Clickhouse在众安的应用实践
百亿保险数据实时分析探索

众安保险
数据智能中心
蒙强
2019年10月27日

众安保险

- 成立于2013年，是中国第一家互联网保险公司。

- 互联网保险特点：

1. 场景化
2. 高频化
3. 碎片化

- 今年上半年众安上半年服务用户3.5亿，销售保单33.3亿张。

CHAPTER 01

报表系统的现状

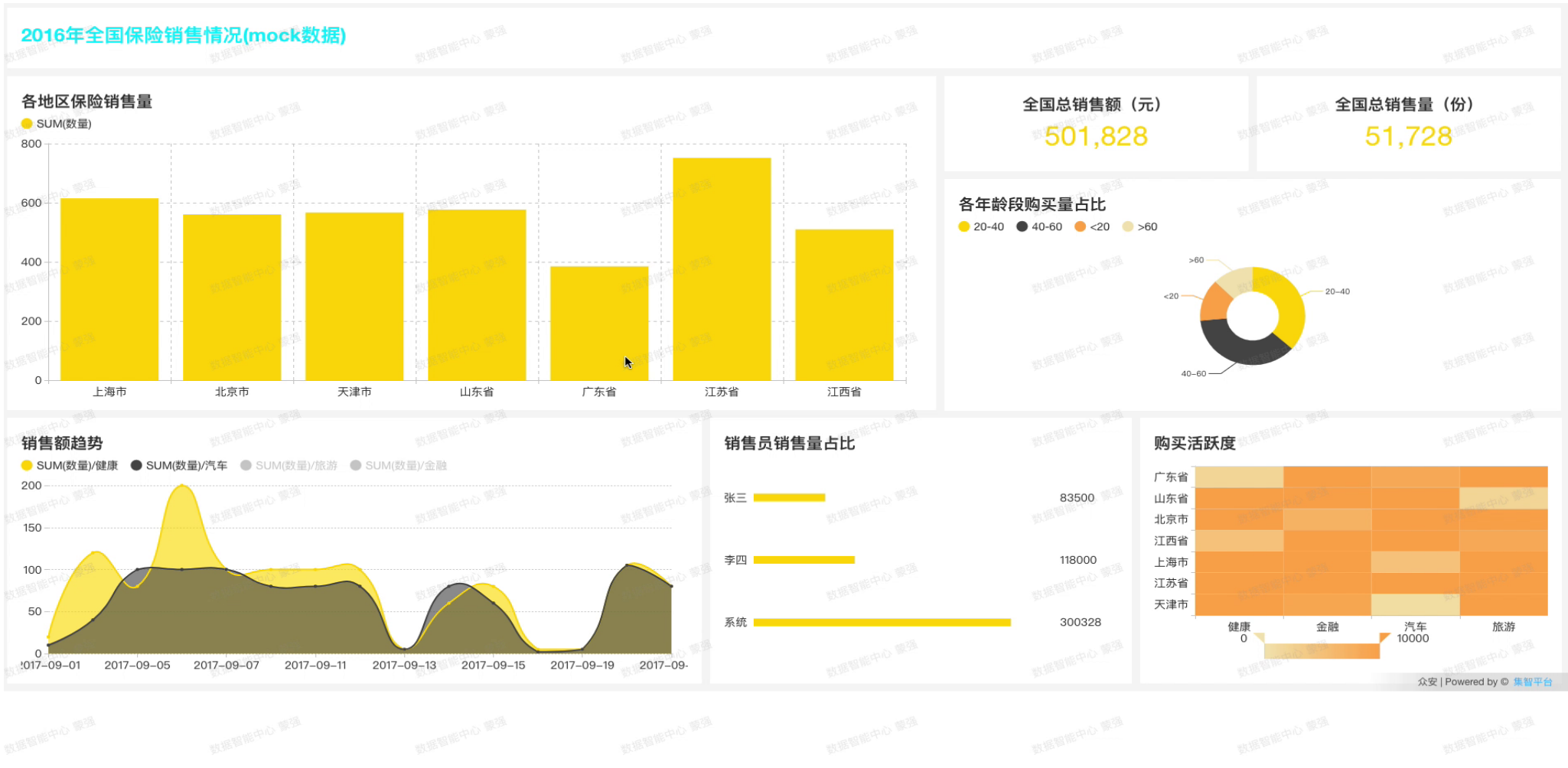
