GOPS 2022
Shanghai

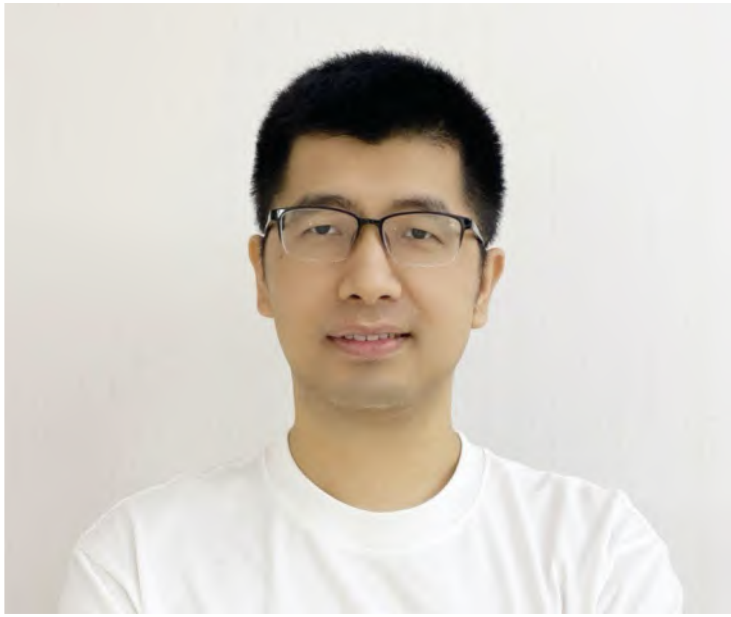
eBay基于ClickHouse事件监控平台建设

李先成　资深软件开发专家

李先成

资深软件开发专家

eBay软件开发工程师，多年专注于云计算和监控平台开发，现负责eBay事件和日志监控平台的建设。

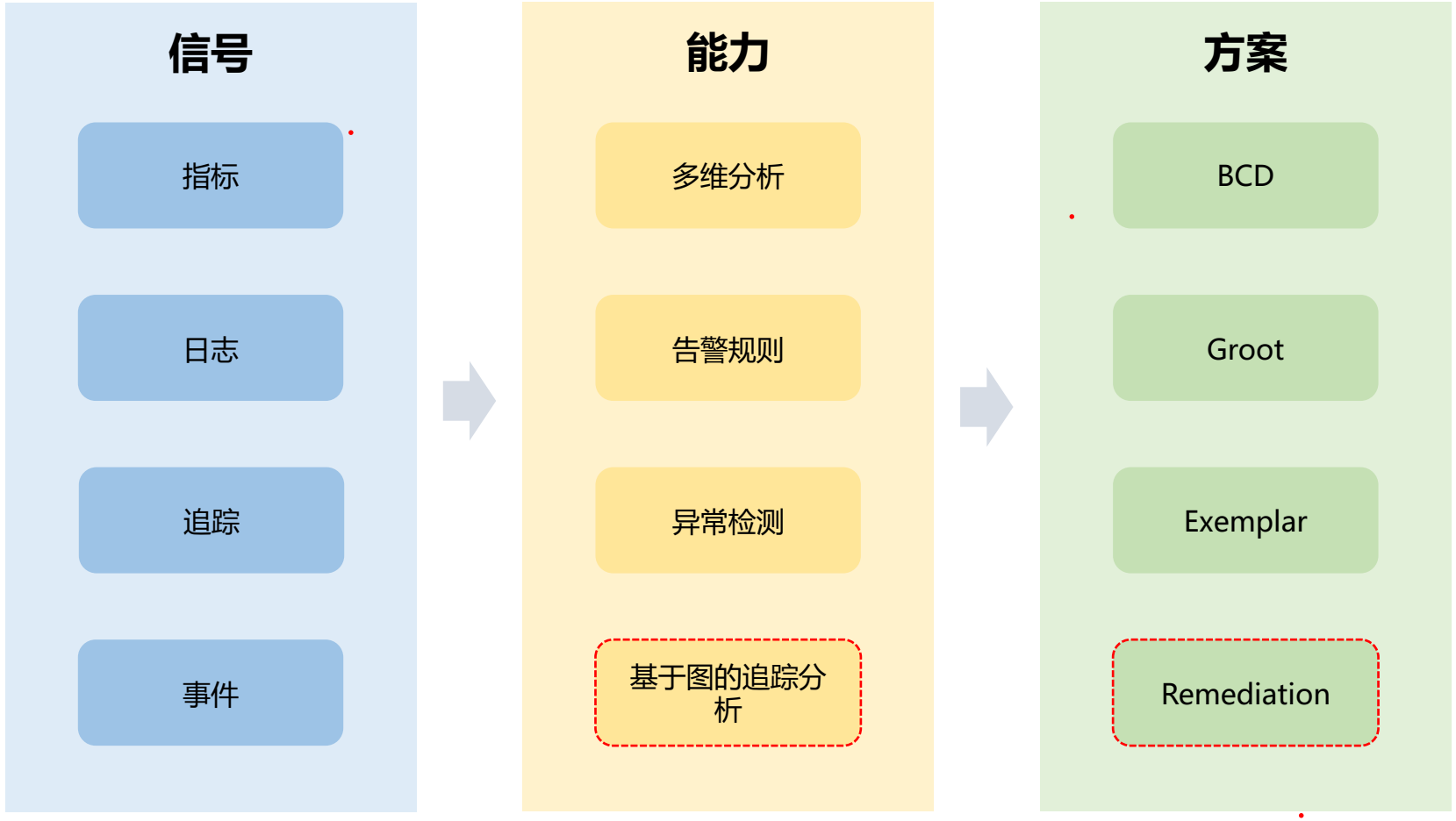


目录
contents

Part 01

背景介绍

监控平台


信号
指标
日志
追踪
事件
能力
多维分析
告警规则
异常检测
基于图的追踪分析
方案
BCD
Groot
Exemplar
Remediation

什么是事件？

1. 用户事件
   - 部署
   - 扩容
   - 配置
2. 系统事件
   - 告警
   - 错误和异常
   - 崩溃和重启
   - 访问日志
   - 审计日志

稀疏

往往不是周期性的

结构化

包含任意一组键值对

聚合

高效多维分析

告警

告警规则和异常检测

Part 02

事件平台

平台现状

2000亿+

日写入量

400+

ClickHouse节点数量

5百万+

日查询量

1PB+

存储容量

技术选型 (ClickHouse)

性能

- 向量化执行(SIMD指令)
- 运行时代码生成(JIT)
- 纵向多核并行计算
- 横向多切片分布式计算
- 数据有序存储
- 主键索引(稀疏索引)
- 跳数索引
- 投影

成本

- 列存储
- 列编码和压缩
- 分层存储
- 支持S3和HDFS的零拷贝复制

功能

- 多样化MergeTree表引擎
- 物化视图和丰富聚合功能
- 数据同步
- 数据分片
- 灵活的集群管理
- 丰富的数据类型
- 物化列
- 身份认证
- 配额
- 审计日志
- SQL

