

ClickHouse 简介



宦传建

目录

- ClickHouse 是什么
- 主要特性
- 使用场景和限制
- 性能
- 表引擎
- 分片和副本
- 本地表和分布式表
- 用户接入
- Zookeeper
- 后续思考

ClickHouse 是什么

- 一款开源高性能，基于列式存储的 DBMS
- 可以实现准实时 OLAP 分析
- 由俄罗斯互联网公司 Yandex 开发，使用 C++ 语言
- 2016 年开源，基于 Apache 2 license

主要特性

- 列式存储
- 高性能：向量化执行（榨干 CPU）
- 支持 PB 级数据
- 支持水平扩展、多副本容错
- 数据压缩
- HDD 优化等

列式存储

- 块遍历：减少查询过程中函数调用次数
- 压缩：相同类型的数据可以使用数组保存，类似的数据存储在一起，压缩比更高
- 延迟物化：将物化 (Materialization) 的过程尽量拖延到整个查询生命周期的后期



向量化执行引擎

列式存储是实现基础

- 可以充分利用 CPU 缓存
- 可以利用 SIMD 技术，充分释放多核并行处理能力

```
#if defined(_SSE2_) && defined(_POPCNT_)
    const __m128i zero16 = _mm_setzero_si128();
    const Int8 * end64 = pos + filt.size() / 64 * 64;

    for (; pos < end64; pos += 64)
        count += __builtin_popcountll(
            static_cast<UInt64>(_mm_movemask_epi8(_mm_cmplt_epi8(
                _mm_loadu_si128(reinterpret_cast<const __m128i *>(pos)),
                zero16))
            | (static_cast<UInt64>(_mm_movemask_epi8(_mm_cmplt_epi8(
                _mm_loadu_si128(reinterpret_cast<const __m128i *>(pos + 16)),
                zero16)) << 16)
            | (static_cast<UInt64>(_mm_movemask_epi8(_mm_cmplt_epi8(
                _mm_loadu_si128(reinterpret_cast<const __m128i *>(pos + 32)),
                zero16)) << 32)
            | (static_cast<UInt64>(_mm_movemask_epi8(_mm_cmplt_epi8(
                _mm_loadu_si128(reinterpret_cast<const __m128i *>(pos + 48)),
                zero16)) << 48));

    /// TODO Add duff device for tail?
#endif
```

ck 主要是基于 SSE2 扩展指令集实现 SIMD

适用场景和限制

When to use ClickHouse	✓ Web and App analytics	✓ Monitoring and telemetry
For analytics over a stream of clean, well structured and <u>immutable events or logs</u> . It is recommended to put each such stream into a <u>single wide fact table with pre-joined dimensions</u> .	✓ Advertising networks and RTB	✓ Time series
When NOT to use ClickHouse	✗ Transactional workloads (OLTP)	✗ Blob or document storage
	✗ Key-value requests with a high rate	✗ Over-normalized data