数据分析的最直观表现形式：报表

报表≠数据驱动

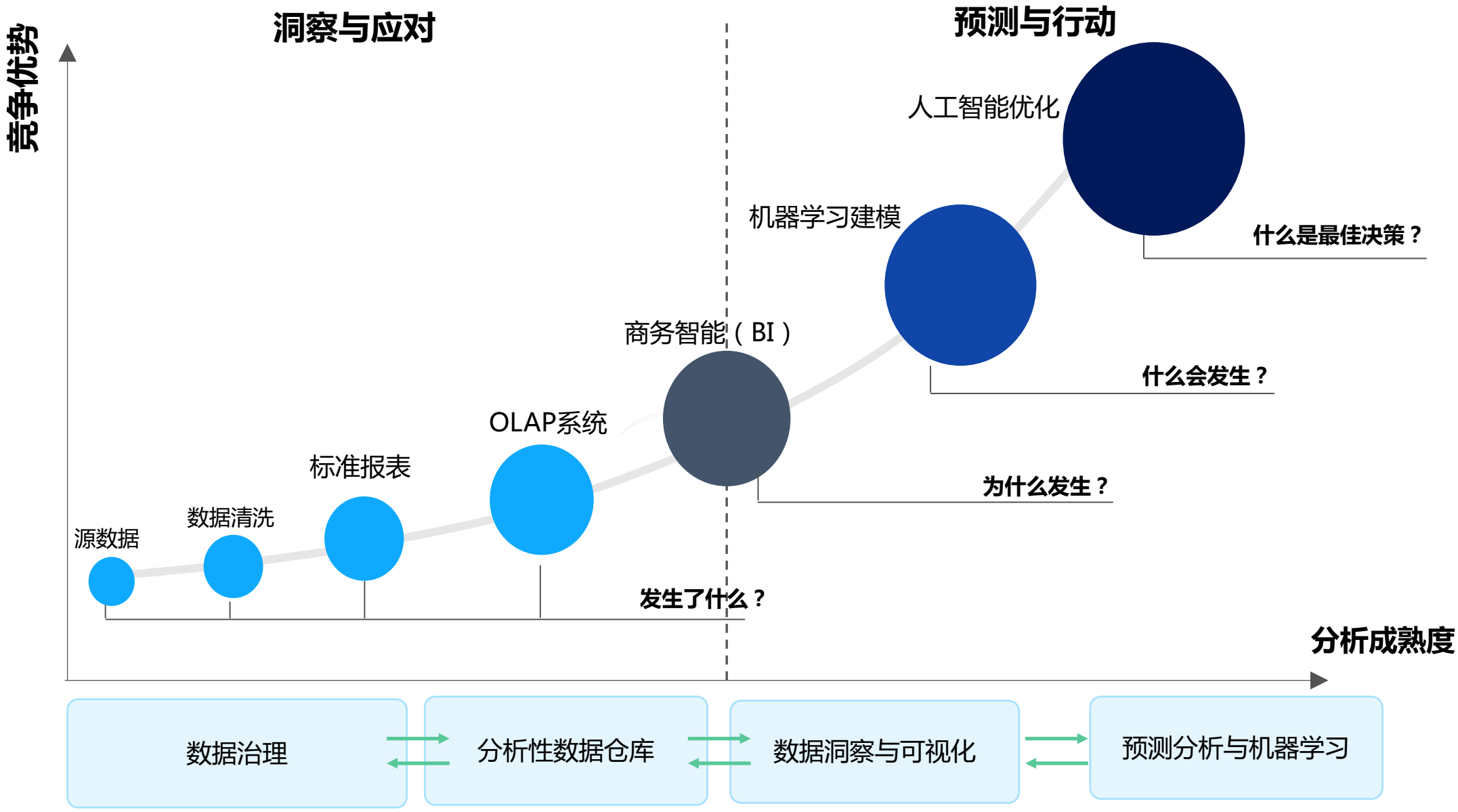
每天被访问超过10次的报表寥寥无几

传统报表访问往往是静态的、高聚合、低频、表单式的

集智平台可视化交互分析



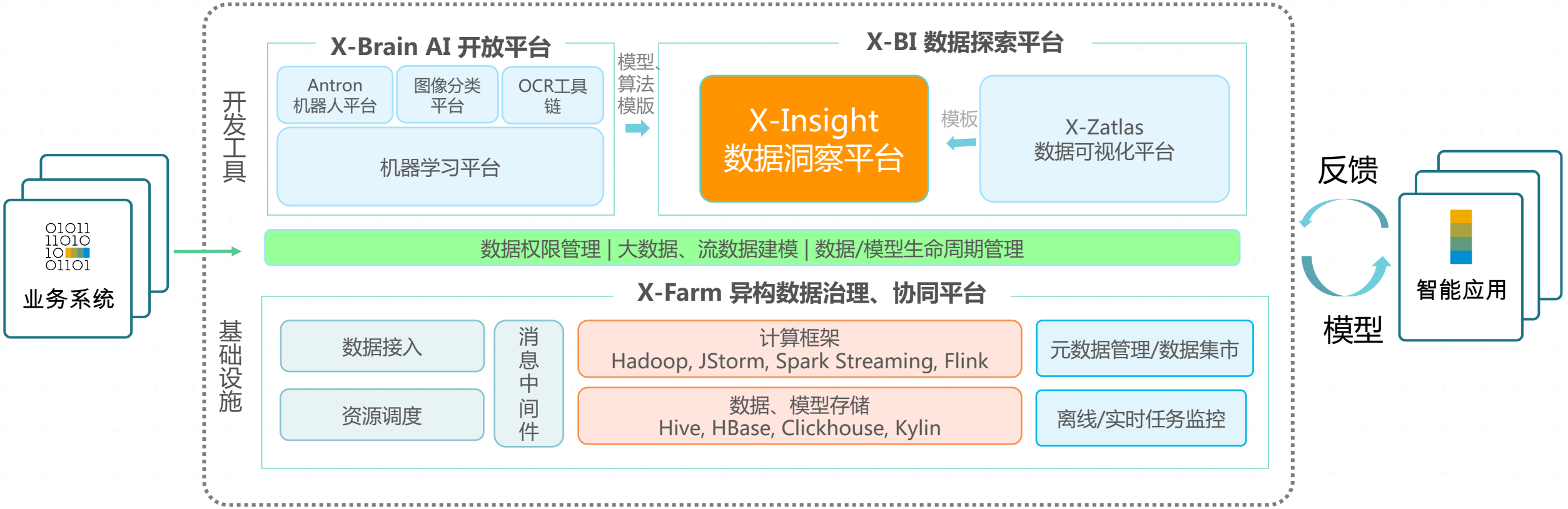
数据加工的链路与数据价值发现



CHAPTER 02

众安集智平台与clickhouse

集智平台



开放与敏捷