哪些是降本增效的关键呢？

为什么ClickHouse可以降本增效？

按列存储

- 相似的数据存储在一起
- LowCardinality、DoubleDelta等编码
- LZ4, ZSTD等压缩算法

向量化执行

- 列式计算数据在内存中式连续的
- 算子的向量化
- 提高CPU缓存的命中率
- 减少虚函数的调用
- 降低分支预测判断失败的概率
- 更好的指令流水
- 充分利用CPU SIMD指令的并发能力

运行时代码生成

- 对表达式计算和逻辑操作进行汇编级别的优化
- 避免大量虚函数调用
- 避免动态类型转换
- 提高L1、L2缓存的利用率
- 更好的利用目标CPU指令

column	type	compressed	uncompressed	compr_rate	rows_cnt
body	String	7.42 TiB	71.90 TiB	9.69	241.54 billion
resource.name.node	String	193.54 GiB	9.94 TiB	52.61	241.54 billion
resource	Map(String, String)	35.34 GiB	7.87 TiB	228.03	241.54 billion
resource.name.filename	String	10.81 GiB	2.38 TiB	225.35	241.54 billion
string.attribute	Map(String, String)	8.00 GiB	1.76 TiB	224.97	241.54 billion
resource.name.application	LowCardinality(String)	1.14 GiB	226.39 GiB	197.96	241.54 billion
resource.name.namespace	LowCardinality(String)	1.14 GiB	225.82 GiB	198.19	241.54 billion
resource.name.applicationinstance	LowCardinality(String)	1.13 GiB	225.83 GiB	199.36	241.54 billion
resource.name.applicationservice	LowCardinality(String)	1.13 GiB	225.82 GiB	199.4	241.54 billion
resource.name.cluster	LowCardinality(String)	1.07 GiB	225.39 GiB	211	241.54 billion
resource.name.colo	LowCardinality(String)	1.06 GiB	225.39 GiB	211.95	241.54 billion
resource.name.container	LowCardinality(String)	1.06 GiB	225.39 GiB	212.74	241.54 billion
resource.name.pod	String	1.00 GiB	224.97 GiB	224.35	241.54 billion

ClickHouse查询并行处理优化

- 纵向：分区+Granule拆分，单机多核并行
- 横向：多分片分布式计算机+单分片多副本并行化

平台设计

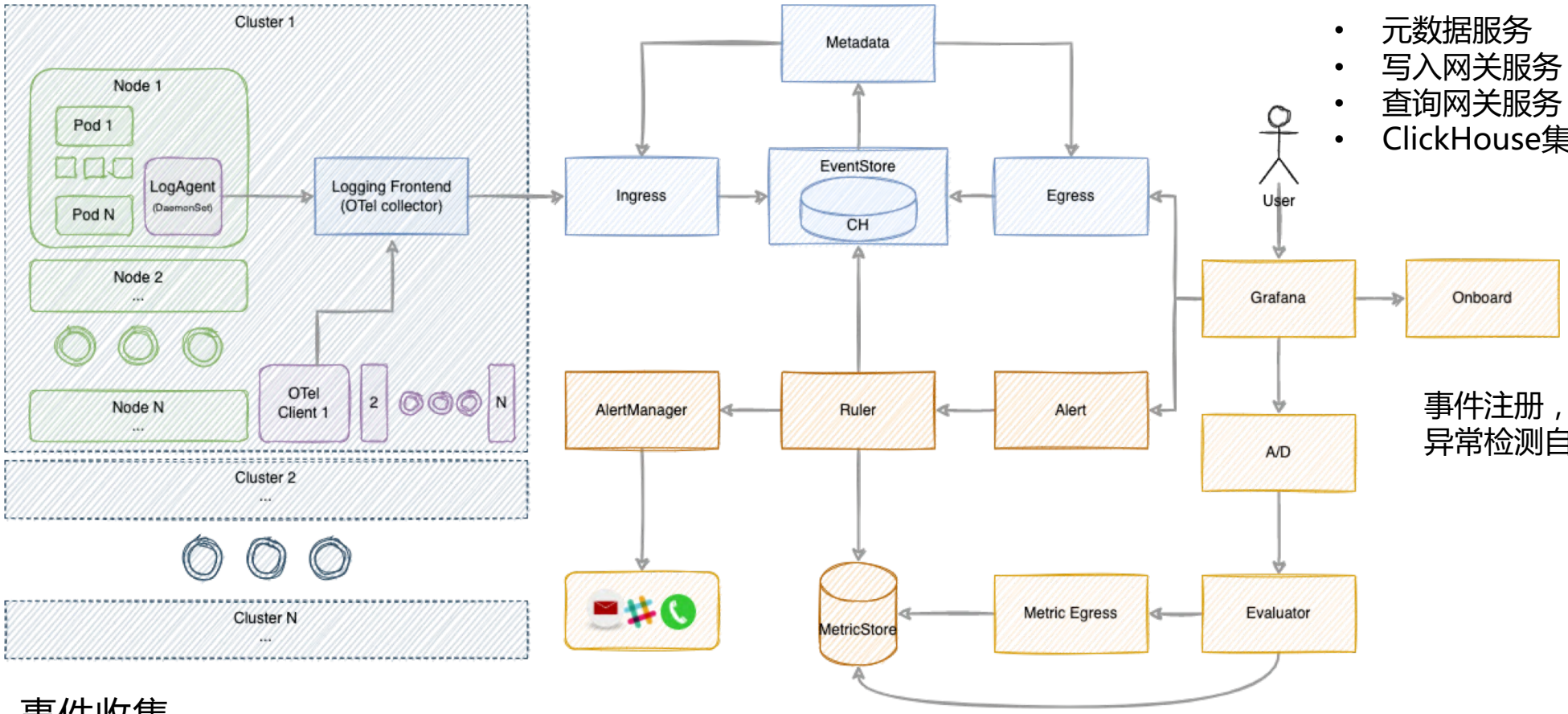
遵循一键化和自动化运维策略。

- 完全容器化部署，使用k8s进行编排，通过k8s CRD来实现统一的资源管理。
  - FCHI CRD管理跨区域ClickHouse集群
    - 一键扩缩容
    - 自动替换故障节点
  - 表格 (Schema CRD)
  - 配额 (ResourceQuota CRD)

同时遵循开源和标准化接口设计。

- 收集：采用CNCF Otel标准的数据模型、采集、处理和导出方案。
- 查询：除了SQL还实现了LogQL来查询事件，从而无缝对接第三方可视化平台Grafana和告警平台Prometheus。