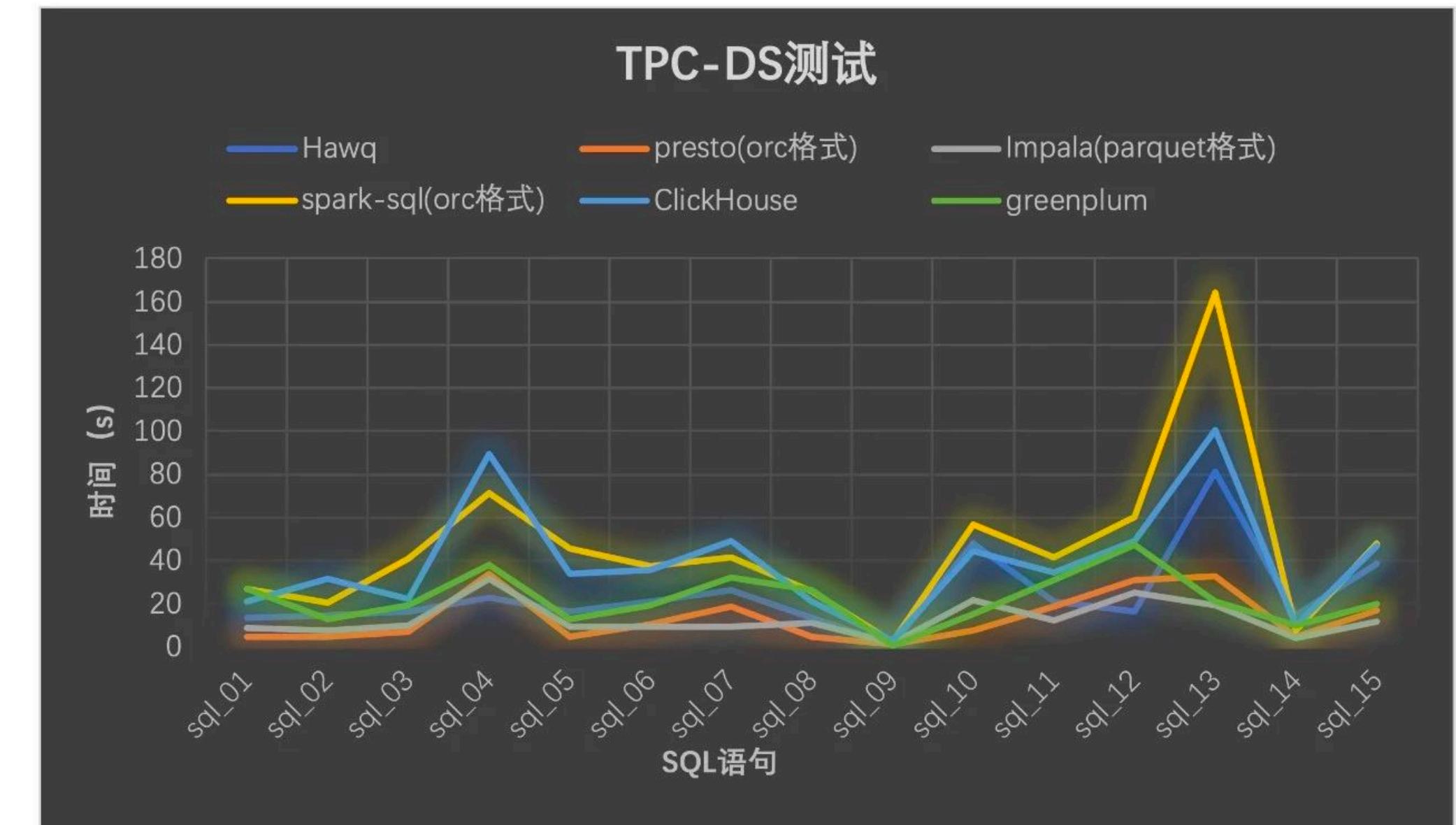
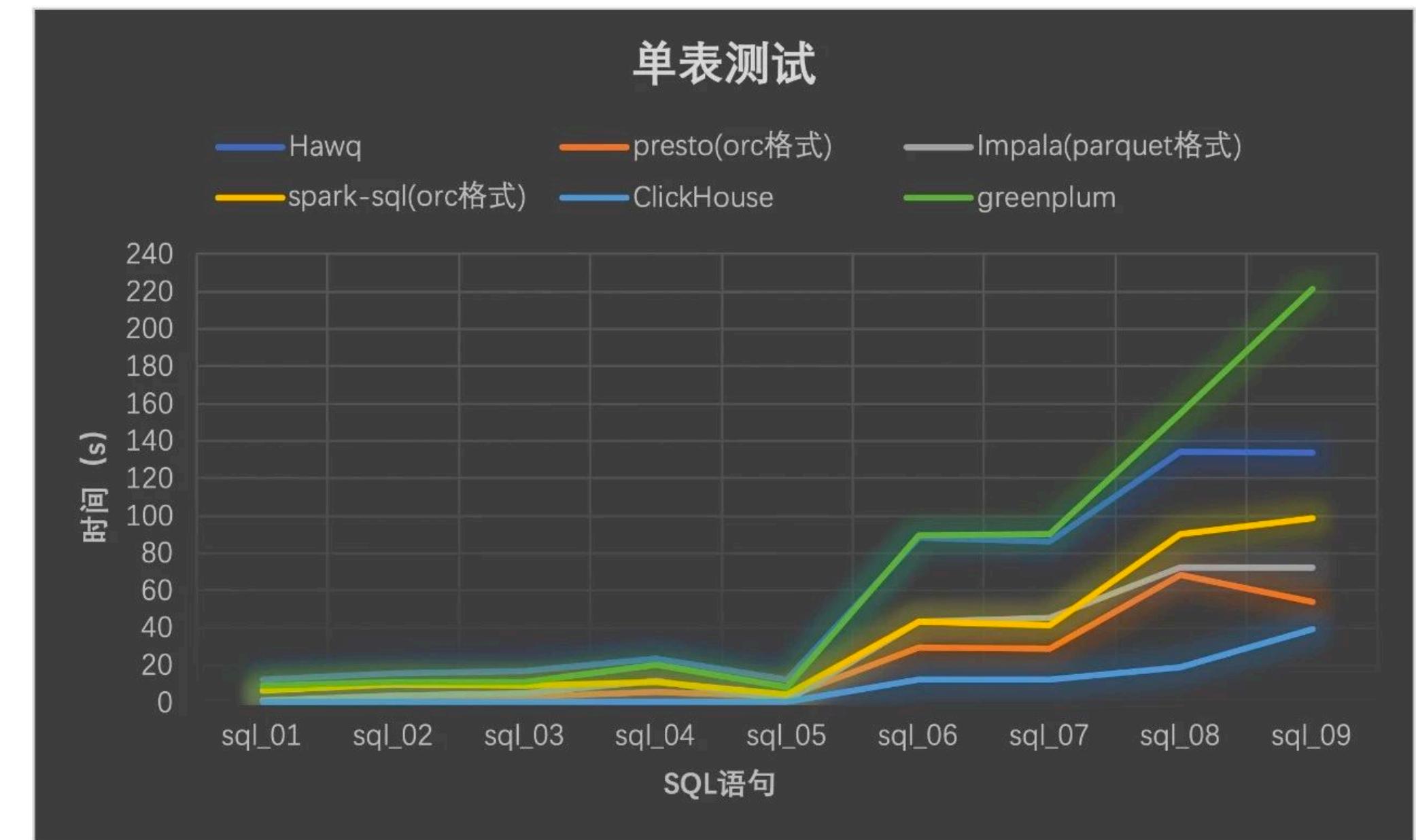
- 不适合：高并发场景、频繁更新场景
- 单表查询性能较高，依赖宽表设计，多表 join 和分布式 join 性能下降较多
- 非标准 SQL
- 适合大批量数据实时写入，如：用户行为埋点分析

查询性能

- 3台机器（配置较低）
- 测试数据集大小 100G
- TPC-DS 测试是多表关联查询

服务器	cpu核数	cpu线程数	内存大小	磁盘空间
server1	4	16	64g	2T
server2	4	16	64g	2T
server3	4	16	64g	2T