- 大数据、流数据统一建模管理
- 垂直方向行业模板，简化开发过程
- 多语言多runtime支持，Bring your own model

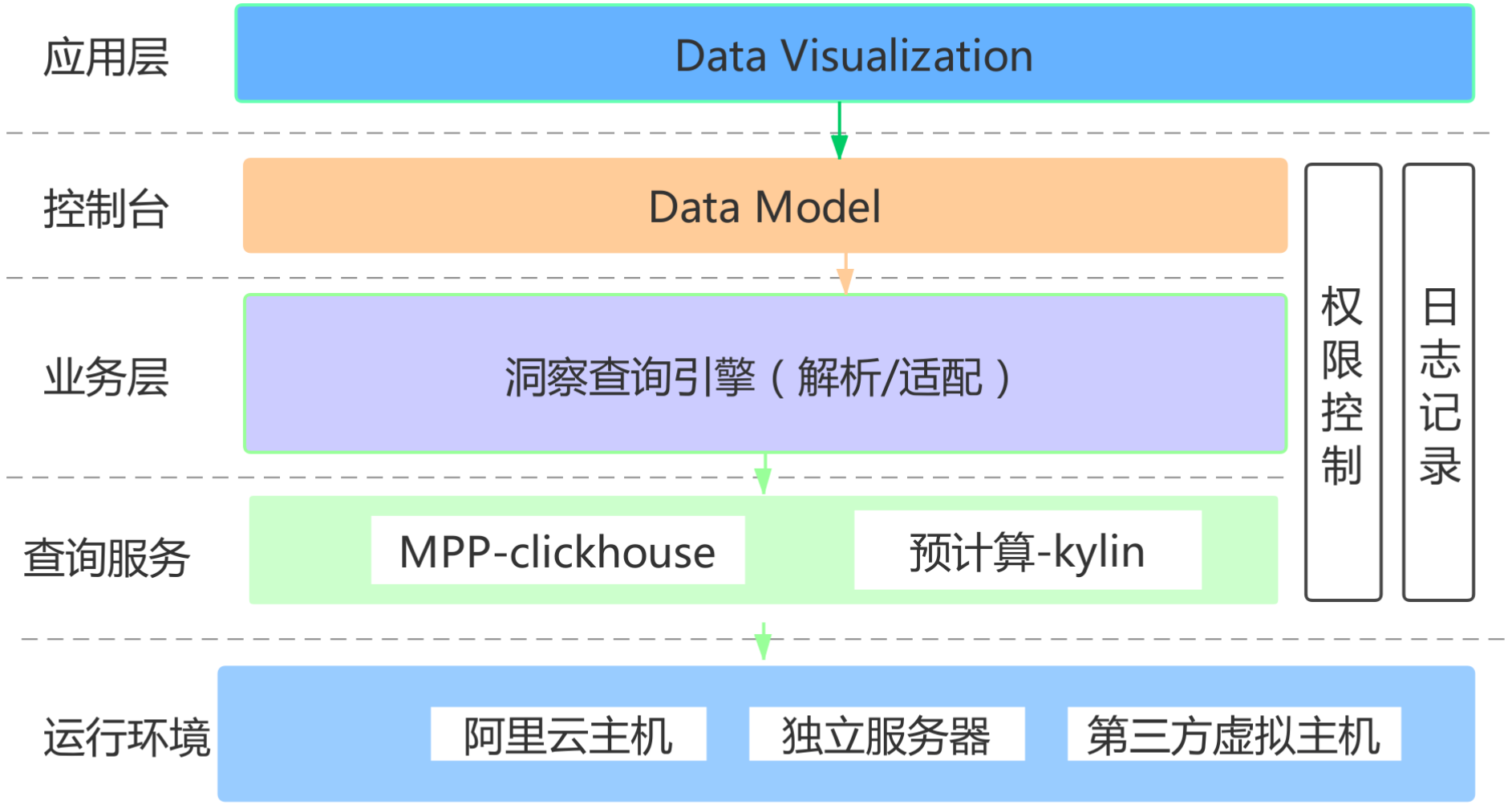
全生命周期管理

- 数据流转、建模、机器学习任务的全生命周期管理
- 大规模在线任务监控、自动模型性能监测、重训练与发布

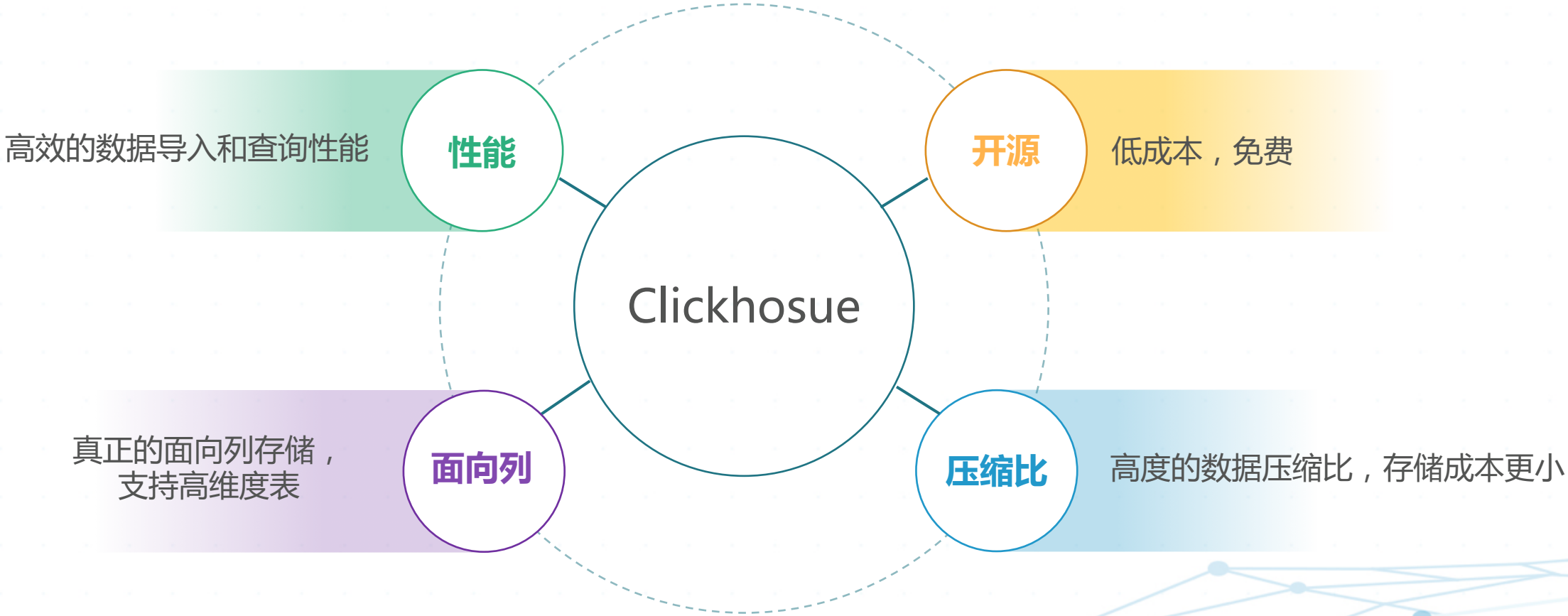
追溯与可重现