架构设计



- 元数据服务
- 写入网关服务
- 查询网关服务
- ClickHouse集群

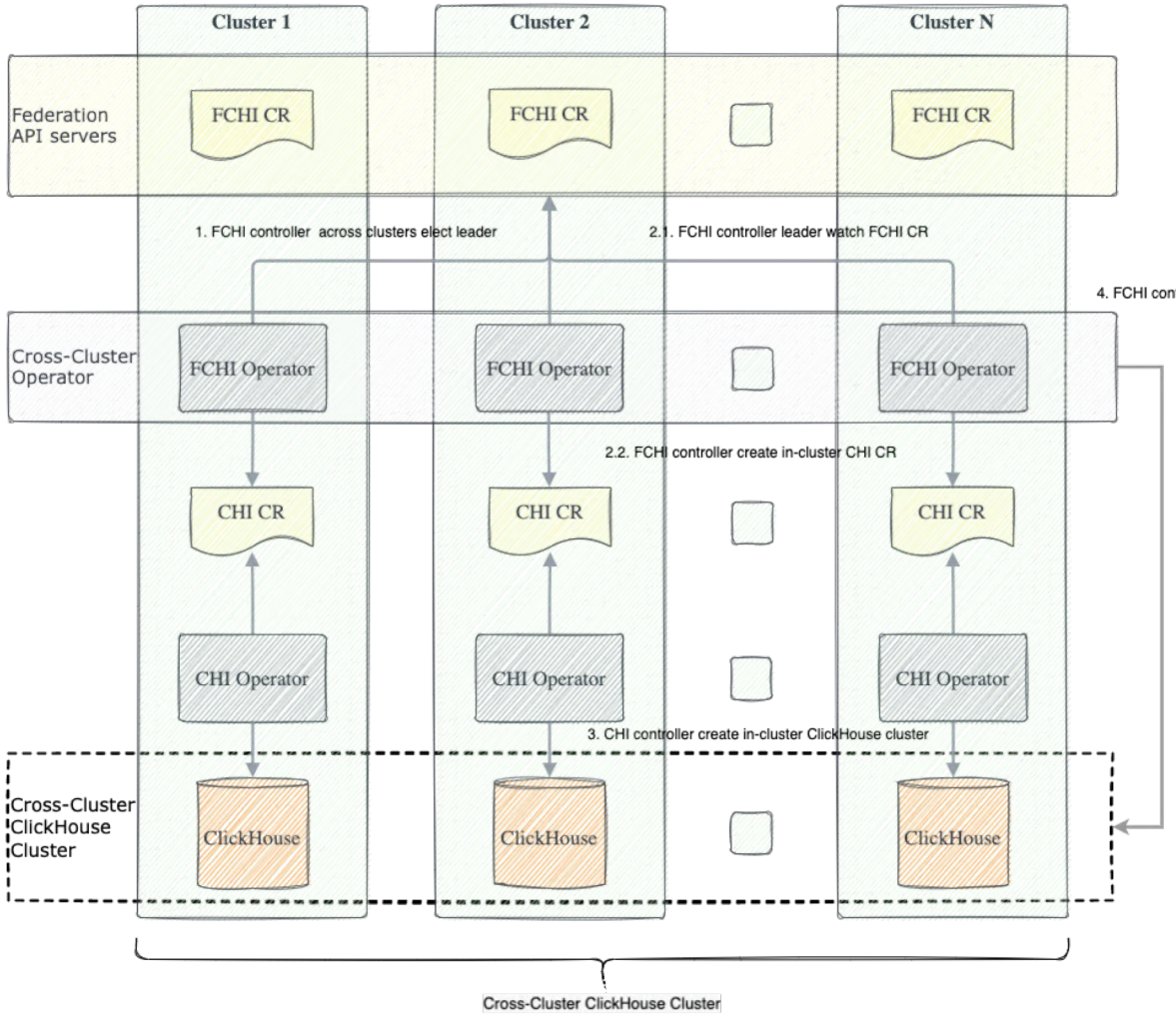
事件注册，告警规则和异常检测自服务平台

事件收集

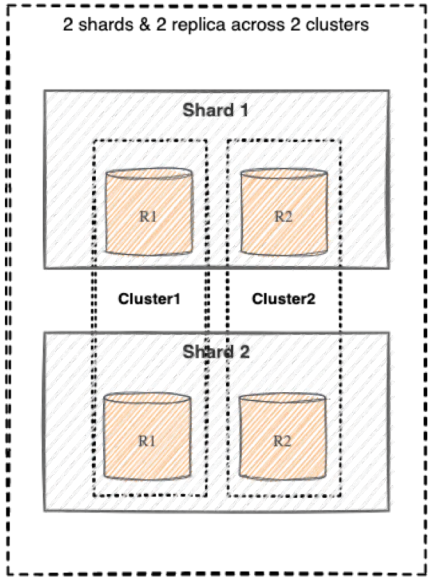
- LogAgent
  - 收集pod日志文件
  - 转化为结构化数据
  - 添加位置信息
  - 自定义（采样、预聚合等）
- OTel SDK

指标和事件共用的告警平台和异常检测平台

跨区域高可用ClickHouse集群



1. 引入CHI CRD描述单个k8s集群的ClickHouse集群
2. 引入FCHI CRD描述跨区域高可用ClickHouse集群
3. 引入Federation API servers描述跨区域部署

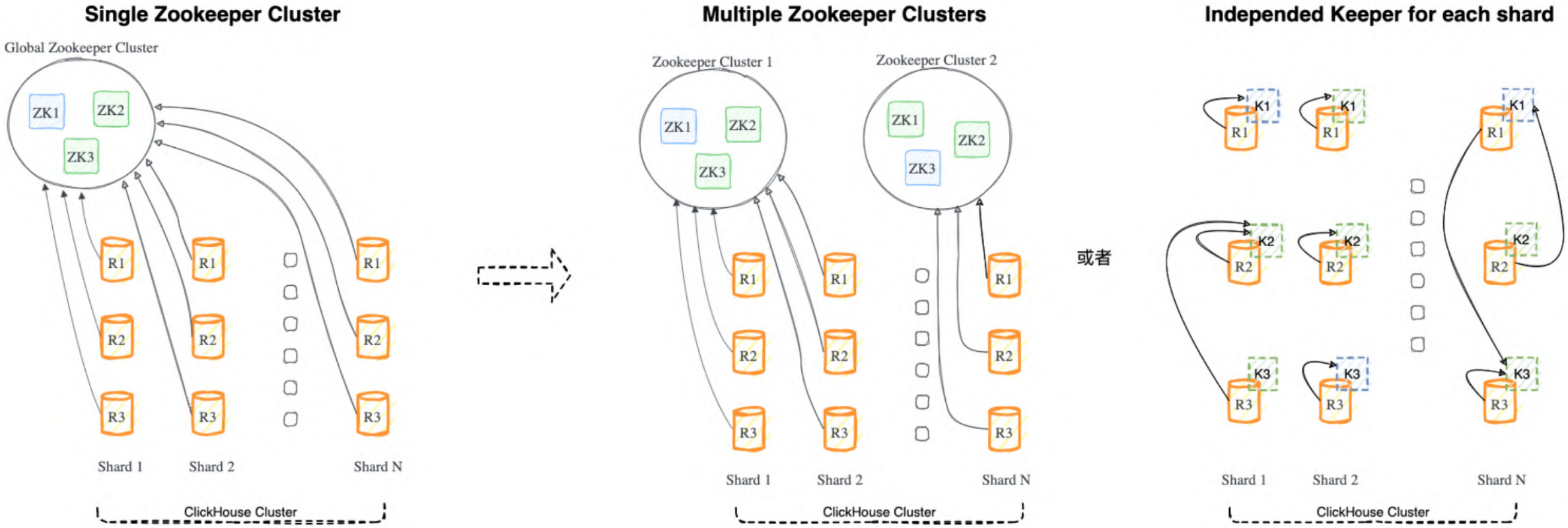



```
<yandex>
 <remote_servers>
  <zeus>
   <shard>
    <internal_replication>true</internal_replication>
    <replica>
     <host>S1R1.Cluster1</host><port>9000</port>
    </replica>
    <replica>
     <host>S1R2.Cluster2</host><port>9000</port>
    </replica>
   </shard>
   <shard>
    <internal_replication>true</internal_replication>
    <replica>
     <host>S2R1.Cluster1</host><port>9000</port>
    </replica>
    <replica>
     <host>S2R2.Cluster2</host><port>9000</port>
    </replica>
   </shard>
  </zeus>
 </remote_servers>
</yandex>
```

