Altinity WEBINAR

STRENGTH IN NUMBERS:

INTRODUCTION TO CLICKHOUSE CLUSTER PERFORMANCE

with Robert Hodges

Introduction to Presenter

Robert Hodges - Altinity CEO

30+ years on DBMS plus virtualization and security.

ClickHouse is DBMS #20

Altinity

www.altinity.com

Leading software and services provider for ClickHouse

Major committer and community sponsor in US and Western Europe

Goals of the talk

- Introduce scaling axes of ClickHouse clusters
- Dig into distributed clusters
  - Using shards to scale writes
  - Using replicas to scale reads
- Describe handy tricks as well as common performance bottlenecks

Non-Goals:

- Boost performance of single nodes (though that’s important, too)
- Teach advanced ClickHouse performance management

Introduction to ClickHouse

Understands SQL

Runs on bare metal to cloud

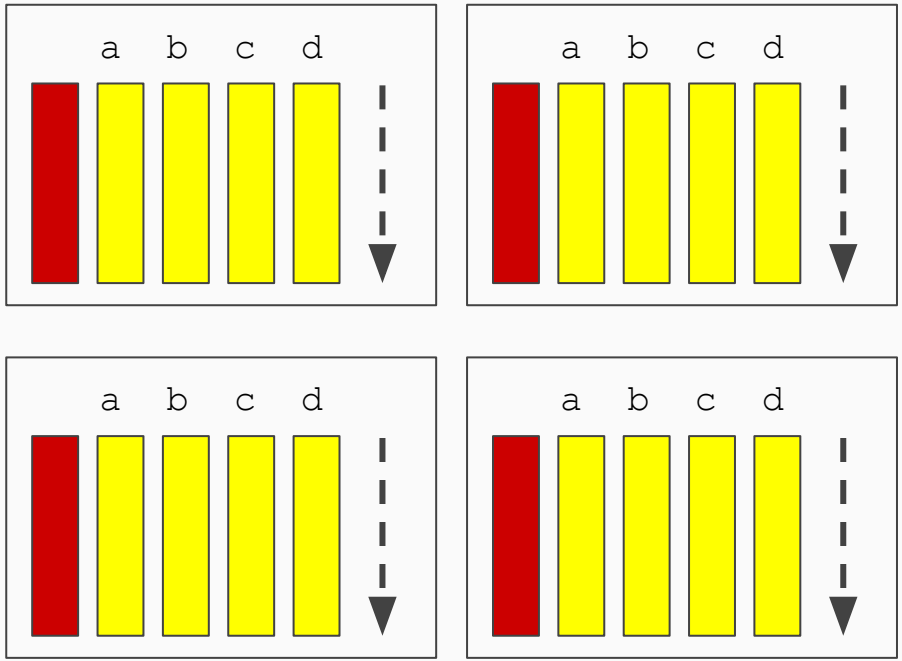
Stores data in columns

Parallel and vectorized execution

Scales to many petabytes

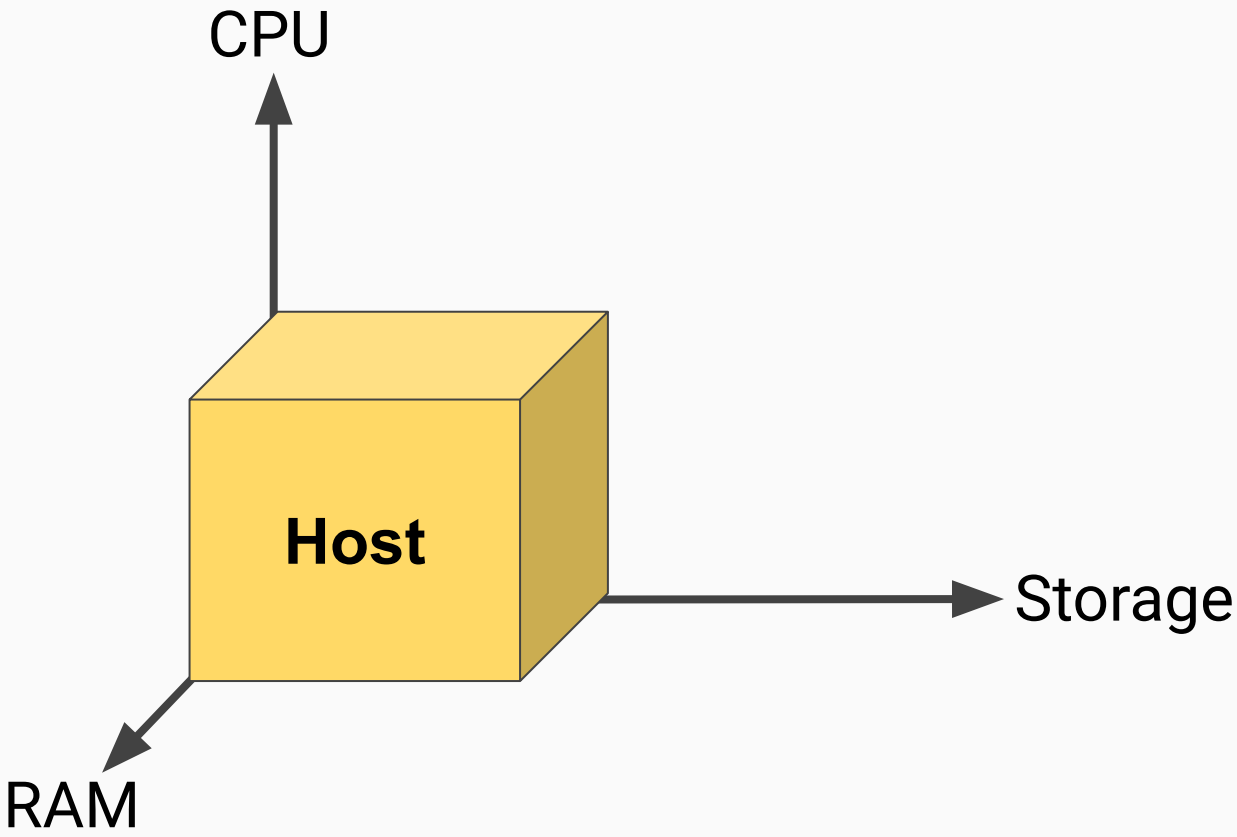
Is Open source (Apache 2.0)

Is WAY fast!