- 追溯数据血缘，数据、算法模型版本管理
- 支持算法模型结果的可重现、可审计
- 缓解AI/机器学习带来的潜在伦理与法律担忧

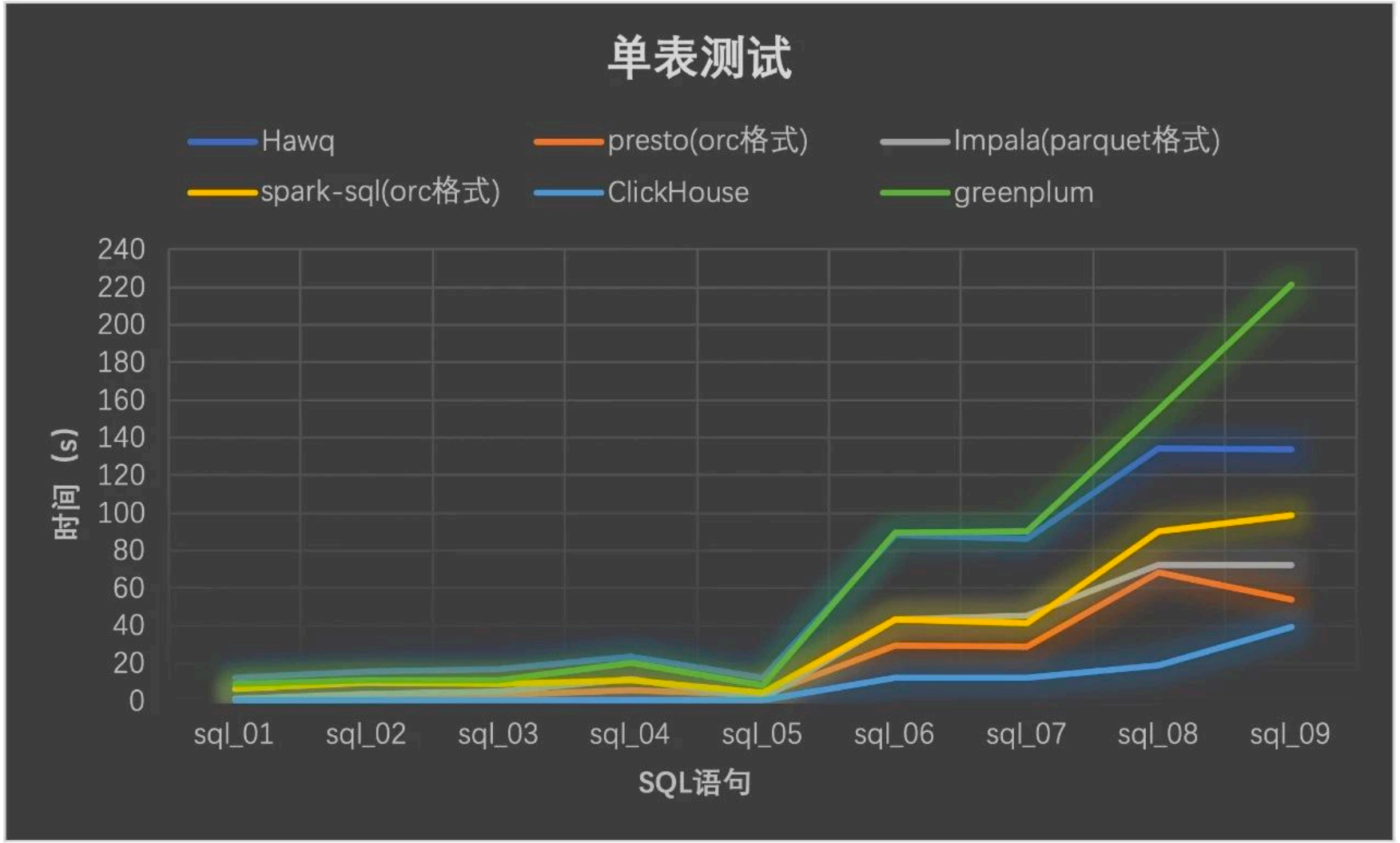
洞察平台架构


应用层
Data Visualization
控制台
Data Model
业务层
洞察查询引擎（解析/适配）
查询服务
MPP-clickhouse
预计算-kylin
运行环境
阿里云主机
独立服务器
第三方虚拟主机
权限控制
日志记录

