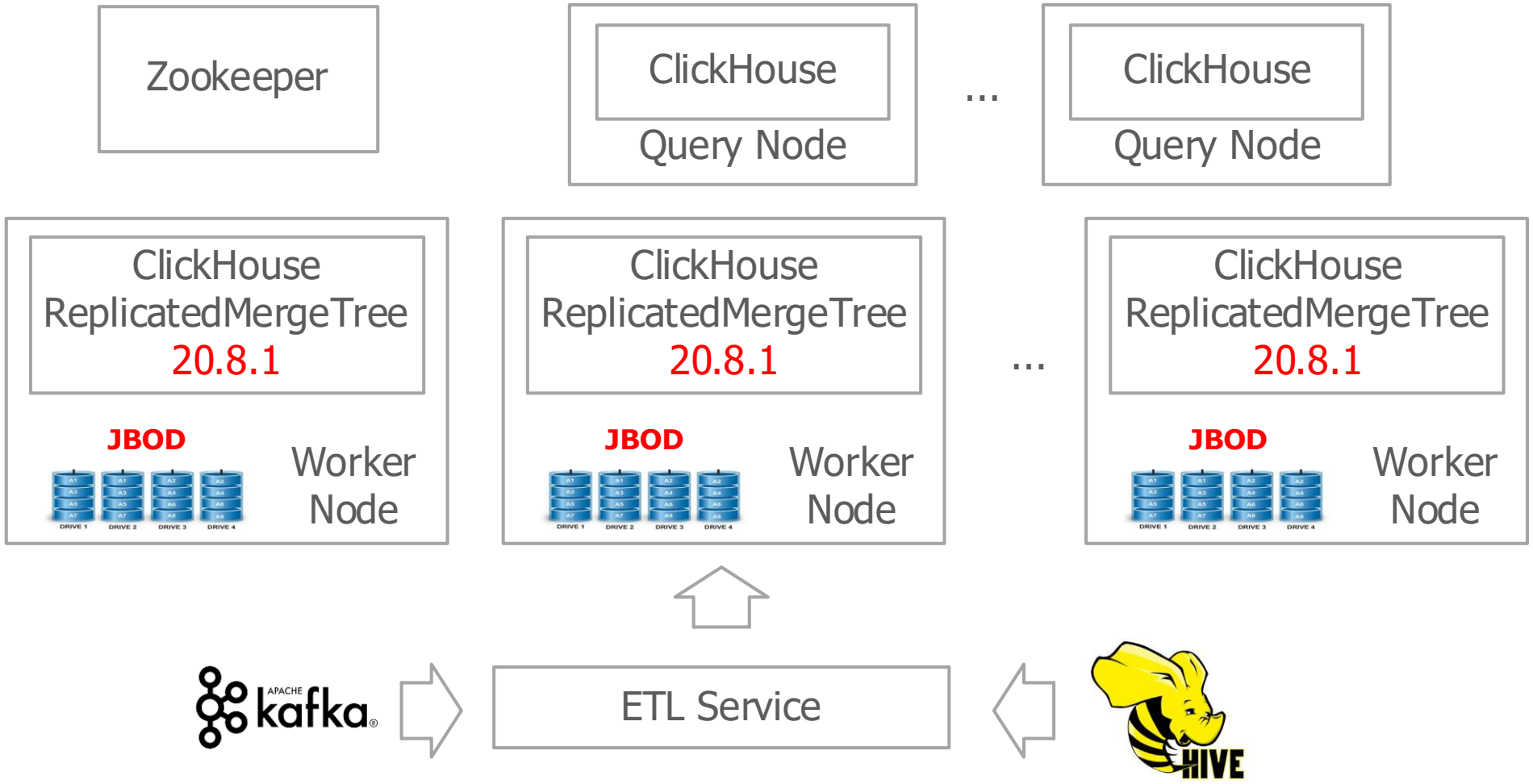
Vanilla ClickHouse (Before)



Vanilla ClickHouse (Pain points recap)



Vanilla ClickHouse (After)



Things we learned in rolling upgrade

- Disable DDL operations or else ReplicatedMergeTree would be stuck at some unrecognizable entry and data insertion will eventually stop

- Do not allow new instances to be leader, or else the new instances assign merges aggressively and the old instances might OOM. Also need to disable *index_granularity_bytes* and *write_final_mark*
  - Bonus point: Prepare to recovery data files from broken marks (kudos to Alexey)