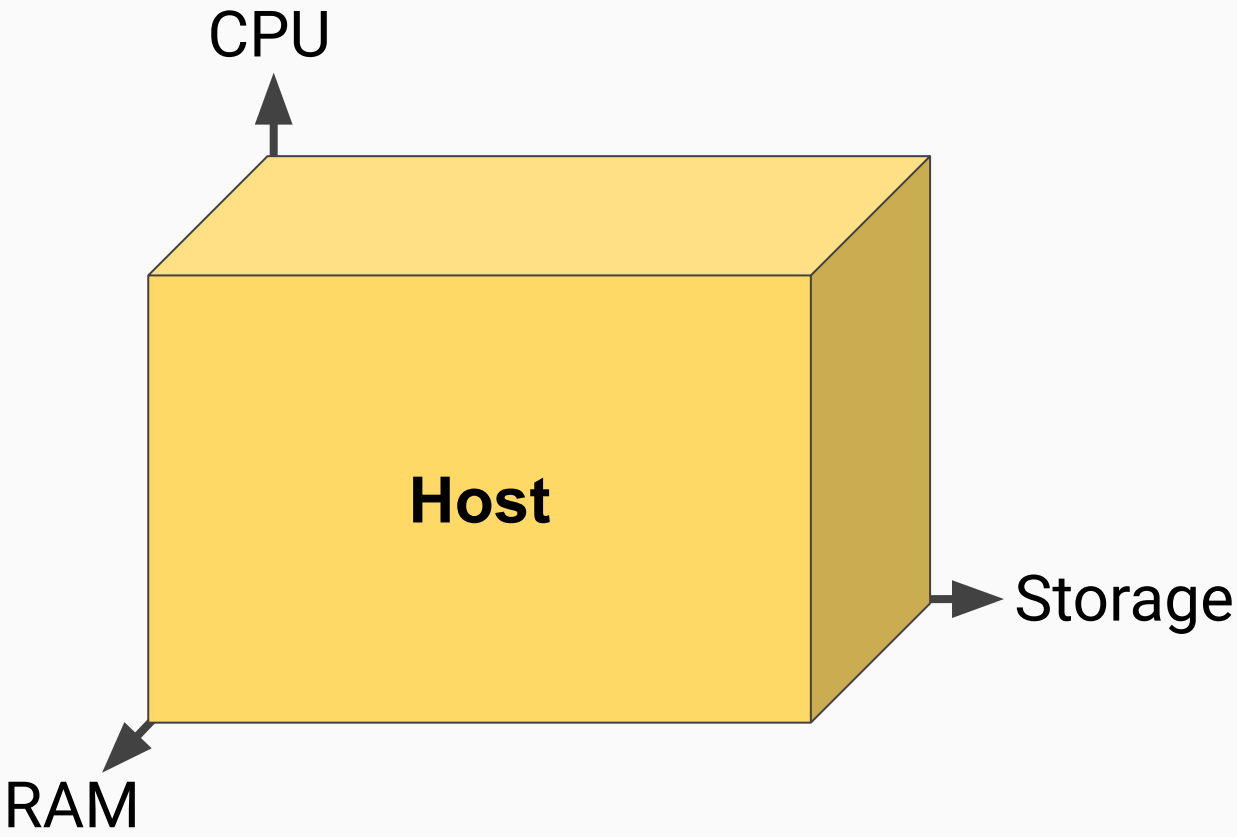


ClickHouse Cluster Model

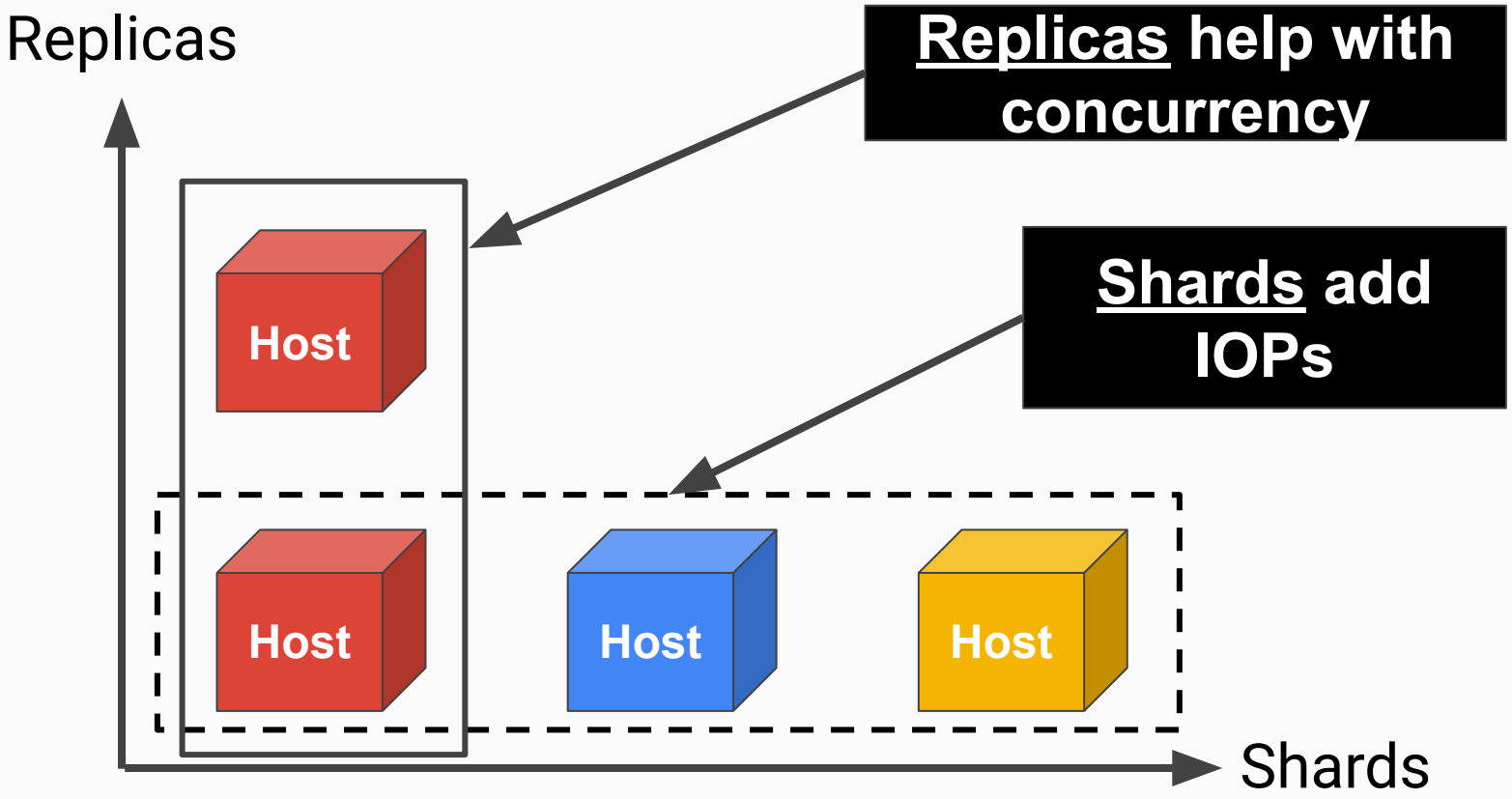
Clickhouse nodes can scale vertically



Clickhouse nodes can scale vertically

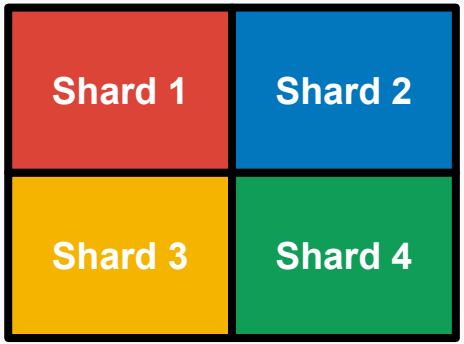


Clusters introduce horizontal scaling



Different sharding and replication patterns

All Sharded