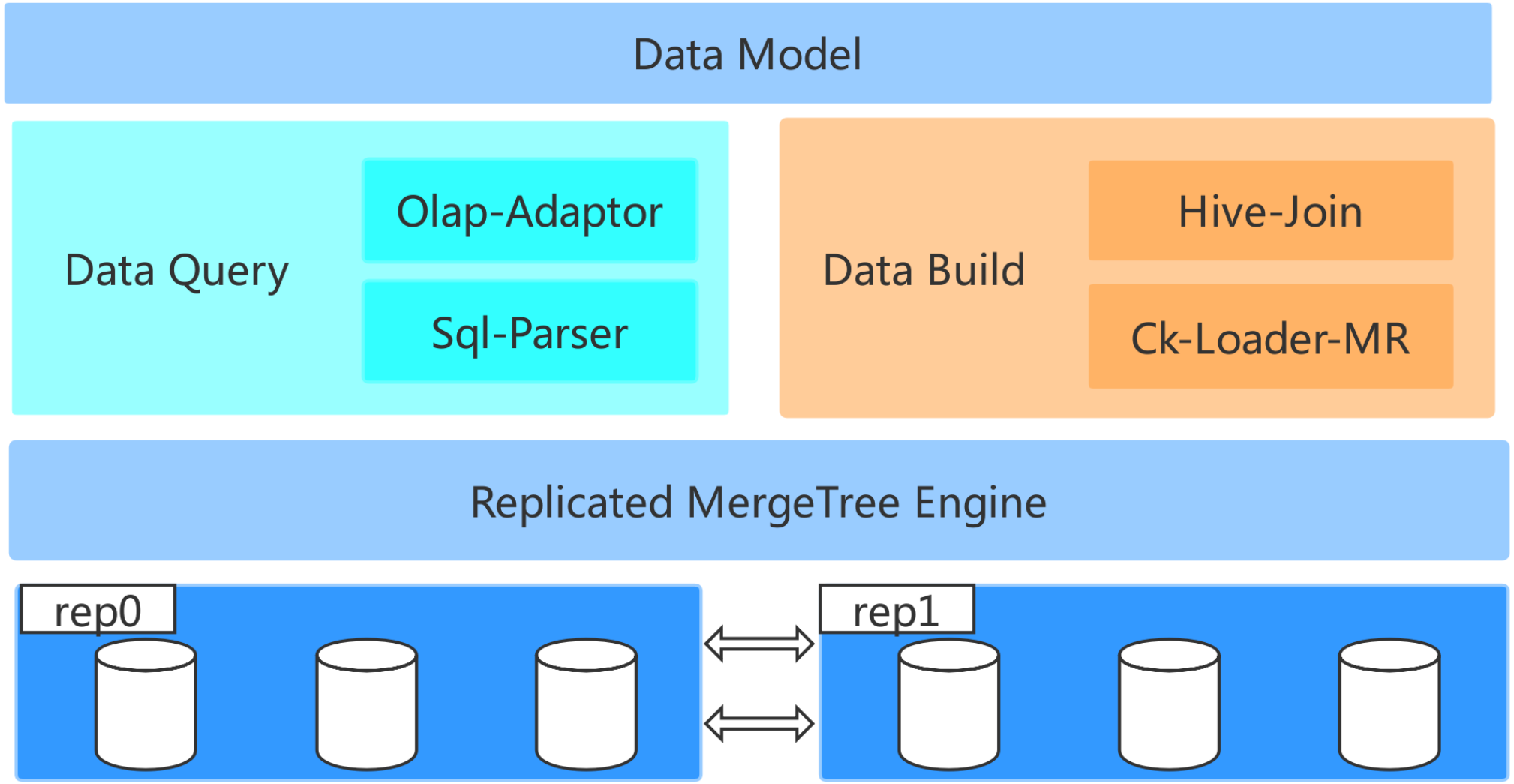
Why Clickhouse?



易观开源OLAP引擎测评报告



洞察数据模型+Clickhouse



使用效果

洞察后台数据查询分析（筛除洞察团队人员）

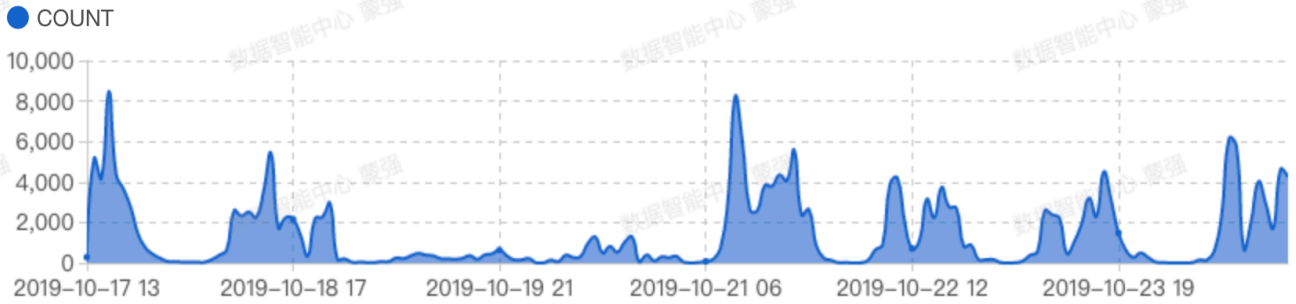
近7日 | 分 | 时 | 天 | 周 | 月 | 年 | 模型筛选 请选择模型 | 用户筛选 请选择用户