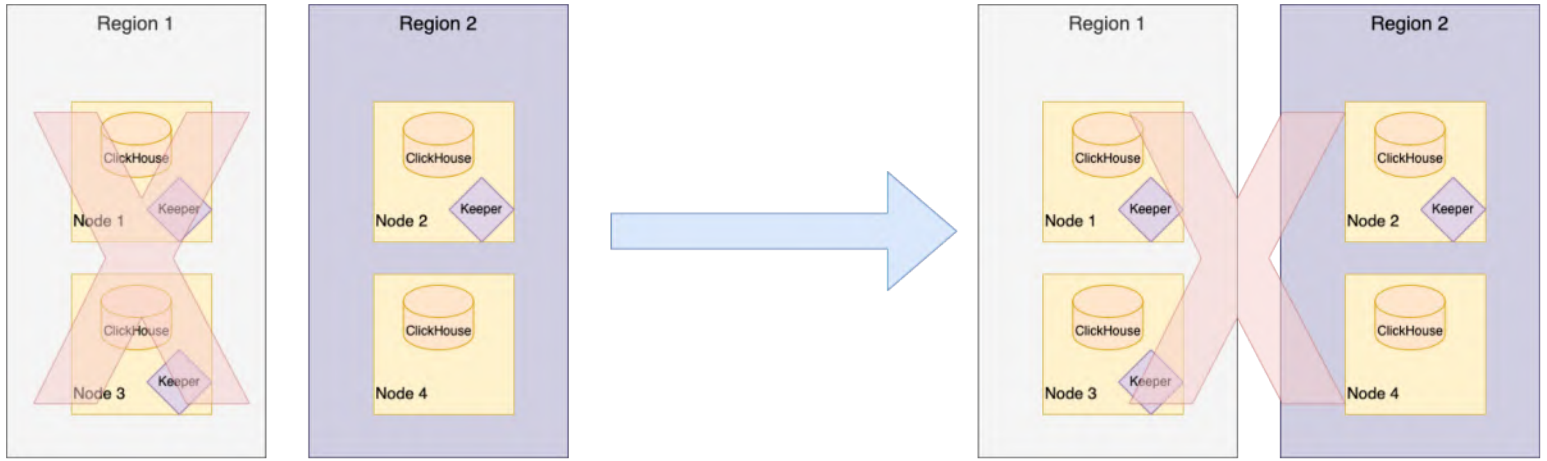
水平可扩展ClickHouse集群

利用FCHI CRD来实现横向和纵向扩缩容，但是Zookeeper（数据同步和分布式DDL）是ClickHouse集群水平扩展的瓶颈，主要解决方案有：

1. 增强FCHI CRD支持不同的切片指向不同的Zookeeper集群
2. 每个分片用内置CH Keeper集群（没有Zookeeper集群管理负担，还解决了ZXID rollover，ZooKeeper日志压缩等问题）