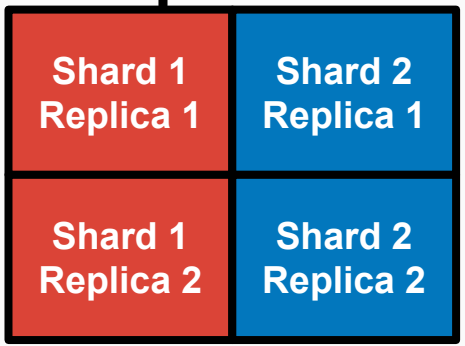


Data sharded 4 ways without replication

All Replicated

Data replicated 4 times without sharding

Sharded and Replicated



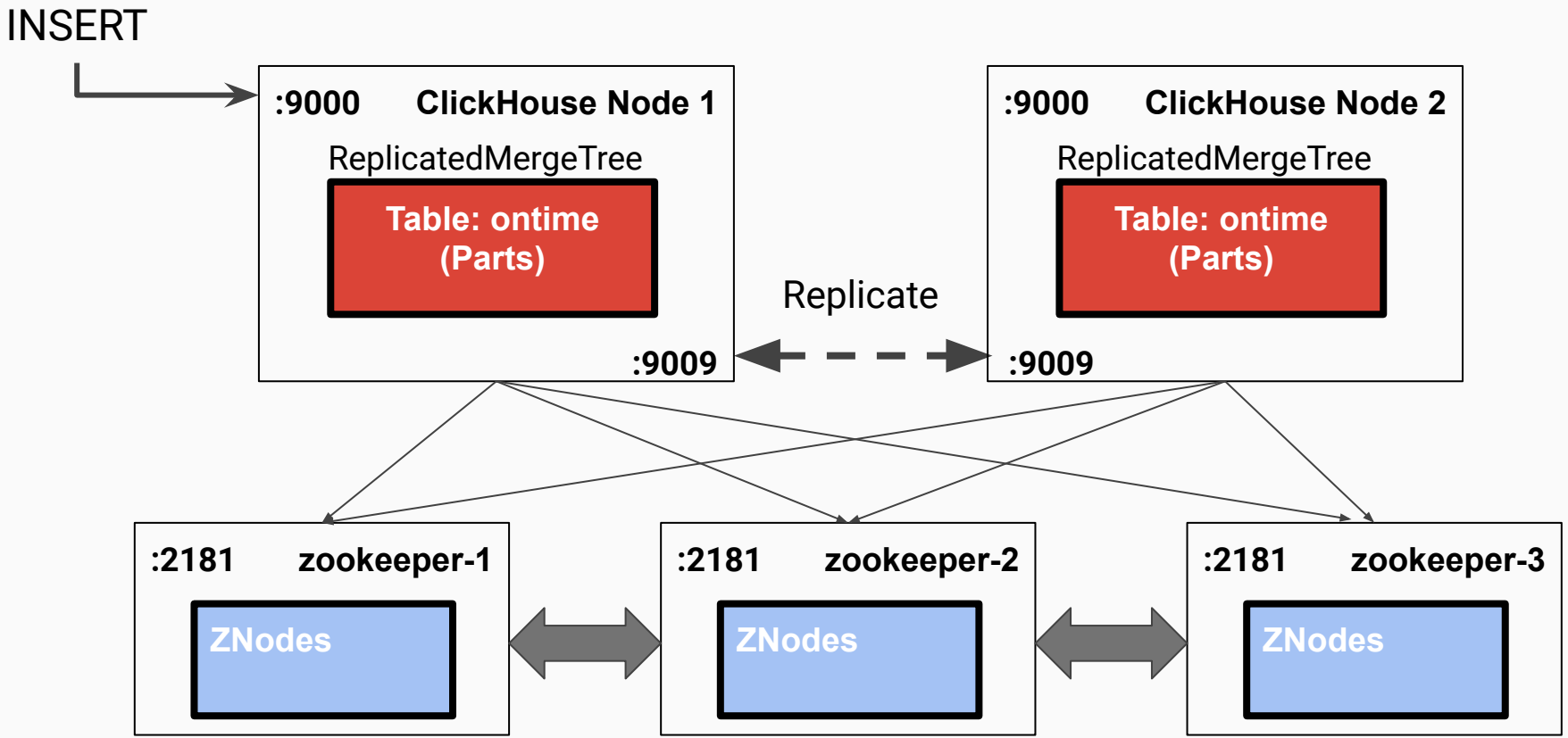
Data sharded 2 ways and replicated 2 times

Clusters define sharding and replication layouts

/etc/clickhouse-server/config.d/remote_servers.xml:

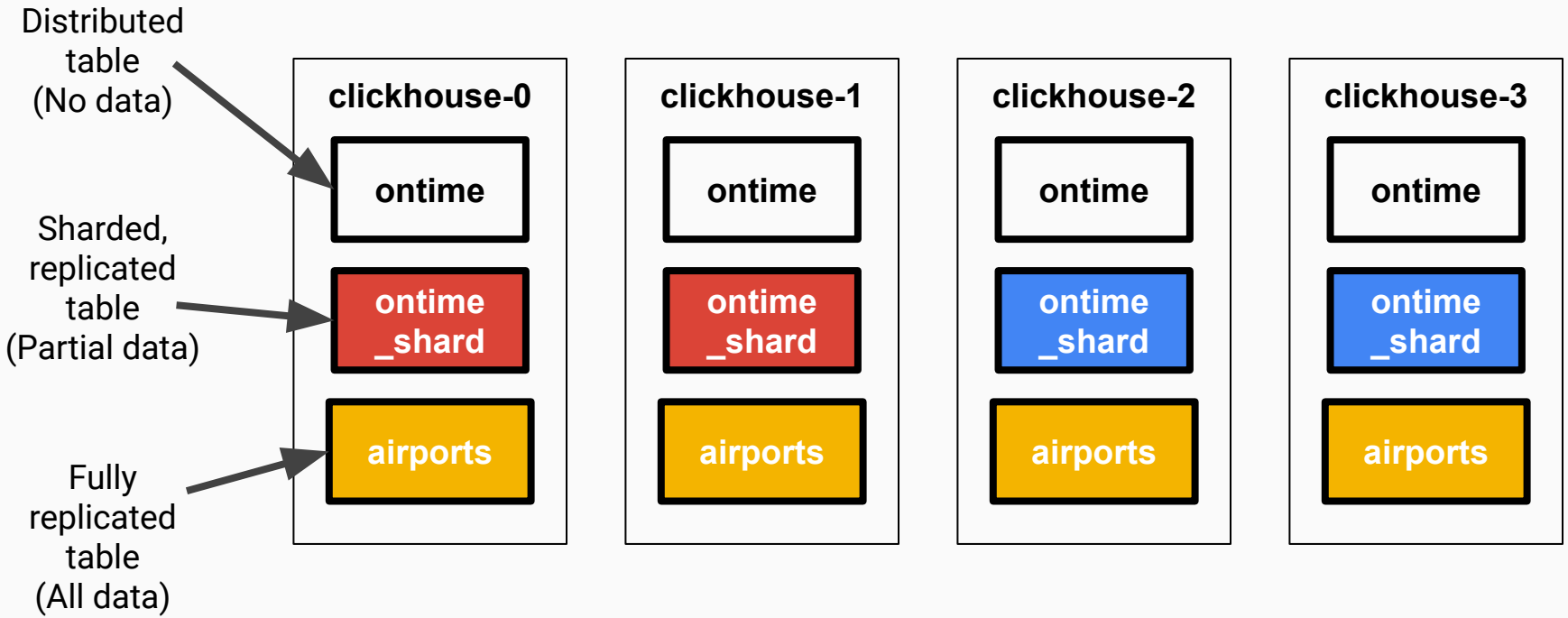
```
<yandex>
  <remote_servers>
    <ShardedAndReplicated>
      <shard>
        <replica><host>10.0.0.71</host><port>9000</port></replica>
        <replica><host>10.0.0.72</host><port>9000</port></replica>
        <internal_replication>true</internal_replication>
      </shard>
      <shard>
        . . .
      </shard>
    </ShardedAndReplicated>
  </remote_servers>
</yandex>
```

How ClickHouse uses Zookeeper



Cluster Performance in Practice

Setting up airline dataset on ClickHouse



Define sharded, replicated fact table

```
CREATE TABLE IF NOT EXISTS `ontime_shard` ON CLUSTER '{cluster}' (
  `Year` UInt16,
  `Quarter` UInt8,
  ...
)
Engine=ReplicatedMergeTree(
'/clickhouse/{cluster}/tables/{shard}/airline_shards/ontime_shard',
'{replica}')
PARTITION BY toYYYYMM(FlightDate)
ORDER BY (FlightDate, `Year`, `Month`, DepDel15)
```

Trick: Use macros to enable consistent DDL

/etc/clickhouse-server/config.d/macros.xml:

```
<yandex>
    <macros>
        <all>0</all>
        <cluster>demo</cluster>
        <shard>0</shard>
        <replica>clickhouse-0</replica>
    </macros>
</yandex>
```

Define a distributed table to query shards

```
CREATE TABLE IF NOT EXISTS ontime ON CLUSTER `{cluster}`
AS airline_shards.ontime_shard
ENGINE = Distributed(
  '{cluster}', airline_shards, ontime_shard, rand())
```