查询请求数	查询平均响应	查询用户数
229,392	311.03 ms	403

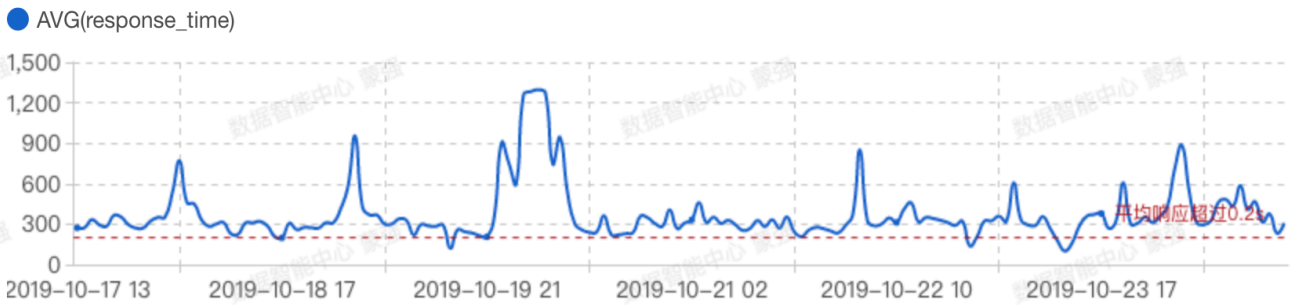
查询请求详情

数据实例	请求次数	平均响应时间
车点通哨兵下沉1-2-3模型	54,627	294.03
零售金融月报_summary	27,556	288.07
直营实时注册数据v2	13,778	284.23
直营实时激活数据	12,856	244.57
dashboard_5-_copy	11,739	349.56
直营实时注册数据	8,819	248.66
直营实时首登数据	6,203	251.64
cdm_pfm_policy_basic_dd_whole_insi	6,033	334.47
直营实时首登数据_copy	5,822	274.22
放款月零售盈利测算	4,923	283.63
cdm_zy_kpi_acitive_insight_dd	4,715	960.66
车险哨兵续保模型new	4,607	286.64

查询请求数



平均响应时长

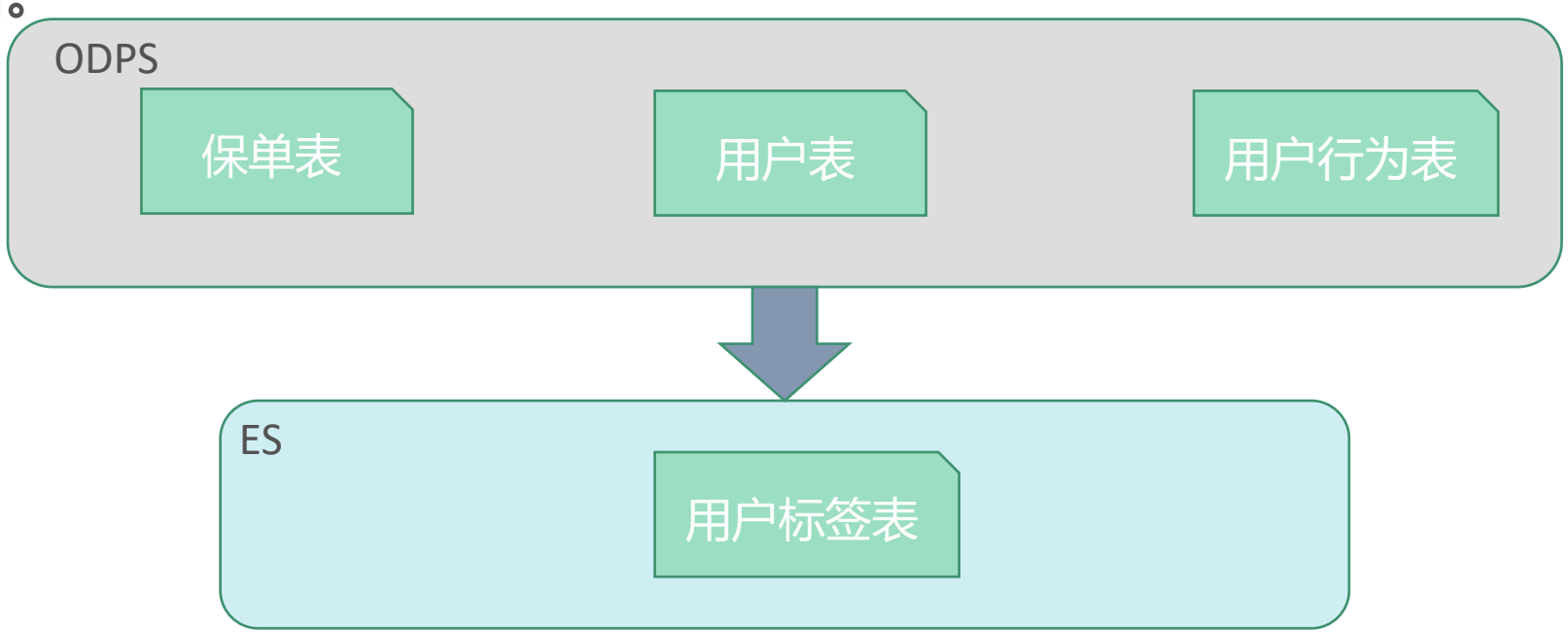


CHAPTER 03

使用ck对百亿数据的探索

背景

我们希望对保单、用户数据进行灵活分析，根据用户标签筛选出符合要求的客户进行精准营销。
原始保单数据百亿+，用户数据数亿，如果用户标签几百个，数据存储和查询以及分析的压力就会很大，原有系统使用es来保存用户标签数据。