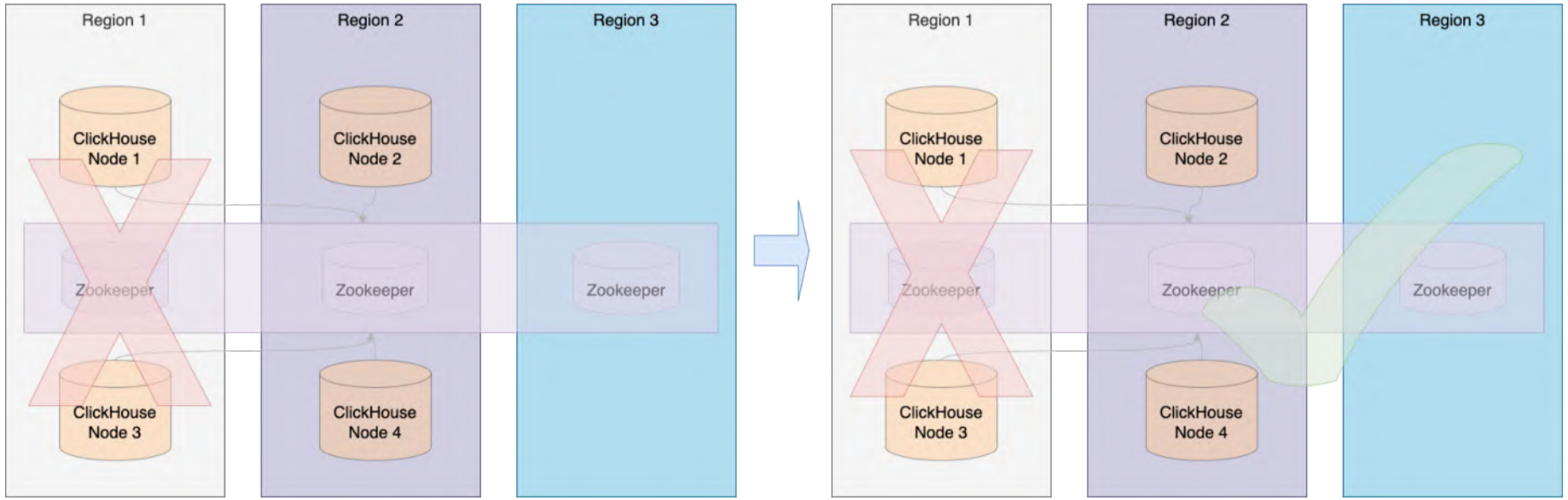


ClickHouse Keeper的问题

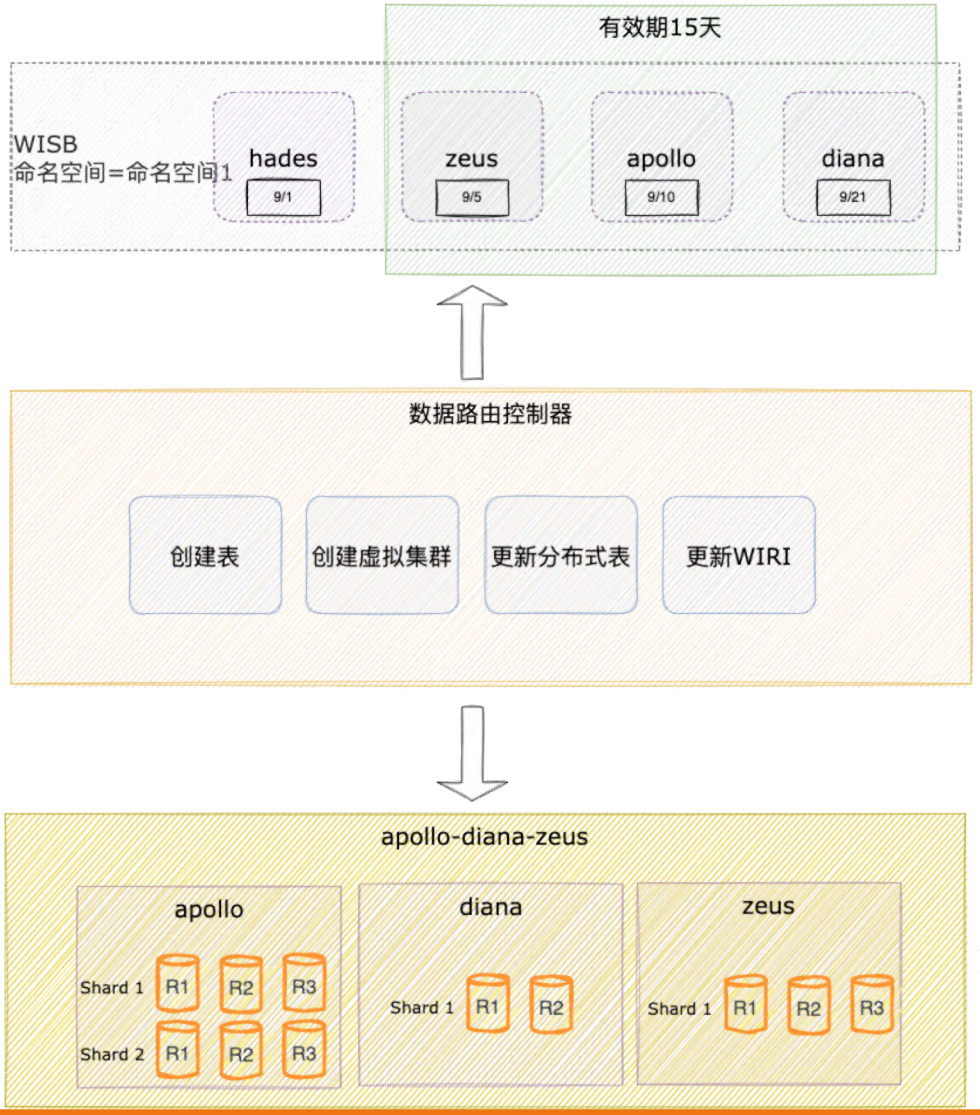


当副本数量是偶数时，为了让ClickHouse Keeper能够正常选主，就需要多区域异构部署，但是这样就无法实现高可用了，因为不能容忍异构部署的那个多副本区域不工作。

当多副本区域不可用的时候，ClickHouse Keeper存活的节点数量达不到半数无法选主，从而导致整个ClickHouse Keeper集群不可用，进而导致整个ClickHouse集群不可用。



数据路由和迁移

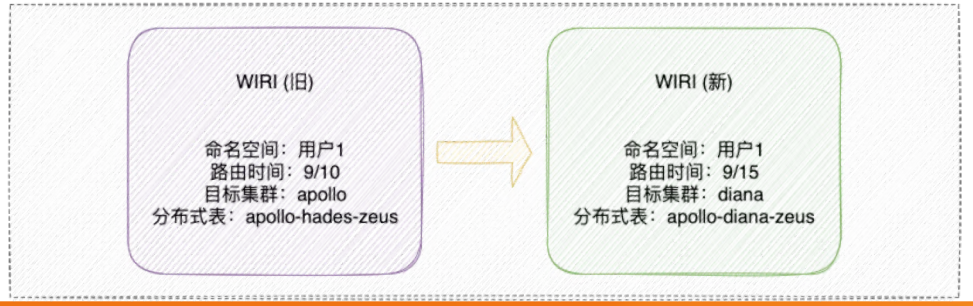


虽然我们的ClickHouse集群是可以水平可扩展的，不过从读写效率和数据隔离的角度，我们事件平台还是采用了多个ClickHouse集群的方案。

类似于k8s，我们通过命名空间来表示虚拟的ClickHouse资源，一个租户有一个或者多个命名空间，我们通过定义命名空间和ClickHouse集群的映射关系来定义数据路由。

具体来说，通过WISB和WIRI来定义数据路由：

1. WISB代表期待的数据路由
2. WIRI代表真实的数据路由
3. 数据路由控制器把WISB转变成WIRI来完成数据路由的调和
4. 新命名空间注册
5. 数据迁移：虚拟集群+分布式表实现无复制迁移
   - ClickHouse-Copier
   - Copy partition/parts
   - Insert … select from remote