数据来源：易观分析 - 开源 OLAP 引擎评测

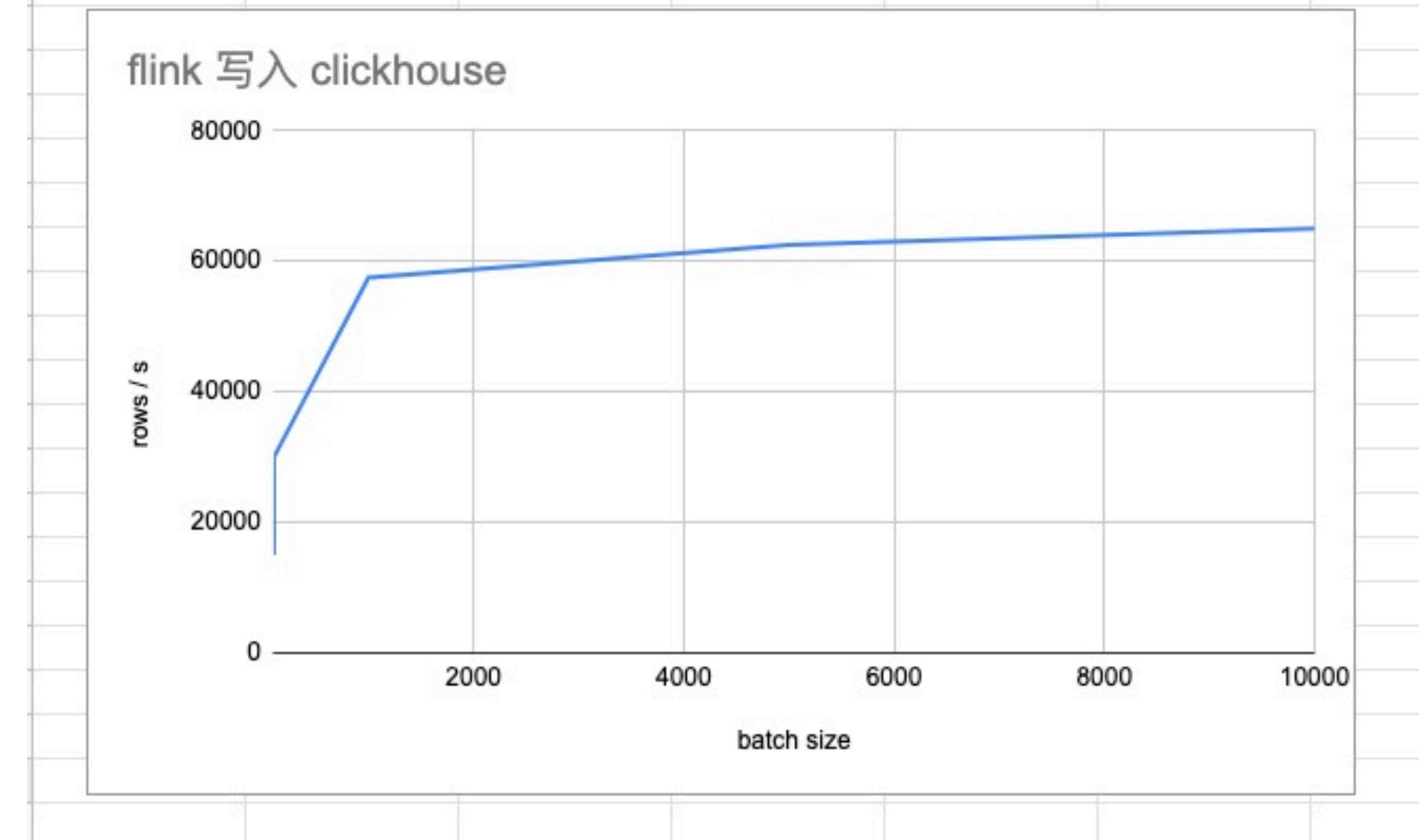


写入性能



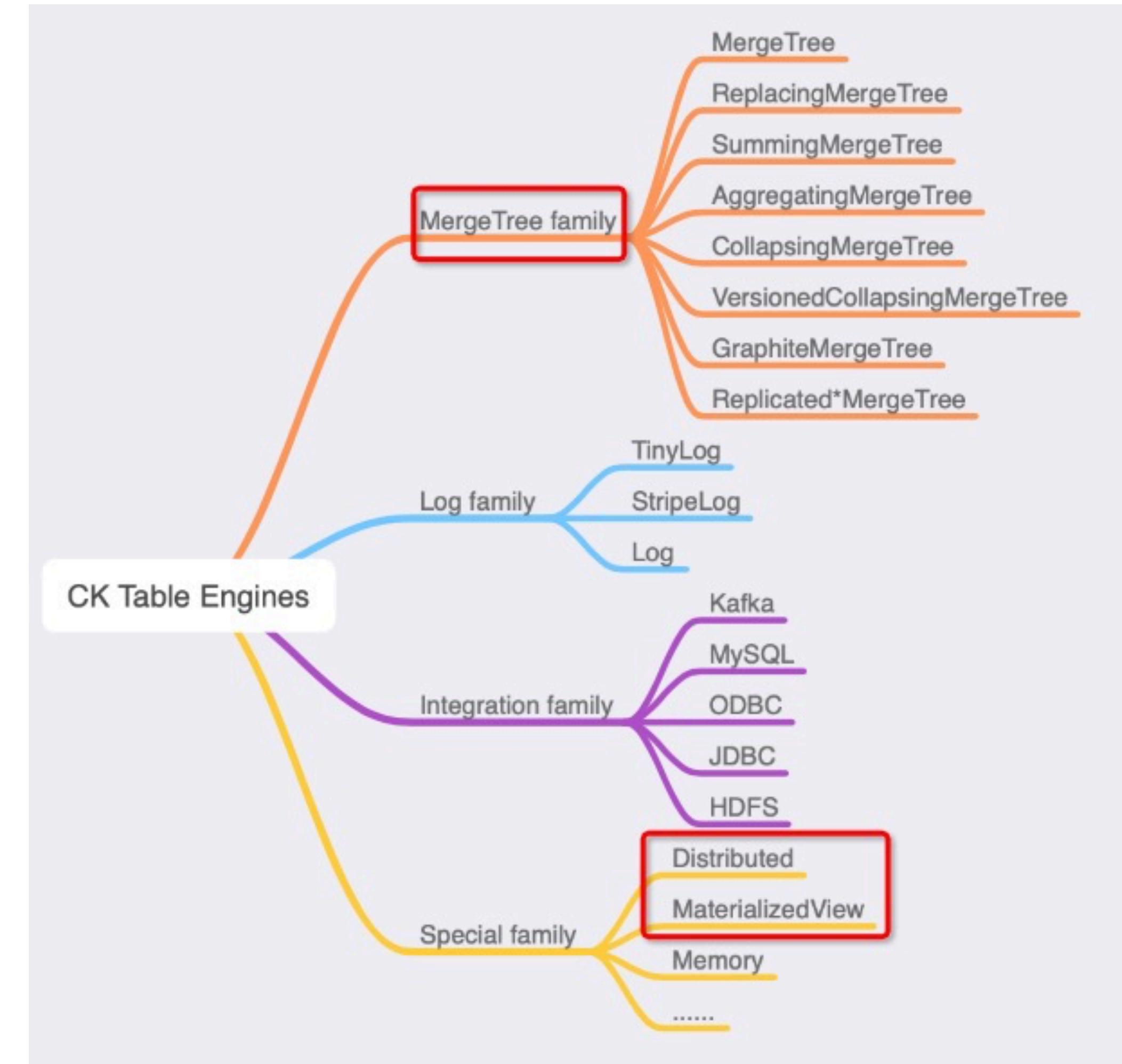
- 测试表 100 列，单行大小 2.5k
- flink 分布式写入 ck 单节点
- flink 5个 worker, 每个节点 3G内存
- 持续写入 1亿条数据, 磁盘 io 无压力