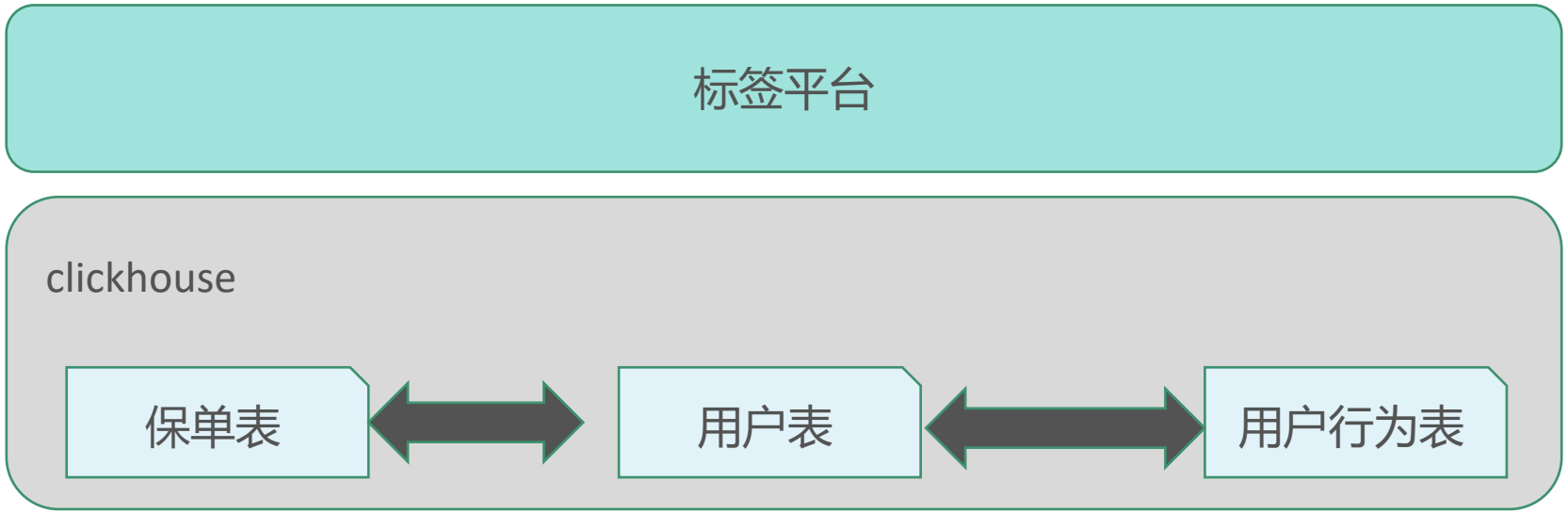


痛点

- 数据查询慢：每个查询需要5 ~ 10分钟；
- 数据更新慢：更新数据可能需要数天时间；
- 不灵活：用户有新标签需求时，需要提需求给标签开发人员排期开发需求，开发人员开发完再更新到系统中，这时离需求提出可能已经过去几天，无法及时给到业务人员反馈。

思路

利用clickhouse实时计算的高效性能，对原始数据进行查询分析，从而支持用户灵活的定义标签并让用户实时得到反馈。



数据

- 历史保单数据 join 用户数据 join 用户行为数据
- 100+亿行，50+列
  - 用户id
  - 事业部
  - 入库时间
  - first_policy_premium
  - ...
  - phone_flag
  - ha_flag
  - ...

clickhouse集群配置

- 阿里云ECS * 6，生产环境集群
  - CPU:
    - Intel(R) Xeon(R) CPU E5-2682 v4 @ 2.50GH
    - 12 cores 24 processors
  - 内存: 96GB
  - 硬盘: 1TB 高效云盘，最大IO吞吐量 140MBps

以事业部、入库时间作双分区导入数据

遇到的问题

导入效率：

- 原有导入数据方式在百亿级数据下会报Too many partitions for single INSERT block的问题
- 数据导入慢

原因：

- ck-loader-mr方式对大数据量场景支持不够友好
- 单次插入分区过多

解决方法：

使用clickhouse原生insert format csv 配合linux pipline导入

hadoop fs -cat 'hdfs://hadoop-namenode:port/user/hive/user/2013/000000_0' | clickhouse-client --host=127.0.0.1 --port=10000 -u user --password password --query="INSERT INTO Insight_zhongan.baodan_yonghushuju FORMAT CSV"

效果：

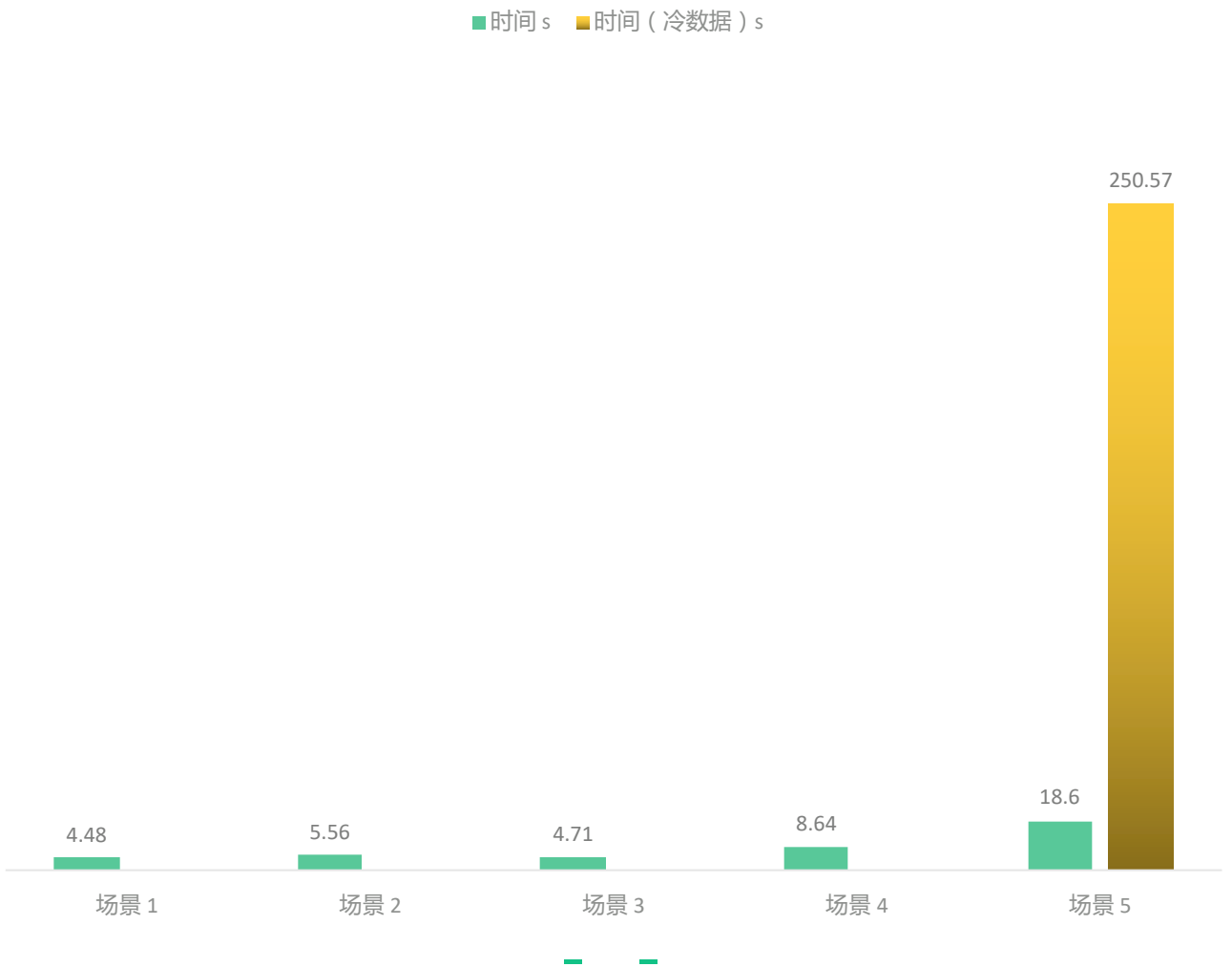
单进程：每分钟2600w条记录，client占用核数=1，server占用核数=1，导入速率=80mb/s