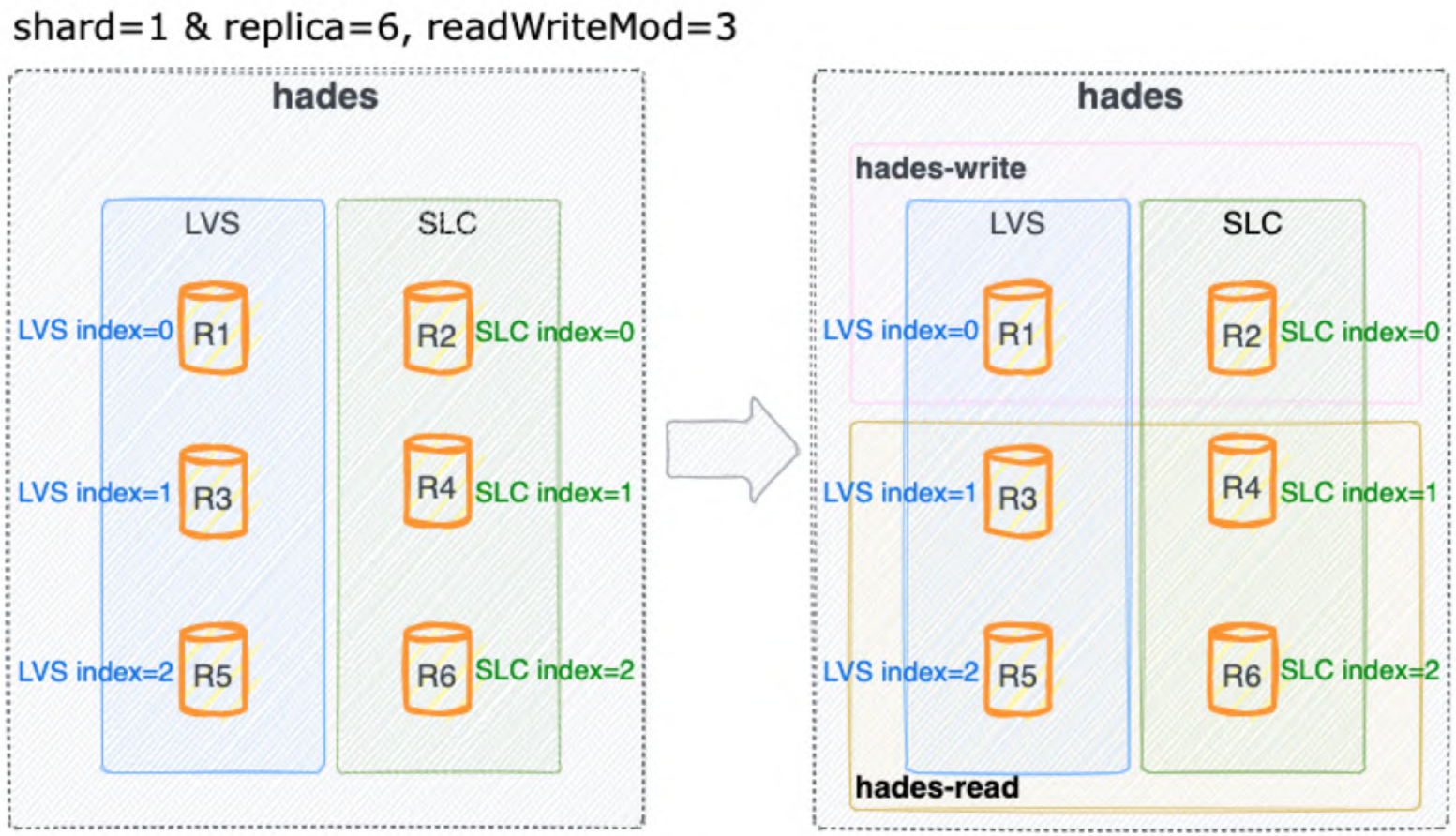
读写分离

有些事件对数据准确性要求很高，我们通过在FCHI CRD中引入readWriteMod实现读写分离的支持。

没有采用ClickHouse存算分离的方案主要是因为存算分离的方案需要使用远程存储，但是我们采用的是冷热分层的架构，而且热存储使用的是本地SSD磁盘。

- 通过readWriteMod>1启用读写分离
- 写入指向xxx-write CH集群 ({replica_num} % readWriteMod == 0)
- 分布式表指向xxx-read CH集群 ({replica_num} % readWriteMod != 0)

分配1个写节点和1个或者多个读节点，因为通常都是读的压力比较大。



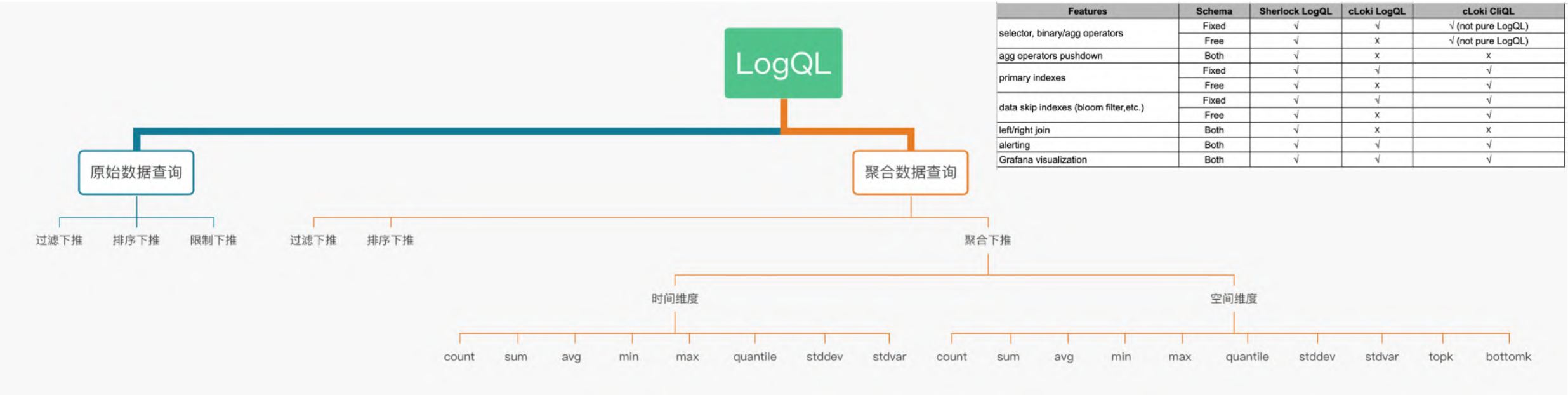
查询语言

除了SQL查询语言，为了提供类似于指标类Prometheus PromQL语法类似的DSL并兼容Prometheus告警平台以及我们的异常检测平台，我们开发了基于ClickHouse的LogQL的DSL。

包含：

1. 原始数据查询 (Logs Query)
2. 聚合数据查询 (Metrics Query)

为了提高查询效率，我们尽可能下推几乎所有LogQL的算子和聚合函数，并且还可以自动回退到LogQL引擎来运算不能下推的部分。

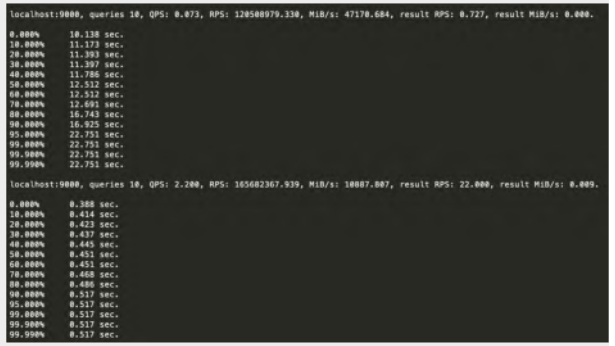


Features	Schema	Sherlock LogQL	cLoki LogQL	cLoki CliQL
selector, binary/agg operators	Fixed	√	√	√ (not pure LogQL)
	Free	√	x	√ (not pure LogQL)
agg operators pushdown	Both	√	x	x
primary indexes	Fixed	√	√	√
	Free	√	x	√
data skip indexes (bloom filter,etc.)	Fixed	√	√	√
	Free	√	x	√
left/right join	Both	√	x	x
alerting	Both	√	√	√
Grafana visualization	Both	√	√	√

进一步优化

{_namespace_=” appstats”}

简单的原始数据查询，选择最近6小时，按时间降序排序，返回前1000条记录。
ClickHouse SQL执行计划的问题




```
SELECT
    toInt32(unixtime) AS unixtime,
    application,
    colo,
    az,
    container,
    buildlabel,
    class,
    cluster,
    commandname,
    configlabel,
    count,
    duration
FROM sherlockio.appevent_v2_all AS table
WHERE (unixtime >= 1666744098) AND (unixtime < 1666747698)
ORDER BY unixtime DESC
LIMIT 1000
```



```
WITH (
        SELECT
            min(unixtime) AS min,
            max(unixtime) AS max
        FROM
        (
            SELECT unixtime
            FROM sherlockio.appevent_v2_all
            WHERE (unixtime >= 1666744098) AND (unixtime < 1666747698)
            ORDER BY unixtime DESC
            LIMIT 10
        )
    ) AS minmaxtime
SELECT
    toInt32(unixtime) AS unixtime,
    application,
    colo,
    az,
    container,
    buildlabel,
    class,
    cluster,
    commandname,
    configlabel,
    count,
    duration
FROM sherlockio.appevent_v2_all AS table
WHERE (unixtime >= (minmaxtime.1)) AND (unixtime < if((minmaxtime.1) = (minmaxtime.2),
(minmaxtime.1) + 1, minmaxtime.2))
ORDER BY unixtime DESC
LIMIT 1000
```