- Distributed queries should be sent to new instances to get correct results

Things we did after using JBOD

- Automatically recover parts when some disk is broken
- Diagnose bad IO utilization and tune disk settings
- Observe read clustering and tail latency issue, solved by balanced JBOD read
- Balance query related data files over JBOD array (WIP)

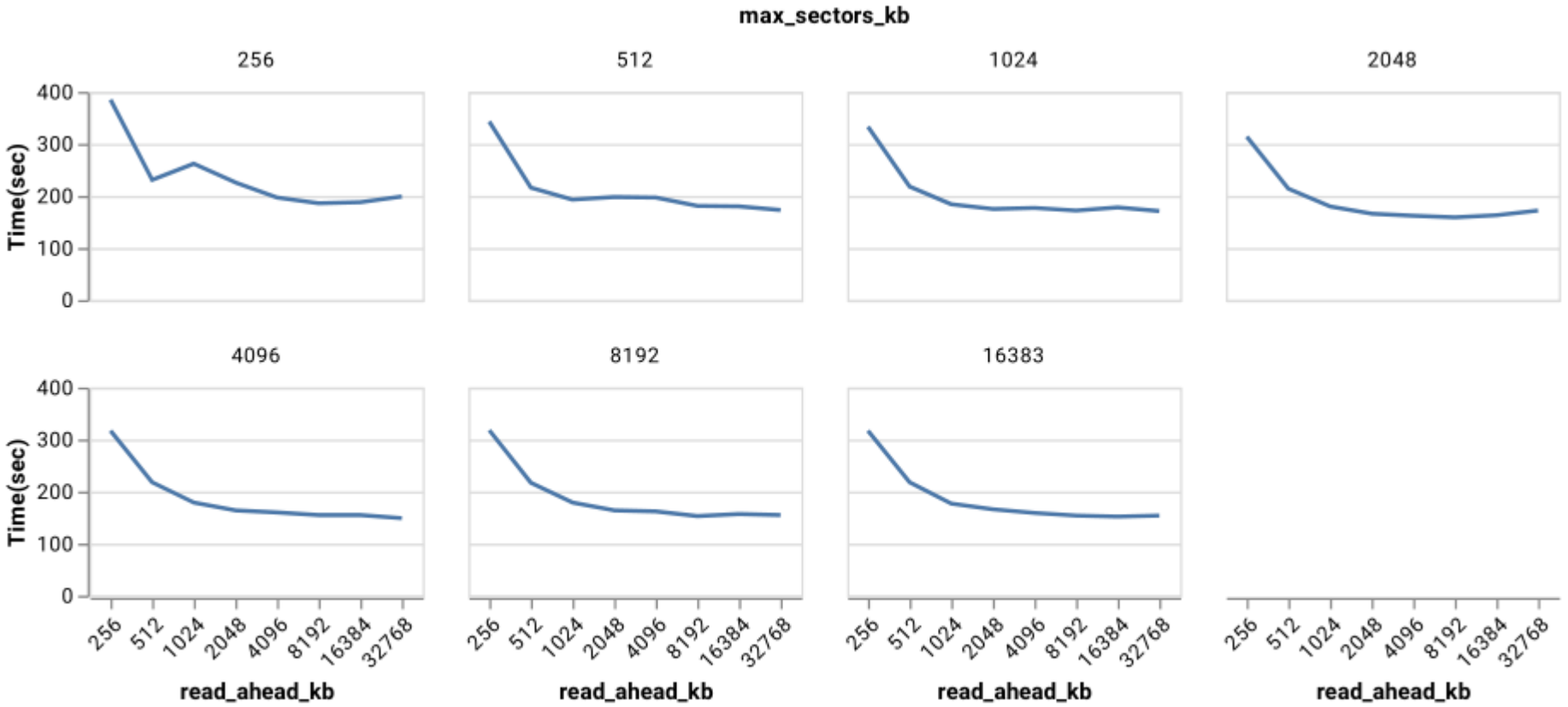
JBOD disk auto-recovery

- Each local disk spawns a disk checker thread, which gives three states: broken, healthy, recovered.
- Disk checker will automatically transit disks from healthy to broken or broken to recovered. One need to create a .recover file in the disk path to bring recovered into healthy (avoid flaky disks)
- Each ReplicatedMergeTree table constantly checks if a part is on an unhealthy disk and starts to recover all related parts.
- Whenever a query scans a broken part, it starts to recover instantly.