A	B	C	D	E	F	G
batch size	rows /s	MB /s	备注			
100	15000	37.5	后台有查询			
100	30000	75				
1000	57500	143.7				
5000	62500	156.2				
10000	65000	162.5				



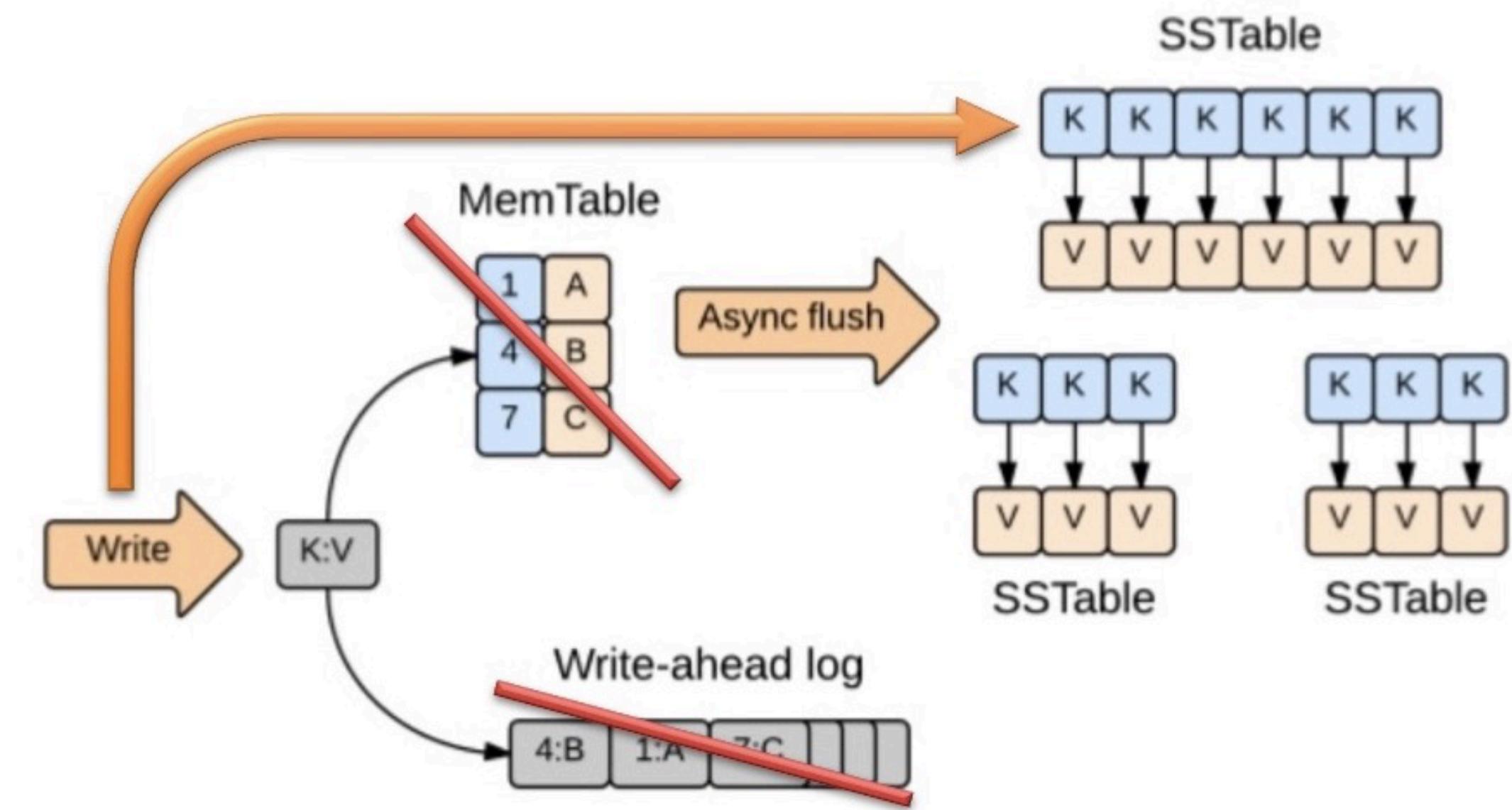
表引擎家族

- MergeTree 是官方主推的表引擎，代表 CK 的核心功能，支持复制
- Integration 系列主要用于将外部数据导入 CK
- Special 系列中的 Memory 适合没有持久化需求的临时表（小于1亿）



MergeTree

- MergeTree \approx LSM Tree
- 没有 MemTable 和 WAL
- SSTable 在 ck 中称为 “parts”
- Compact SSTable 在 ck 中称为 “merge”



MergeTree 示例

- 如果没有显示指定主键，CK 使用排序键作为主键
- index_granularity 默认值是 8k，是指 CK 会对 parts 进行逻辑切分，称为一个颗粒(granule)，8k 就是一个颗粒包含的行数