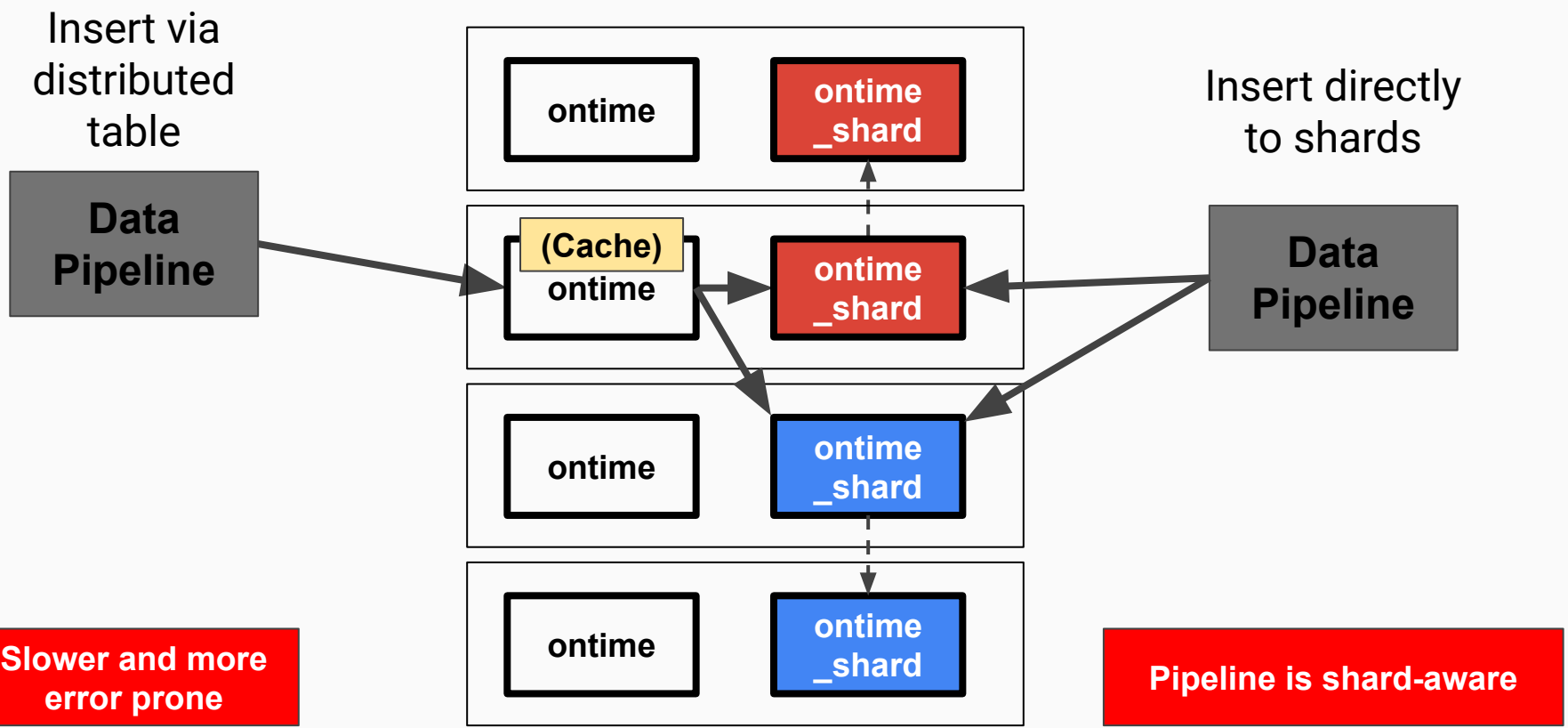
Define a fully replicated dimension table

```
CREATE TABLE IF NOT EXISTS airports ON CLUSTER 'all-replicated' (
  AirportID String,
  Name String,
  ...
)
Engine=ReplicatedMergeTree(
  '/clickhouse/{cluster}/tables/{all}/airline_shards/airports',
'{replica}')
PARTITION BY tuple()
PRIMARY KEY AirportID
ORDER BY AirportID
```

Overview of insertion options

- Local versus vs distributed data insertion
  - Local – no need to sync, larger blocks, faster
  - Distributed – sharding by ClickHouse
  - CHProxy -- distributes transactions across nodes
- Asynchronous (default) vs synchronous insertions
  - insert_distributed_sync
  - insert_quorum, select_sequential_consistency – linearization at replica level

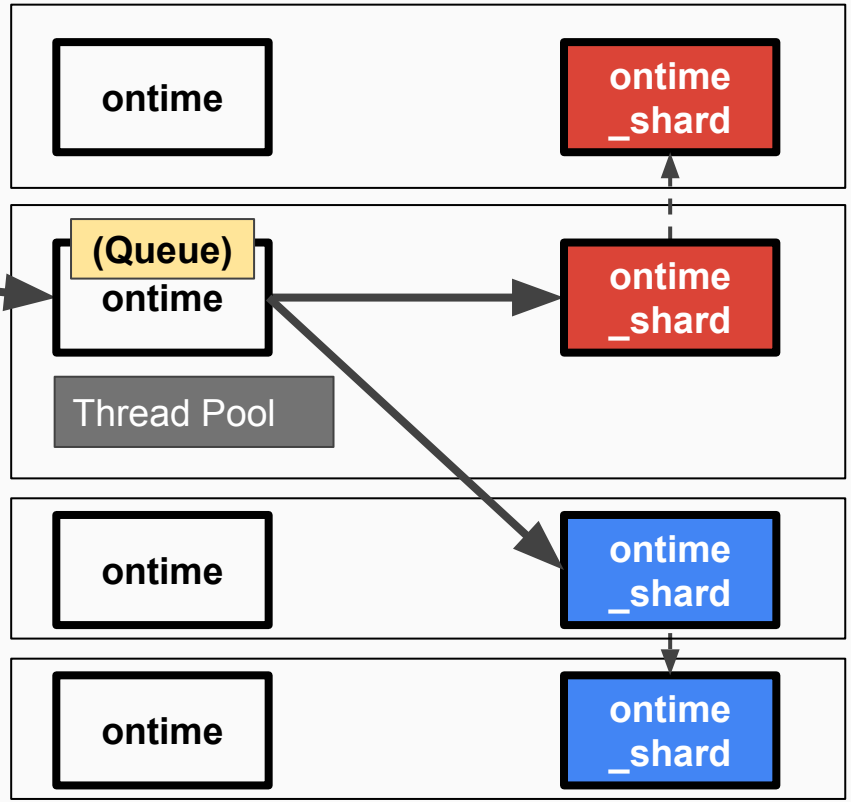
Distributed vs local inserts in action



Distributed insert semantics

Insert via distributed table

Data Pipeline

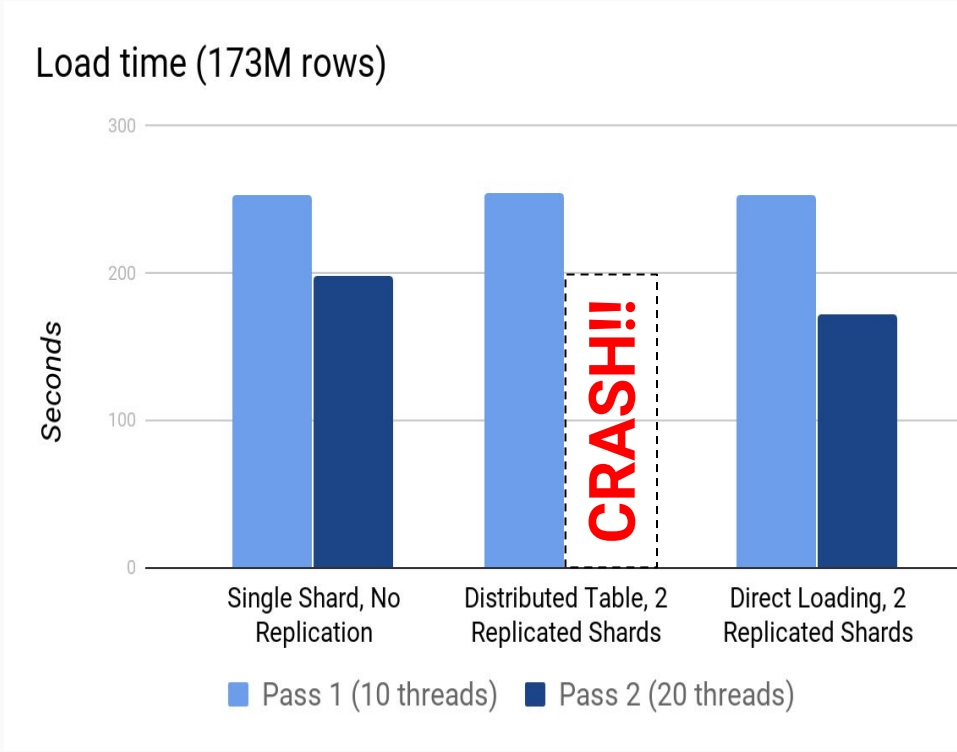


insert_distributed_sync:

- 0 = Async propagation
- 1 = Sync propagation

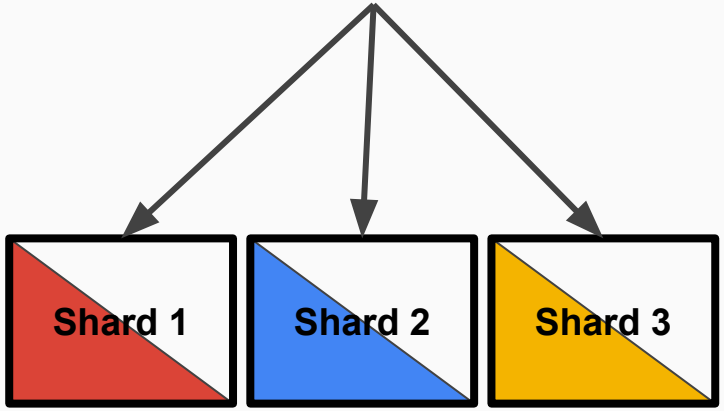
Testing cluster loading trade-offs

- With adequate I/O, RAM, CPU all load options have equal performance
- Direct loading is fastest for high volume feeds
- Loading via distributed table is most complex
  - Resource-inefficient
  - Can fail or lose data due to async insertion
  - May generate more parts
  - Requires careful monitoring



Selecting the sharding key

Randomized Key, e.g., cityHash64(Url)

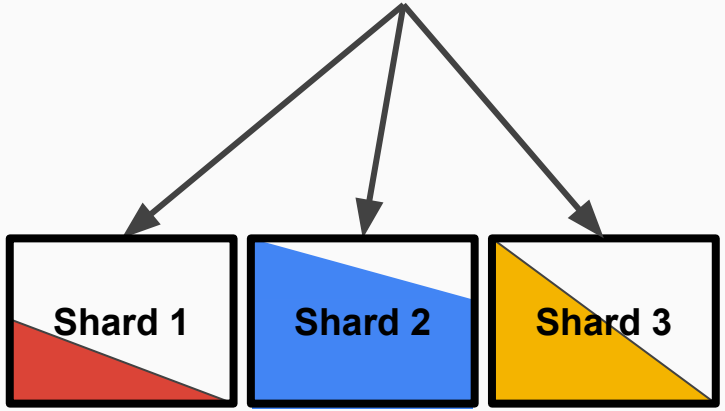


Must query all shards

Nodes are balanced

Easier to add nodes

Specific Key e.g., cityHash64(TenantId)

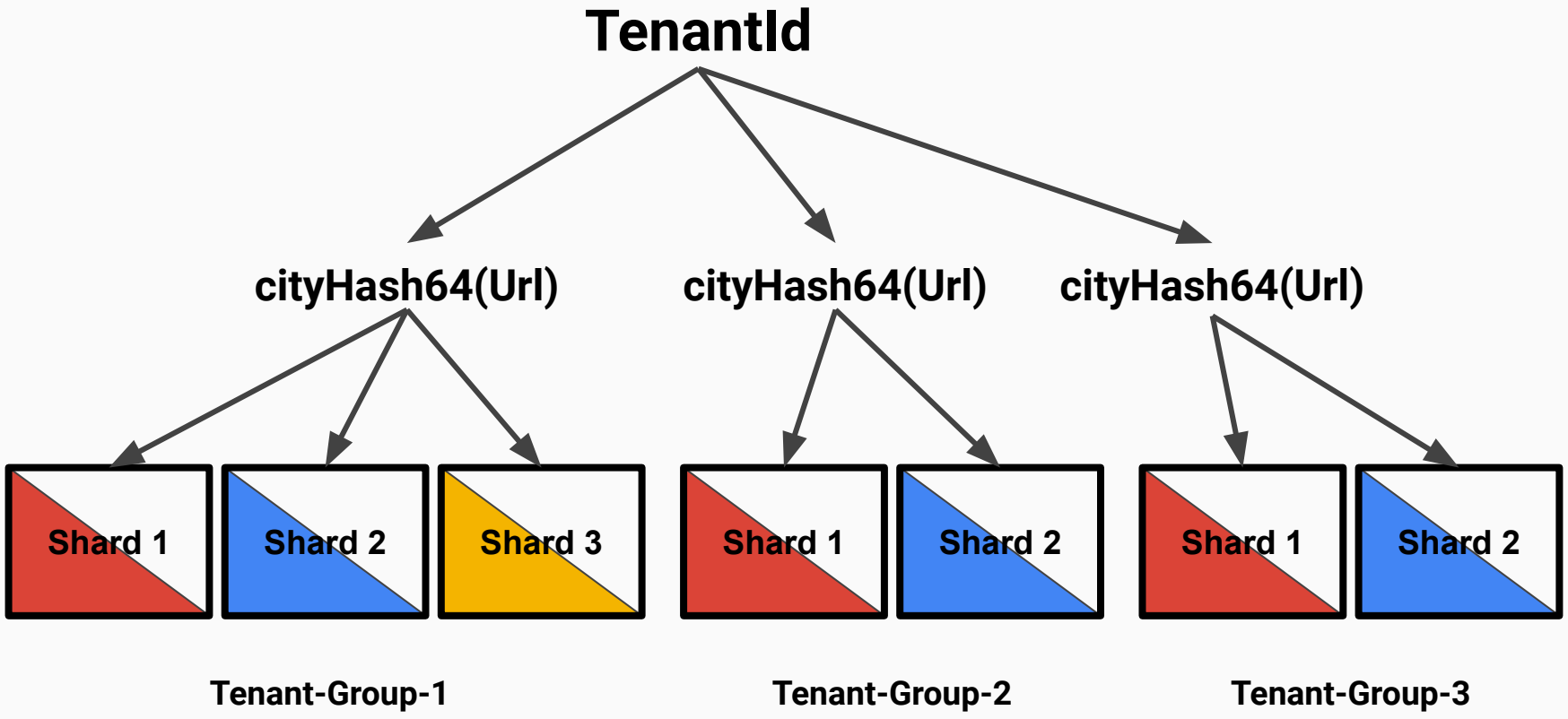


Unbalanced nodes

Hard to add nodes

Queries can skip shards

Bi-level sharding combines both approaches



Implement any sharding scheme via macros

/etc/clickhouse-server/config.d/macros.xml:

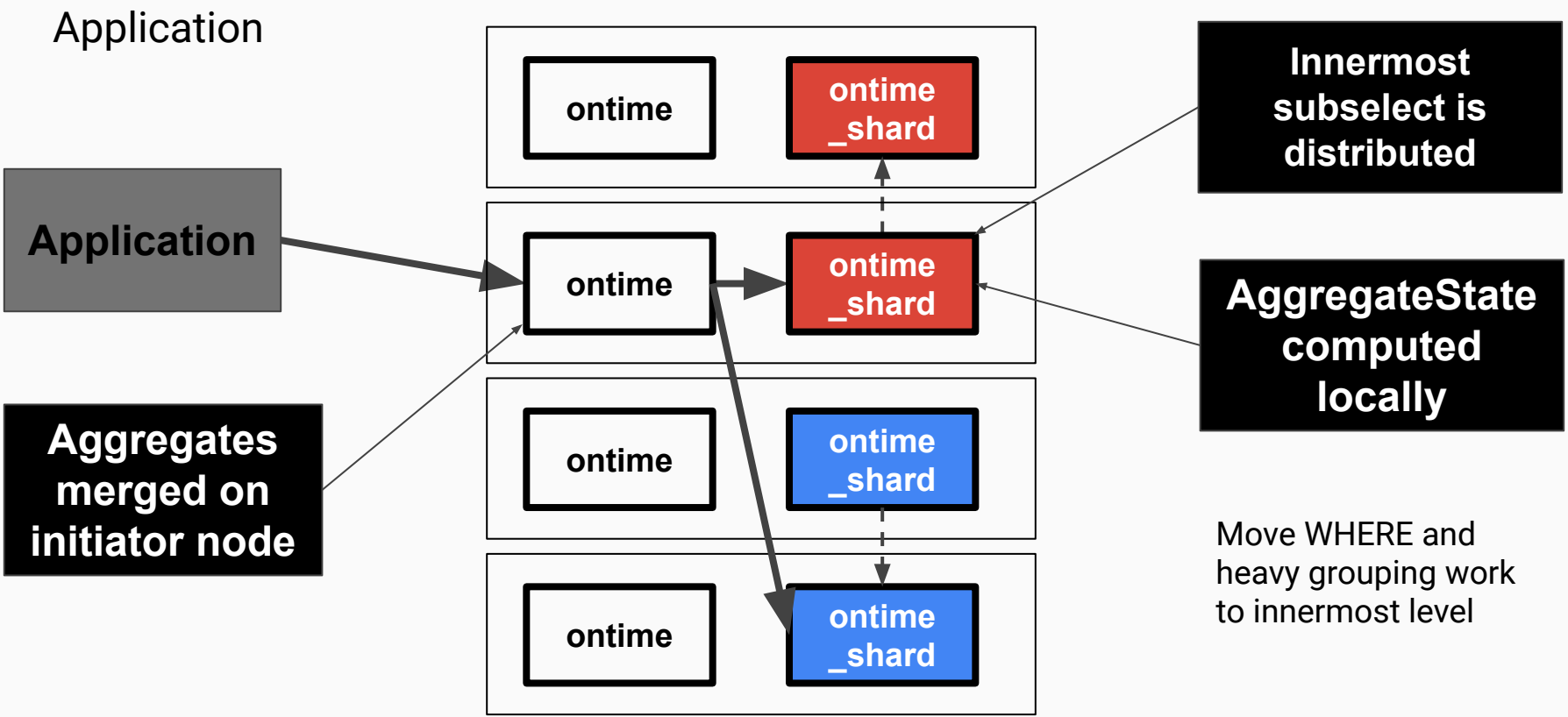
```
<yandex>
    <macros>
        <all>0</all>
        <cluster>demo</cluster>
        <group>2</group>
        <shard>0</shard>
        <replica>clickhouse-0</replica>
    </macros>
</yandex>
```

```
CREATE TABLE IF NOT EXISTS `ontime_shard` ON CLUSTER '{cluster}' (
. . .)
Engine=ReplicatedMergeTree(
'/clickhouse/{cluster}/tables/ {group}/{shard}/airline_shards/ontime_shard',
'{replica}')
```

Adding nodes and rebalancing data

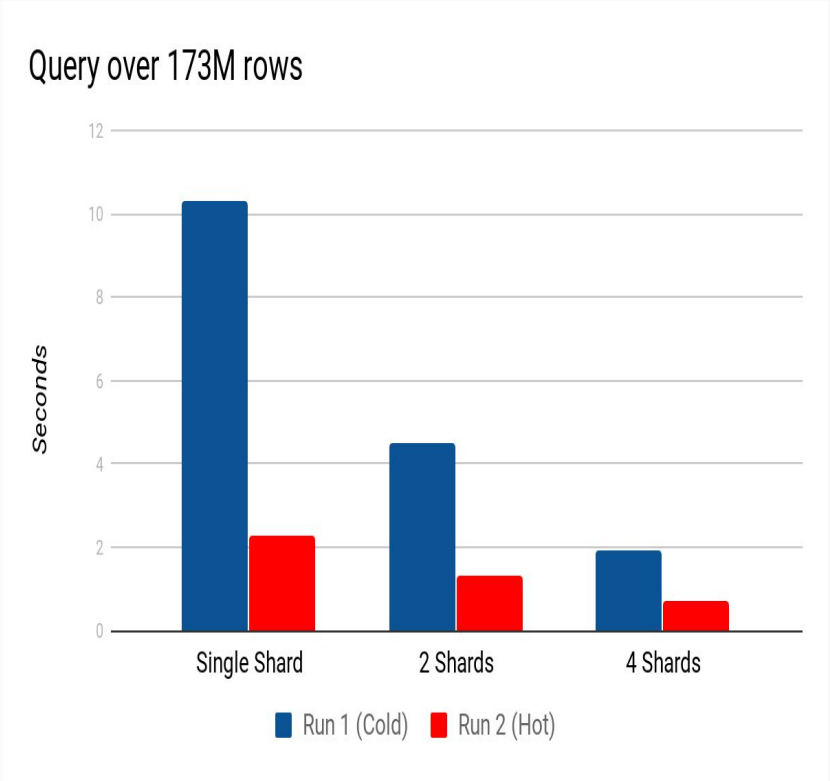
- To add servers:
  - Configure and bring up ClickHouse
  - Add schema
  - Add server to cluster definitions in remote_servers.xml, propagate to other servers
- Random sharding schemes allow easier addition of shards
  - Common pattern for time series--allow data to rebalance naturally over time
  - Use replication to propagate dimension tables
- Keyed partitioning schemes do not rebalance automatically
  - You can move parts manually using ALTER TABLE FREEZE PARTITION/rsync/ALTER TABLE ATTACH PARTITION
  - Other methods work too