分析

生成PreWhere
根据Partition筛选Parts
根据索引筛选Granule

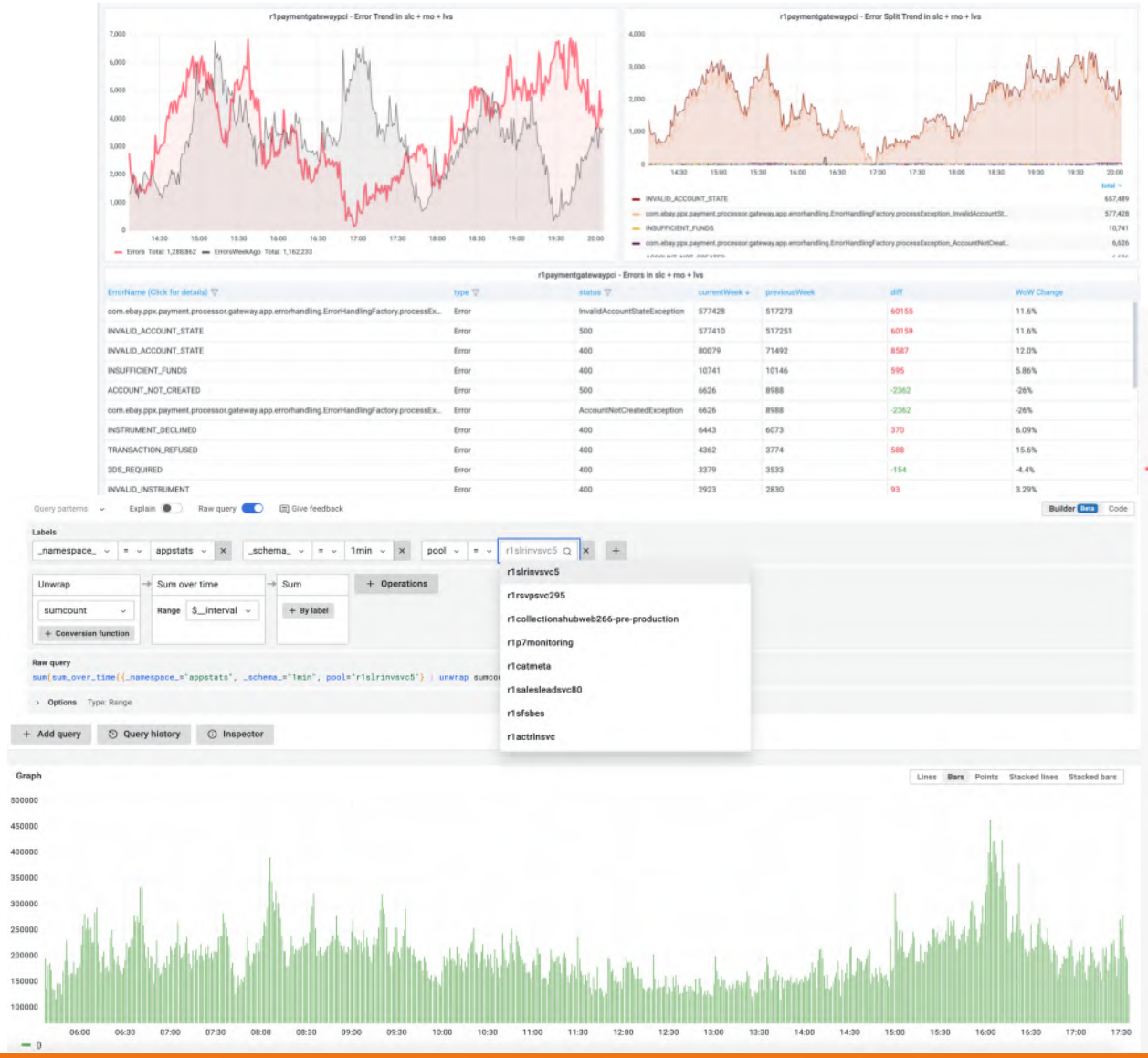
并行读取

并行化
读取PreWhere列并Filter
读取其他列

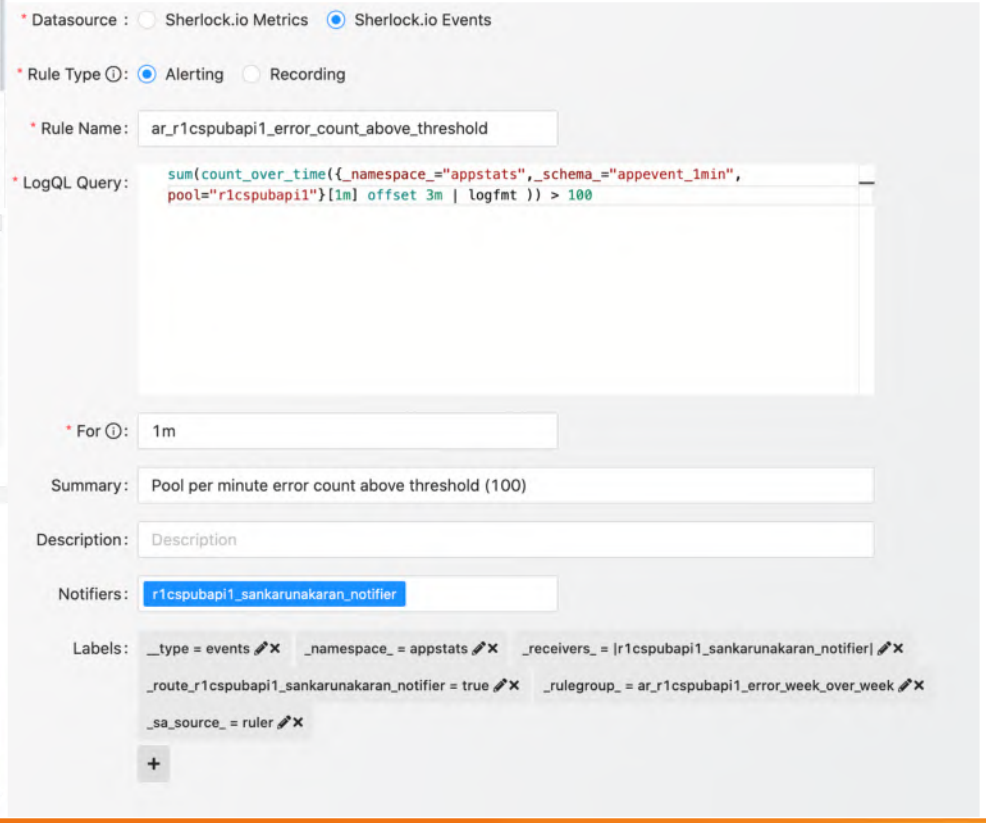
排序限制

根据OrderBy排序
应用Limit

视图和告警



用户可以通过SQL和LogQL来创建可视化视图，不过配置告警规则和设置异常检测目前支持LogQL。对PromQL和LogQL不熟悉的用户可以通过Grafana提供的查询生成器，通过简单的下拉菜单选择完成聚合查询，明细查询，和可视化视图的创建。