延伸：insert into 执行后发生了什么？

```
1 CREATE TABLE test.event_log (
2     event_time DateTime,
3     user_id UInt64,
4     event_type String,
5     site_id UInt64
6 ) ENGINE = MergeTree()
7 PARTITION BY toYYYYMMDD(event_time)
8 ORDER BY (user_id,site_id)
9 SETTINGS index_granularity = 8192;
10
11 INSERT INTO test.event_log VALUES
12 ('2020-09-14 12:00:00',12345678,'appStart',16789),
13 ('2020-09-14 12:00:01',12345679,'appStart',26789);
14 INSERT INTO test.event_log VALUES
15 ('2020-09-14 13:00:00',22345678,'openGoodsDetail',16789),
16 ('2020-09-14 13:00:01',22345679,'buyNow',26789);
```

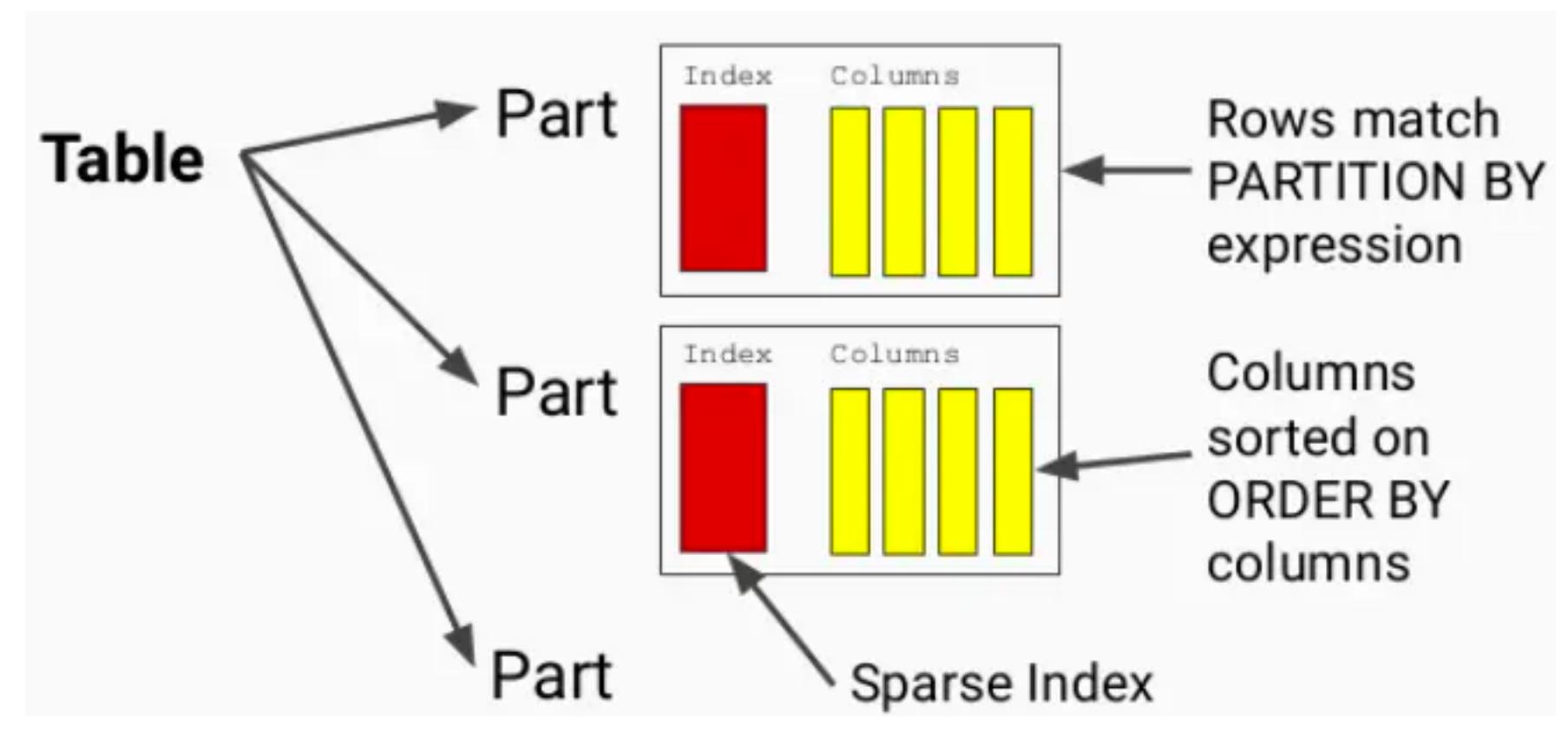
全部数据:	[-----]
id:	[aaaaaaaaaaaaaaaaaaaaabbbbcdeeeeeeeeeeeefggggggggghhhhhhhhiiiiiiiiklllllll]
num:	[11111112222223331233211112222233321111112122222231111222331112233]
标记:	
	a,1 a,2 a,3 b,3 e,2 e,3 g,1 h,2 i,1 i,3 l,3
标记号:	0 1 2 3 4 5 6 7 8 9 10

(id, num) 为主键

id in ('a', 'g') 那么 ck 会在 [0, 3] 和 6 里检索数据

Part

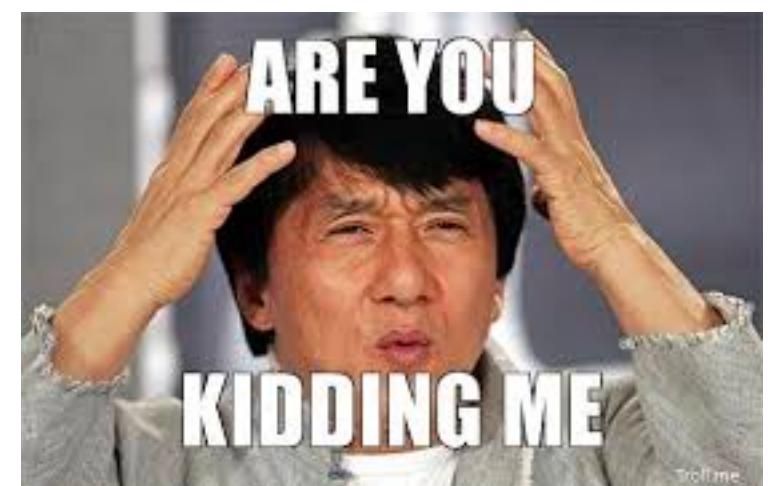
- 每次写入会产生一个 part，后台合并（相同分区）
- part 分为 Wide 和 Compact 类型
- Compact 存储形式有效减少了文件数量
- 频繁的写入会导致短时间内产生大量小 parts，导致后台合并压力过大。（官方建议**低频大批量**写入，例如：写入前预分组、每个 batch 10w 条记录、每秒不超过1次）



```
[clickhouse@SHTL009046112 ~]$ tree /var/lib/clickhouse/data/test/test_event_log/
/var/lib/clickhouse/data/test/test_event_log/
└── 20210101_1_1_0
    ├── checksums.txt
    ├── columns.txt
    ├── count.txt
    ├── default_compression_codec.txt
    ├── event_time.bin
    ├── event_time.mrk2
    ├── event_type.bin
    ├── event_type.mrk2
    ├── minmax_event_time.idx
    ├── partition.dat
    ├── primary.idx
    ├── site_id.bin
    ├── site_id.mrk2
    ├── user_id.bin
    └── user_id.mrk2
    └── detached
    └── format_version.txt
```