https://github.com/ClickHouse/ClickHouse/pull/13544

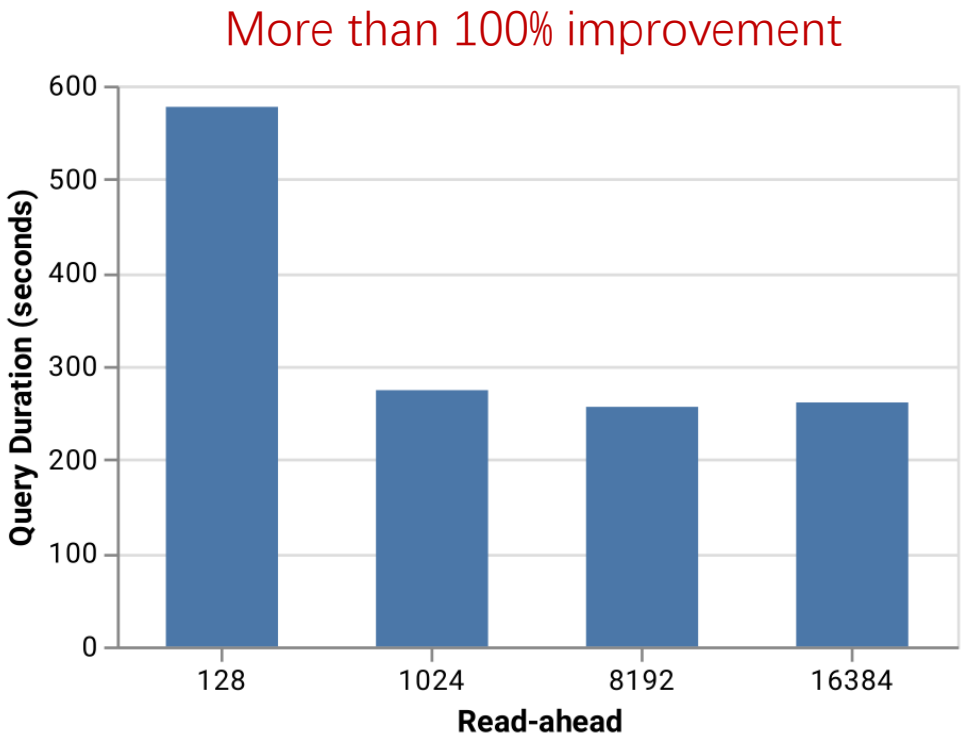
Some test results of JBOD

- Disk read-ahead tuning benchmark using long running queries



Some test results of JBOD

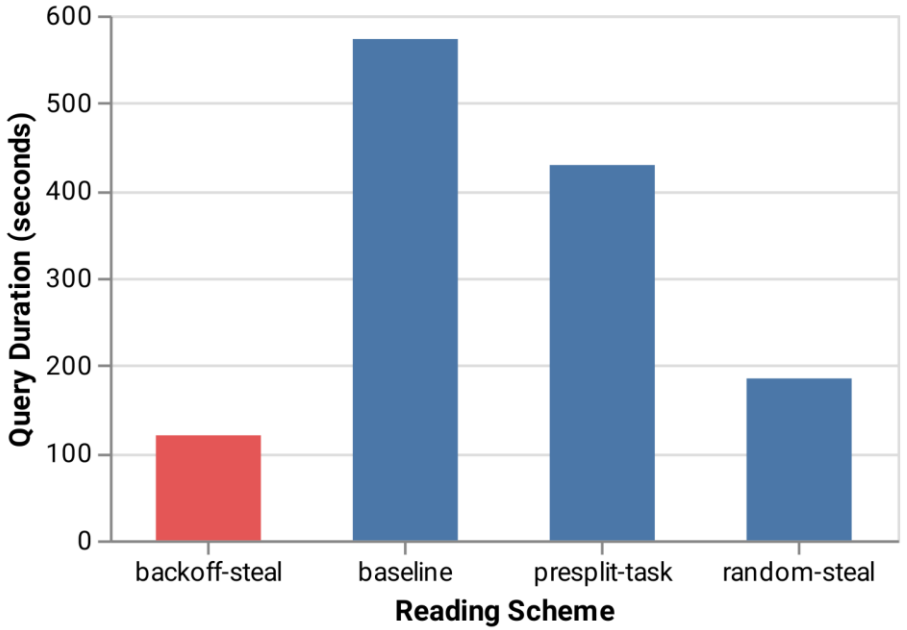
- Disk read-ahead tuning benchmark using long running queries



Balanced read among JBOD disks

- Avoid read clustering and tail latency

Significant improvements, more than 400%



https://github.com/ClickHouse/ClickHouse/pull/16423

Balanced data among JBOD disks (WIP)