How do distributed queries work?



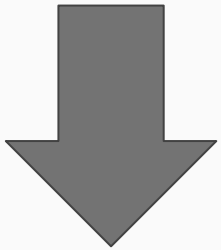
Read performance using distributed tables

- Best case performance is linear with number of nodes
- For fast queries network latency may dominate parallelization



ClickHouse pushes down JOINs by default

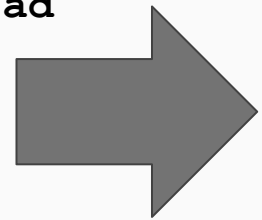
```
SELECT o.Dest d, a.Name n, count(*) c, avg(o.ArrDelayMinutes) ad
  FROM airline_shards_4.ontime o
    JOIN airline_shards_4.airports a ON (a.IATA = o.Dest)
    GROUP BY d, n HAVING c > 100000 ORDER BY d DESC
      LIMIT 10
```



```
SELECT Dest AS d, Name AS n, count() AS c, avg(ArrDelayMinutes) AS ad
  FROM airline_shards_4.ontime_shard AS o
    ALL INNER JOIN airline_shards_4.airports AS a ON a.IATA = o.Dest
      GROUP BY d, n HAVING c > 100000 ORDER BY d DESC LIMIT 10
```

...Unless the left side is a subquery

```
SELECT d, Name n, c AS flights, ad
FROM
(
  SELECT Dest d, count(*) c, avg(ArrDelayMinutes) ad
    FROM airline_shard_4.ontime
      GROUP BY d HAVING c > 100000
        ORDER BY ad DESC
)
LEFT JOIN airports ON airports.IATA = d
LIMIT 10
```



Remote Servers

Distributed product modes affect joins

- ‘Normal’ IN/JOIN – run subquery locally on every server
  - Many nodes – many queries, expensive for distributed
- GLOBAL IN/JOIN - run subquery on initiator, and pass results to every server
- Distributed_product_mode alters “normal” IN/JOIN behavior :
  - deny (default)
  - allow – run queries in ‘normal’ mode, distributed subquery runs on every server, if GLOBAL keyword is not used
  - local – use local tables for subqueries
  - global – automatically rewrite queries to ‘GLOBAL’ mode

Examples of IN operator processing

```
select foo from T1 where a in (select a from T2)
```

1) Subquery runs on a local table

```
select foo from T1_local
    where a in (select a from T2_local)
```

2) Subquery runs on every node

```
select foo from T1_local
    where a in (select a from T2)
```

3) Subquery runs on initiator node

```
create temporary table tmp Engine = Set AS select a from T2
select foo from T1_local where a in tmp;
```

Distributed query limitations and advice

- If joined table is missing, pushdown will fail
- Releases prior to 20.1 do not push down row-level security predicates
- Fully qualify table names to avoid syntax errors
- Distributed join behavior still somewhat limited

Settings to control distributed query

- **distributed_product_mode** -- How to handle joins of 2 distributed tables
- **enable_optimize_predicate_expression** -- Push down predicates
- **max_replica_delay_for_distributed_queries** -- Maximum permitted delay on replicas
- **load_balancing** -- Load balancing algorithm to choose replicas
- **prefer_localhost_replica** -- Whether to try to use local replica first for queries
- **optimize_skip_unused_shards** -- One of several settings to avoid shards if possible

(Plus others…)

Advanced Topics

Capacity planning for clusters

(Based on CloudFlare approach, see Resources slide below)

1. Test capacity on single nodes first
    a. Understand contention between INSERTs, background merges, and SELECTs
    b. Understand single node scaling issues (e.g., mark cache sizes)
2. If you can support your design ceiling with a single shard, stop here
    a. Ensure you have HA covered, though
3. Build the cluster
4. Test full capacity on the cluster
    a. Add shards to handle INSERTs
    b. Add replicas to handle SELECTs

Debugging slow node problems

Distributed queries are only as fast as the slowest node

Use the remote() table function to test performance across the cluster. Example:

```
SELECT sum(Cancelled) AS cancelled_flights
FROM remote('clickhouse-0', airline_shards_4.ontime_shard)

SELECT sum(Cancelled) AS cancelled_flights
FROM remote('clickhouse-1', airline_shards_4.ontime_shard)

...
```

A non-exhaustive list of things that go wrong

Zookeeper becomes a bottleneck (avoid excessive numbers of parts)

Choosing a bad partition key

Degraded systems

Insufficient monitoring

Wrap-up and Further Resources

Key takeaways

- Shards add read/write capacity over a dataset (IOPs)
- Replicas enable more concurrent reads
- Choose sharding keys and clustering patterns with care
- Insert directly to shards for best performance
- Distributed query behavior is more complex than MergeTree
- It's a big distributed system. **Plan for things to go wrong**

Well-managed clusters are extremely fast! Check your setup if you are not getting good performance.

Resources

- Altinity Blog
- Secrets of ClickHouse Query Performance -- Altinity Webinar
- ClickHouse Capacity Planning for OLAP Workloads by Mik Kocikowski of CloudFlare
- ClickHouse Telegram Channel
- ClickHouse Slack Channel

Thank you!

Special Offer:

Contact us for a
1-hour consultation!

Contacts:
info@altinity.com

Visit us at:
https://www.altinity.com

Free Consultation:
https://blog.altinity.com/offer