其他 MergeTree 引擎

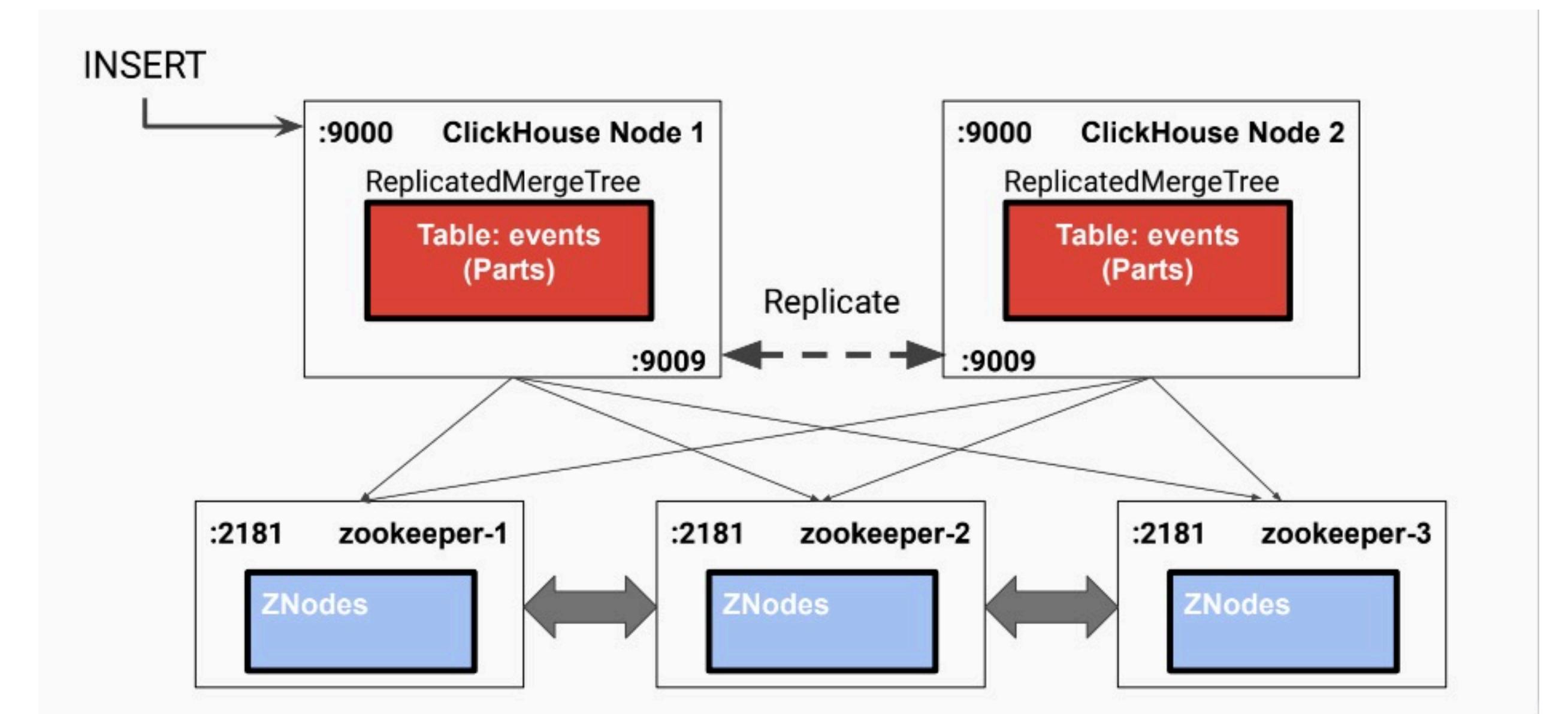
- ReplacingMergeTree: 可以解决主键去重问题, 限制: 跨节点数据无法去重、合并时间点不确定、不能保证没有重复数据。 (只能确保最终去重)
- CollapsingMergeTree: 可以解决 ReplacingMergeTree 无法即时删除的问题, 限制: 跨节点数据无法去重、合并时间点不确定、需要业务结合标记位查询、标记位乱序将导致无法删除
- VersionedCollapsingMergeTree: 针对上述再加版本位来解决乱序无法删除的问题
- SummingMergeTree: 支持对主键列进行预聚合, 在后台合并时, 会对主键相同的行进行 sum, 适合结合明细表做预计算加速
- AggregatingMergeTree: 相比 sum 预计算, 支持更丰富的聚合函数, 但也更难用。



如何实现水平扩展、高可用？

分片和副本

- 配置定义集群，节点间不需要相互感知
- 分片实现**异步多主复制**（依赖 zk 协调）



```
HZPL127160 :) select * from system.clusters;  
SELECT *  
FROM system.clusters  
Query id: de52f1e2-cc4a-44c6-b783-42d0e0b56058
```

cluster	shard_num	shard_weight	replica_num	host_name	host_address	port	is_local	user	default_database	errors_count	estimated_recovery_time
ckcluster	1	1	1	HZPL127160	192.168.127.160	9000	1	default		0	0
ckcluster	1	1	2	HZPL127161	192.168.127.161	9000	0	default		0	0
ckcluster	2	1	1	HZPL127162	192.168.127.162	9000	0	default		0	0
ckcluster	2	1	2	HZPL127163	192.168.127.163	9000	0	default		0	0
ckcluster	3	1	1	HZPL127164	192.168.127.164	9000	0	default		0	0
ckcluster	3	1	2	HZPL127165	192.168.127.165	9000	0	default		0	0