- Focused on long running queries targeting for given partitions (usually a day)
- Lower *max_bytes_to_merge_at_max_space_in_pool* to have smaller but more parts (hopefully gets better balanced)
- Currently balance via ALTER PART MOVE in crontab manner
- Eventually will have automatic part balancer over JBOD array

https://github.com/ClickHouse/ClickHouse/pull/16481

Nullable primary key

- Almost 99% of columns are Nullable in production
- Nulls are default to be greater than any other values. It's trivial to allow Nullable types appearing in the primary key expression, but a bit tricky to have index condition work correctly.
- We introduce '-inf' and '+inf' as two value sentinels and let Null = +inf. Index processing is extended to handle these sentinels.

https://github.com/ClickHouse/ClickHouse/pull/12455

Index usage optimization (Monotonicity)

- toDate is not satuerated
  - toDate will not work when a part doesn't have a final mark (+Inf)
  - toDate might not work when a part's primary key range outside one monotonic interval

- Binary operator with a constant argument (notably division)

- Now we can have `toDate(timestamp_ms / 1000)` work as index column

https://github.com/ClickHouse/ClickHouse/pull/13497

https://github.com/ClickHouse/ClickHouse/pull/14513

Partition predicates optimization

- Prune partition in verbatim way
  - Partition expressions can be wrapped in non-monotonic functions, such as hashing and modulo.
  - We use the invariant: Partition value is fixed in each part, and filter parts with that value and given predicates.
- Instant count() with partition predicates
  - We use similar technology to select parts that are always true and filter parts that are always false. When there is nothing left, we return the count

https://github.com/ClickHouse/ClickHouse/pull/16253

https://github.com/ClickHouse/ClickHouse/pull/15074

Consistent efforts in the open-source field

- Column transformers
- untuple (feat. Nikolai)
- CTE and global with
- protobuf format schema via HDFS
- View function
- Fetch partitions from another cluster
- clickhouse-client Introspection usability

Column transformers

- Idea is originated from Big Query and suggested by Alexey
- Three kinds of column matchers: “ `*, table.* and COLUMNS(<regexp>)`” are extended to do column transformations

https://github.com/ClickHouse/ClickHouse/pull/14233

Column transformers

- Some examples

```
SELECT * APPLY(sum)
FROM columns_transformers

┌─sum(i)─┬─sum(j)─┬─sum(k)─┐
│    220 │     18 │    347 │
└────────┴────────┴────────┘
```

```
SELECT columns_transformers.* EXCEPT(j) APPLY(avg)
FROM columns_transformers

┌─avg(i)─┬─avg(k)─┐
│    110 │  173.5 │
└────────┴────────┘
```

```
SELECT columns_transformers.* REPLACE(j + 2 AS j, i + 1 AS i) APPLY(avg)
FROM columns_transformers

┌─avg(plus(i, 1))─┬─avg(plus(j, 2))─┬─avg(k)─┐
│             111 │              11 │  173.5 │
└─────────────────┴─────────────────┴────────┘
```

Column transformers

- Useful for INSERT SELECT

```
INSERT INTO insert_select_dst(* EXCEPT (middle_a, middle_b)) SELECT * FROM insert_select_src
```

- Schema transformers?

```
INSERT INTO table SELECT * FROM file('./table/*','CSV', schema(table EXCEPT …))
```

  - Some discussion https://github.com/ClickHouse/ClickHouse/issues/16295

untuple (feat. Nikolai)

- The missing feature of named tuples.

```
SELECT
    key,
    untuple(argMax(tuple(* EXCEPT(key)), v1))
FROM kv
GROUP BY key
```

key	v1	v2	v3	v4	v5
4	10	20	10	20	30
3	70	20	10	20	30
2	11	20	10	20	30
5	10	20	10	20	30
1	20	20	10	20	30
6	10	20	10	20	30
7	18	20	10	20	30
8	30	20	10	20	30

https://github.com/ClickHouse/ClickHouse/pull/16242

CTE (common table expression)

- ClickHouse has exotic support of the WITH statement. It's used to introduce scalar aliases into the query context.
  - WITH 1 AS a, (select * from table) AS b SELECT ⋯
- Sometimes we need to have named subqueries introduced as table objects (not scalars). The syntax is consistent with standard.
  - WITH s AS (select * from table) SELECT * from s

https://github.com/ClickHouse/ClickHouse/pull/14771

Near future in kuaishou

- Projections
  - Consistent materialized views at part level
- External table powered by elastic search
  - Used to serve updates and full text search
- Resource management

Thank You!