2进程：每分钟4000w条记录，client占用核数=2，server占用核数约2-5，导入速率=140mb/s

4进程: 每分钟8000w条记录，每个client占核数=1，server占用核约2-5，导入速率=280mb/s

ClickHouse 百亿数据性能测试与优化

- **数据查询**



一些典型查询的性能

测试1：手机号非空&健康险365天保费>100的用户车险总保费分布情况

```
[za-data-bigdata0023] 2019.08.08 13:49:19.860760 {90d48ca5-e892-424f-982d-7f7de0d5ae63} [ 706 ] <Information> executeQuery: Read 132031922 rows, 8.
61 GiB in 4.479 sec., 29481217 rows/sec., 1.92 GiB/sec.
[za-data-bigdata0023] 2019.08.08 13:49:19.860807 {90d48ca5-e892-424f-982d-7f7de0d5ae63} [ 706 ] <Debug> MemoryTracker: Peak memory usage (for query
): 1.77 GiB.
Ok.

0 rows in set. Elapsed: 4.480 sec. Processed 132.03 million rows, 9.24 GB (29.47 million rows/s., 2.06 GB/s.)
```

Elapsed	Processed rows	Throughput	Peak memory
4.480s	132.03 million, 9.24GB	29.47millon/s, 2.08GB/s	1.77GiB

测试2：健康险365天保费>100的用户前一年保费分布情况

```
[za-data-bigdata0023] 2019.08.08 13:45:51.751702 {05ac9a89-6d30-4815-833f-50d702f36e59} [ 706 ] <Information> executeQuery: Read 215262768 rows,
 15.70 GiB in 5.565 sec., 38679419 rows/sec., 2.82 GiB/sec.
[za-data-bigdata0023] 2019.08.08 13:45:51.751746 {05ac9a89-6d30-4815-833f-50d702f36e59} [ 706 ] <Debug> MemoryTracker: Peak memory usage (for qu
ery): 2.46 GiB.

3 rows in set. Elapsed: 5.567 sec. Processed 215.26 million rows, 16.86 GB (38.67 million rows/s., 3.03 GB/s.)
```

Elapsed	Processed rows	Throughput	Peak memory
5.567s	215.26 million, 16.86GB	38.67millon/s, 3.03GB/s	2.46GiB

ClickHouse 百亿数据性能测试与优化

- **场景5涉及到全表百亿行数据，第一次执行与后续执行花费时间差距较大**
  - 第一次执行，数据在硬盘上

    花费~250s，性能瓶颈在硬盘io (iostat util 100%)

  - 第二次执行，大部分数据已经在内存里

    花费~18s，性能瓶颈在cpu (top cpu usage ~1447%)

  - 两次运行的比较：

Metric	First run	Second run
top %CPU	~116%	~1447%
Peak Memory	1.84GiB	1.91GiB
iostat %util	100%	0.0%

Metric	First run	Second run
Elapsed	~250s	~18s
ReadBytes	4.2GiB	~0GiB
IOWait	>205.084s	0.001s

ClickHouse 百亿数据性能测试与优化

- **性能瓶颈在硬盘io，实验验证**
  - 数据分布在三台服务器上
    - 执行涉及到全表数据的查询（cold data，从硬盘读取），处理速度为~24.28million rows/s
    - 只用到三块硬盘的io：3*140=420mb/s
  - 数据分布在六台服务器上
    - 执行涉及到全表数据的查询（cold data，从硬盘读取），处理速度为~43.60million rows/s
    - 用到六块硬盘的io：6*140=840mb/s
  - io吞吐量加倍时，对于冷数据的处理速度是之前的~180%

ClickHouse 百亿数据性能测试与优化

- **硬盘存储升级**
  - 高效云盘 --> SSD + RAID0
  - 140MBps --> ~600MBps, ~4x
- **升级后**
  - ~250s --> ~69s，~3.62x

- **数据加热后**

~69s -- > 18s ，~3.8x

- **ToDos**
  - 优化数据导入流程
  - 支持多分区，支持指定主键
  - 常用字段加热

常用分析性能的命令分享

- **linux命令**
  - **top：查看系统cpu使用率，内存使用率等**
  - **iotop：查看系统进程占用io情况**
  - **iostat -dmx 1: 查看磁盘io使用情况，每秒更新**

- **Clickhouse命令：**
  - set send_logs_level = 'trace'：查看sql执行步骤详情
  - 根据query_id查看内存使用情况，io情况等详细信息：

```
system flush logs;
select ProfileEvents.Names as name, match(name, 'Bytes|Chars') ?
formatReadableSize(ProfileEvents.Values) : toString(ProfileEvents.Values) as value
from system.query_log array join ProfileEvents where event_date = today() and type
= 2 and query_id = '05ff4e7d-2b8c-4c41-b03d-094f9d8b02f2';
```

Thanks !