本地表和分布式表

本地表

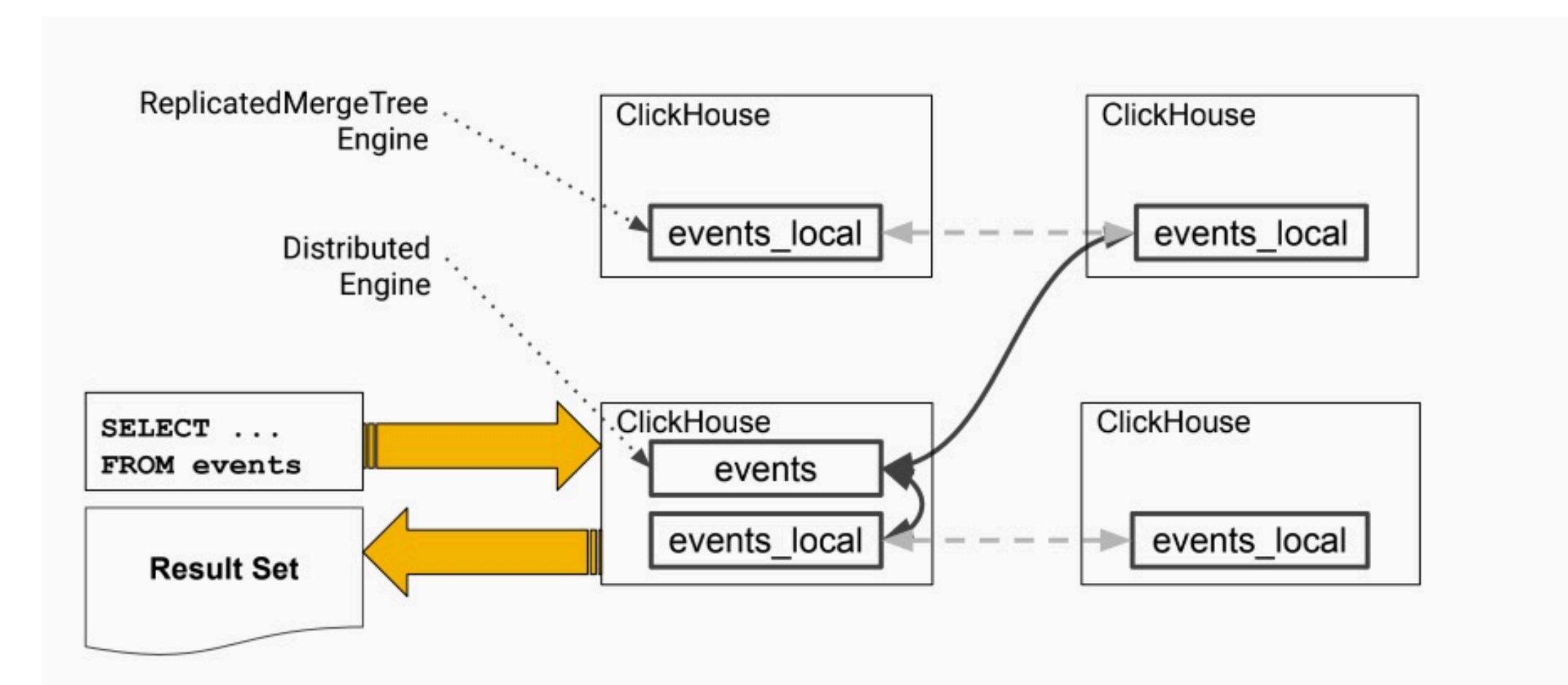
- 分布式 DDL，可以批量创建
- zk 上的表元数据 path，副本标识

```
— 本地表
CREATE TABLE IF NOT EXISTS default.events_local
    ON CLUSTER '{cluster}'
(
    `tenant_id` UInt32,
    `alert_id` String,
    `timestamp` DateTime,
    `alert_data` String,
    `acked` UInt8 DEFAULT 0,
    `ack_time` DateTime DEFAULT toDateTime(0),
    `ack_user` LowCardinality(String) DEFAULT ''
)
ENGINE = ReplicatedReplacingMergeTree('/clickhouse/tables/{shard}/default/events_local', '{replica}')
ORDER BY (tenant_id, timestamp, alert_id)
SETTINGS index_granularity = 8192;

— 分布式表
CREATE TABLE IF NOT EXISTS default.events_all
    ON CLUSTER '{cluster}'
    AS default.events_local
ENGINE = Distributed'{cluster}', 'default', 'events_local', rand()
```