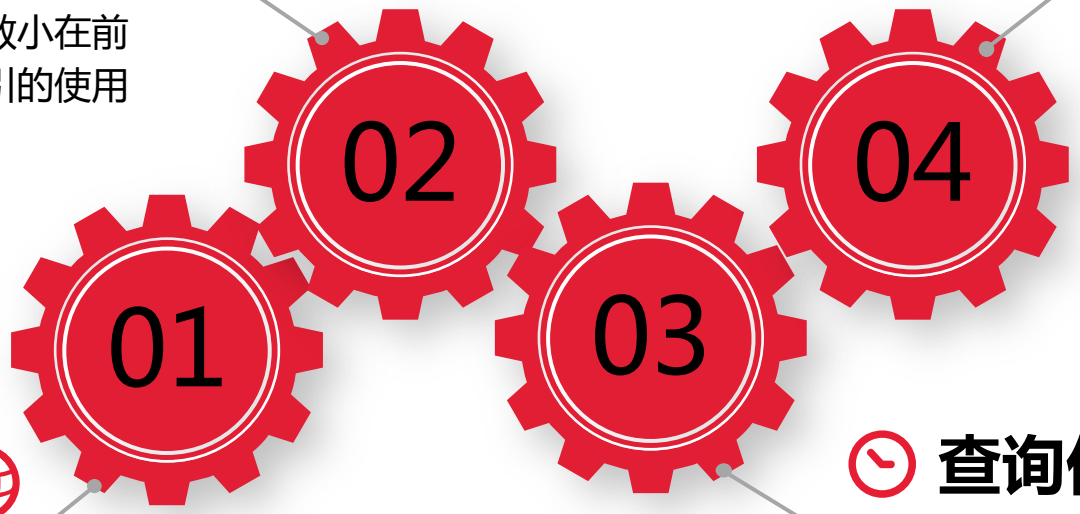
ClickHouse最佳实践

稀疏索引

复合索引按照查询频率大和基数小在前
注意跳数索引的使用

参数配置

最大并发，单个查询最大CPU个数、
最大内存、最大扫描/返回行数限制



写入优化

批量写入和复用网络连接
按照SortingKey排序后写入

查询优化

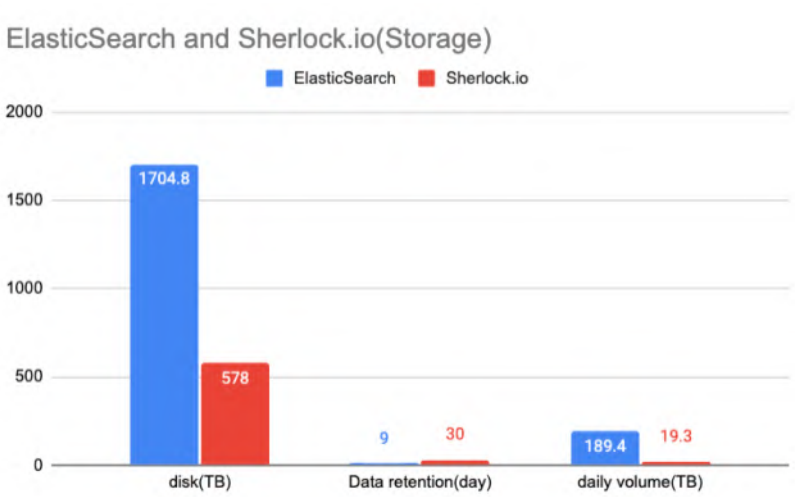
列裁剪和分区裁剪，用IN代替JOIN，大
小表JOIN时表顺序，Colocate JOIN

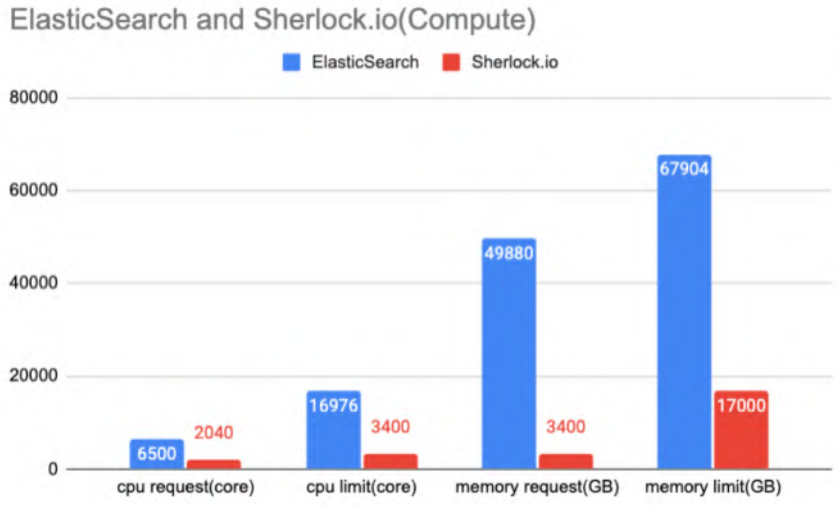
Part 03

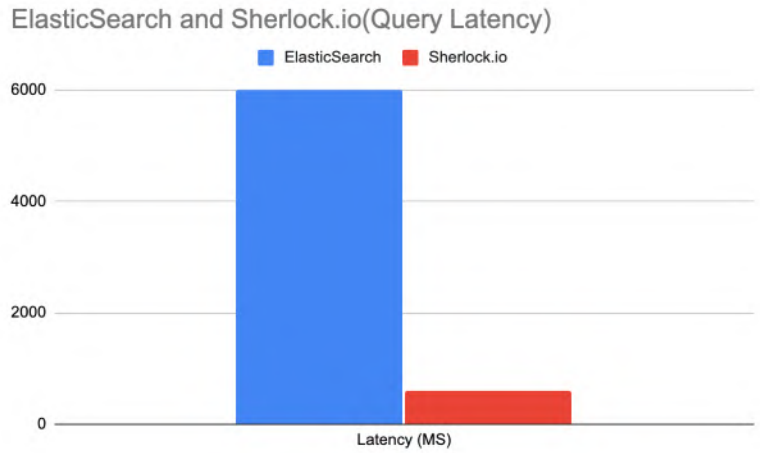
典型案例

服务网格访问日志异常检测

通过服务网格进行流量控制包括路由和负载均衡。服务网格的监控一直是个难点，主要是数据量很大，面向用户的服务网格访问日志一天有320亿条，并且有超过500个并发执行的异常检测的工作。
最初它是存储在ElasticSearch上的，当我们迁移到ClickHouse上，只用了30%的资源把保留时间从9天延长到30天，存储节省大约90%，并通过索引和物化视图的预计算把异常检测的查询性能也提高了10倍。
当然面向用户的服务网格只是一小部分，我们正在把整个公司的服务网格的访问日志放到事件平台上，大概有100倍的流量，从成本角度考虑我们会在边缘启用采样和预计算等来降低成本。







Part 04

未来展望

未来展望

- 扩展事件平台支持非结构化和半结构化日志
  - Free schema （Map+物化列 OR JSON）
  - In-region部署节省跨区域网络带宽

```
SELECT
   x,
   avg(y)
FROM
(
   SELECT
      x,
      y
   FROM region_1
   UNION ALL
   SELECT
      x,
      y
   FROM region_2
)
GROUP BY x
```

这个查询需要从远程节点分别拿到region1和region2两张表的原始数据传输到发起人节点后再聚合，这样效率是很低的。

```
SELECT
   x,
   avgMerge(sy)
FROM
(
   SELECT
      x,
      avgState(y) AS sy,
   FROM region_1
   GROUP BY x
   UNION ALL
   SELECT
      x,
      avgState(y) AS sy
   FROM region_2
   GROUP BY x
)
GROUP BY x
```

对region1和region2的表在远程进行预聚合，然后把预聚合的中间状态发送给发起人节点完成聚合，这样就可以避免原始数据的网络传输，从而大大提高查询效率。

Thanks

开放运维联盟
高效运维社区
DevOps 时代

荣誉出品