分布式表

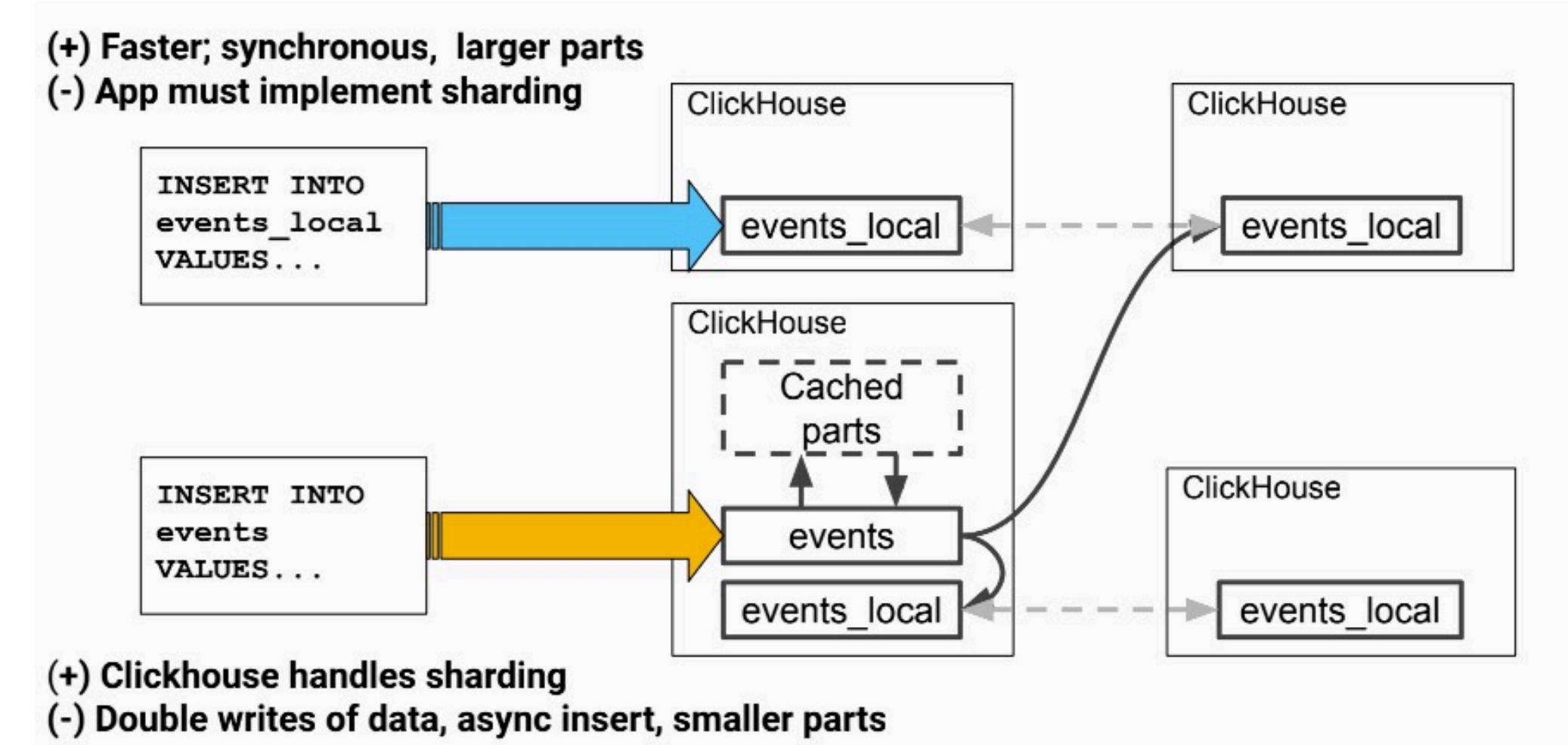
- 基于 Distributed 表引擎
- 实际不存储数据、是本地表的完整视图
- rand() 是 insert 分布式表的 sharding key , 生产环境禁止写分布式表



分布式写入

写分布式表

- 优点：操作简单、由 ck 控制 sharding
- 缺点：写入被放大，浪费资源；会产生较多的小 parts，使合并压力变大；数据写入是异步的，短时间内会有不一致。



写本地表（**推荐**）

- 优点：高效的同步写入，合适的 parts 大小
- 缺点：需要应用层实现 sharding 和路由逻辑

应用如何接入?

chproxy

[English](#) | [简体中文](#)

Chproxy 是一个用于 [ClickHouse](#) 数据库的 http 代理、负载均衡器。具有以下特性：

- 支持根据输入用户代理请求到多个 ClickHouse 集群。比如，把来自 `appserver` 的用户请求代理到 `stats-raw` 集群，把来自 `reportserver` 用户的请求代理到 `stats-aggregate` 集群。
- 支持将输入用户映射到每个 ClickHouse 实际用户，这能够防止暴露 ClickHouse 集群的真实用户名、密码信息。此外，chproxy 还允许映射多个输入用户到某一个单一的 ClickHouse 实际用户。
- 支持接收 HTTP 和 HTTPS 请求。
- 支持通过 IP 或 IP 掩码列表限制 HTTP、HTTPS 访问。
- 支持通过 IP 或 IP 掩码列表限制每个用户的访问。
- 支持限制每个用户的查询时间，通过 [KILL QUERY](#) 强制杀执行超时或者被取消的查询。
- 支持限制每个用户的请求频率。
- 支持限制每个用户的请求并发数。
- 所有的限制都可以对每个输入用户、每个集群用户进行设置。
- 支持自动延迟请求，直到满足对用户的限制条件。
- 支持配置每个用户的[响应缓存](#)。
- 响应缓存具有内建保护功能，可以防止 [惊群效应 \(thundering herd\)](#)，即 dogpile 效应。
- 通过 `least loaded` 和 `round robin` 技术实现请求在副本和节点间的均衡负载。
- 支持检查节点健康情况，防止向不健康的节点发送请求。
- 通过 [Let's Encrypt](#) 支持 HTTPS 自动签发和更新。
- 可以自行指定选用 HTTP 或 HTTPS 向每个配置的集群代理请求。
- 在将请求代理到 ClickHouse 之前，预先将 User-Agent 请求头与远程/本地地址，和输入/输出的用户名进行关联，因此这些信息可以在 `system.query_log.http_user_agent` 中查询到。
- 暴露各种有用的符合 [prometheus](#) 内容格式的[指标 \(metrics\)](#)。
- 支持配置热更新，配置变更无需重启——只需向 `chproxy` 进程发送一个 `SIGHUP` 信号即可。
- 易于管理和运行——只需传递一个配置文件路径给 `chproxy` 即可。
- 易于配置：



就这?



Zookeeper

- 实现分布式 DDL 和 Replicated*MergeTree 数据同步

推荐

- 为 ck 部署单独的 zk 集群，生产 3 或 5 个节点
- 使用 SSD 存储 zk log
- 禁止手动清理 zk 数据
- 使用 snapshot 备份 zk 数据
- 开启自动清理

后续思考

- 运维复杂度高
- 高度依赖 zk
- C++ 源码、数据结构、体系结构
- 监控系统
- 业务 + 表引擎使用
-

The End

