B站基于StarRocks构建大数据元仓和诊断系统

杨洋-bilibili-高级开发

DataFunSummit # 2023

目录 CONTENT

01

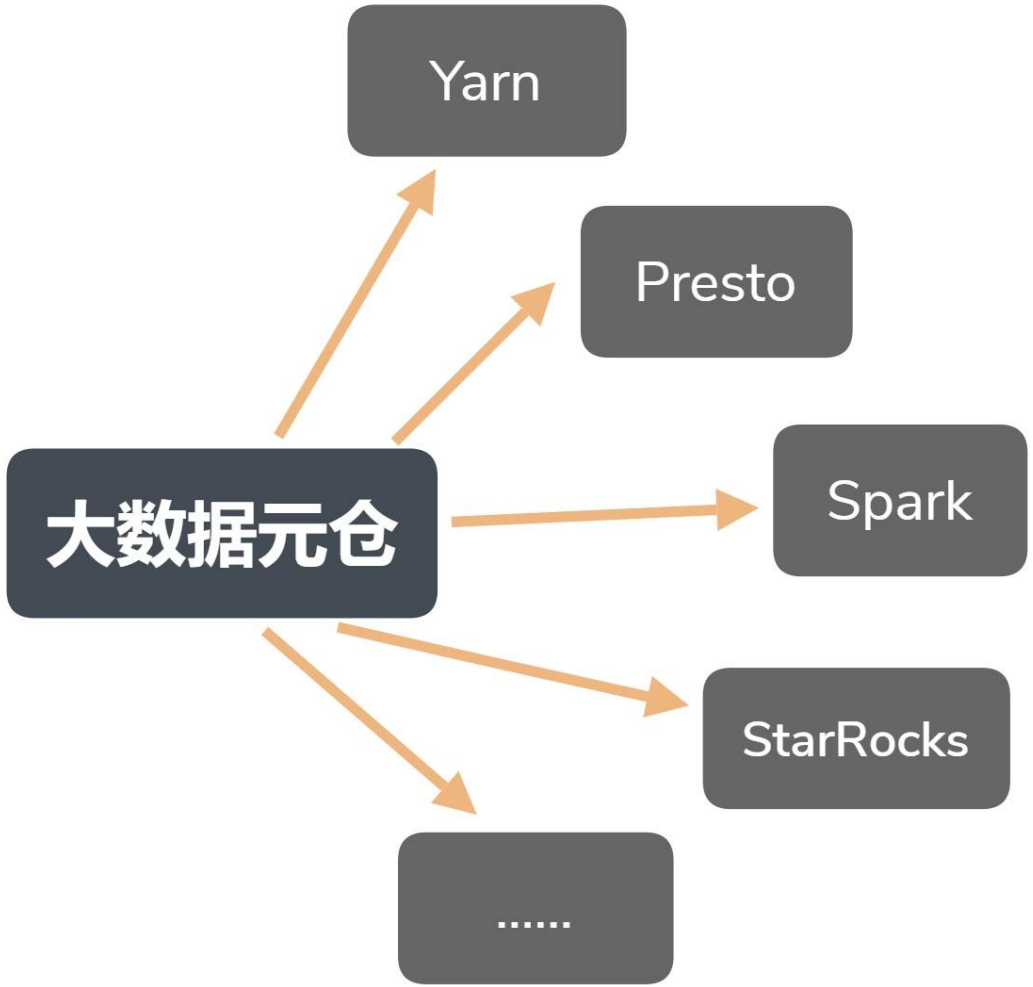
大数据元仓背景

DataFun.

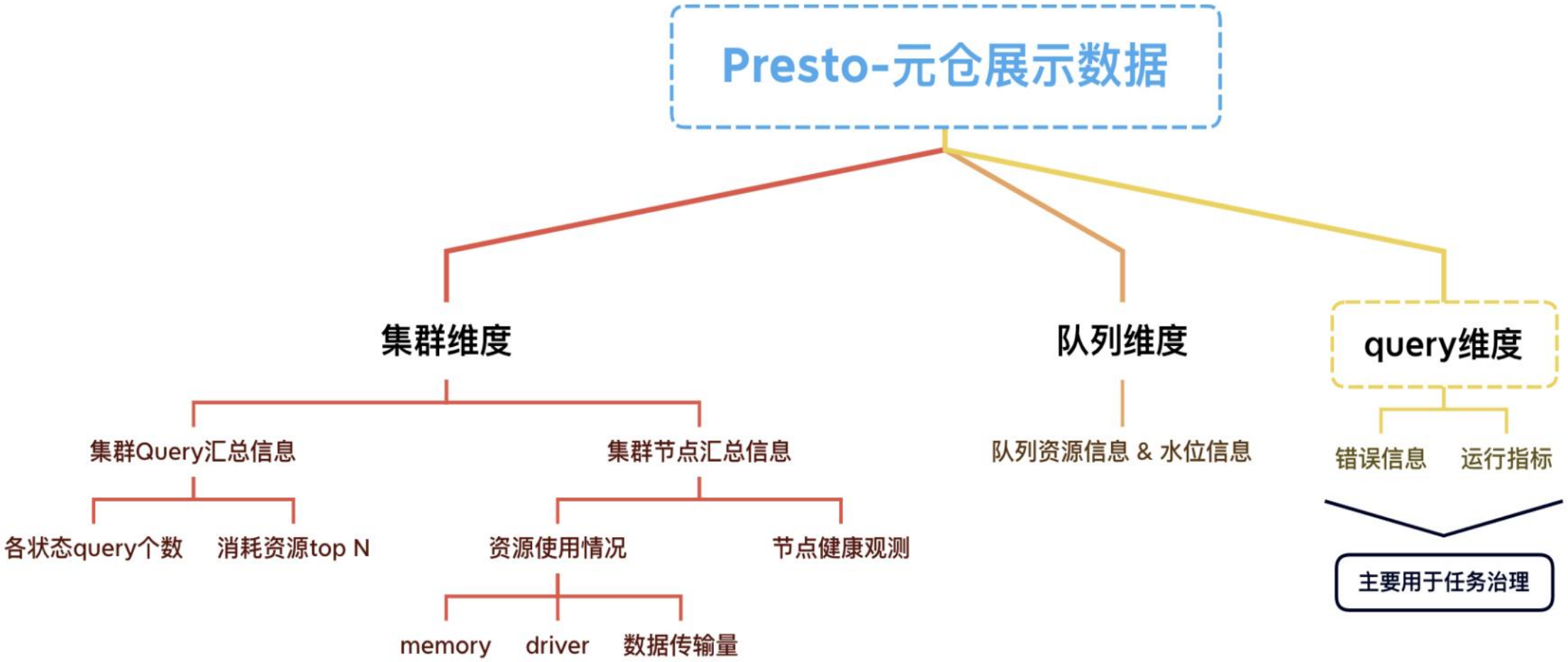
大数据元仓背景

主要背景：

- 缺乏对集群资源使用情况的观测在大数据引擎运行时
- 缺少作业维度的统计信息，难以推动用户治理作业。



大数据元仓背景


Presto-元仓展示数据
集群维度
队列维度
query维度
集群Query汇总信息
集群节点汇总信息
队列资源信息 & 水位信息
错误信息
运行指标
各状态query个数
消耗资源top N
资源使用情况
节点健康观测
memory
driver
数据传输量
主要用于任务治理

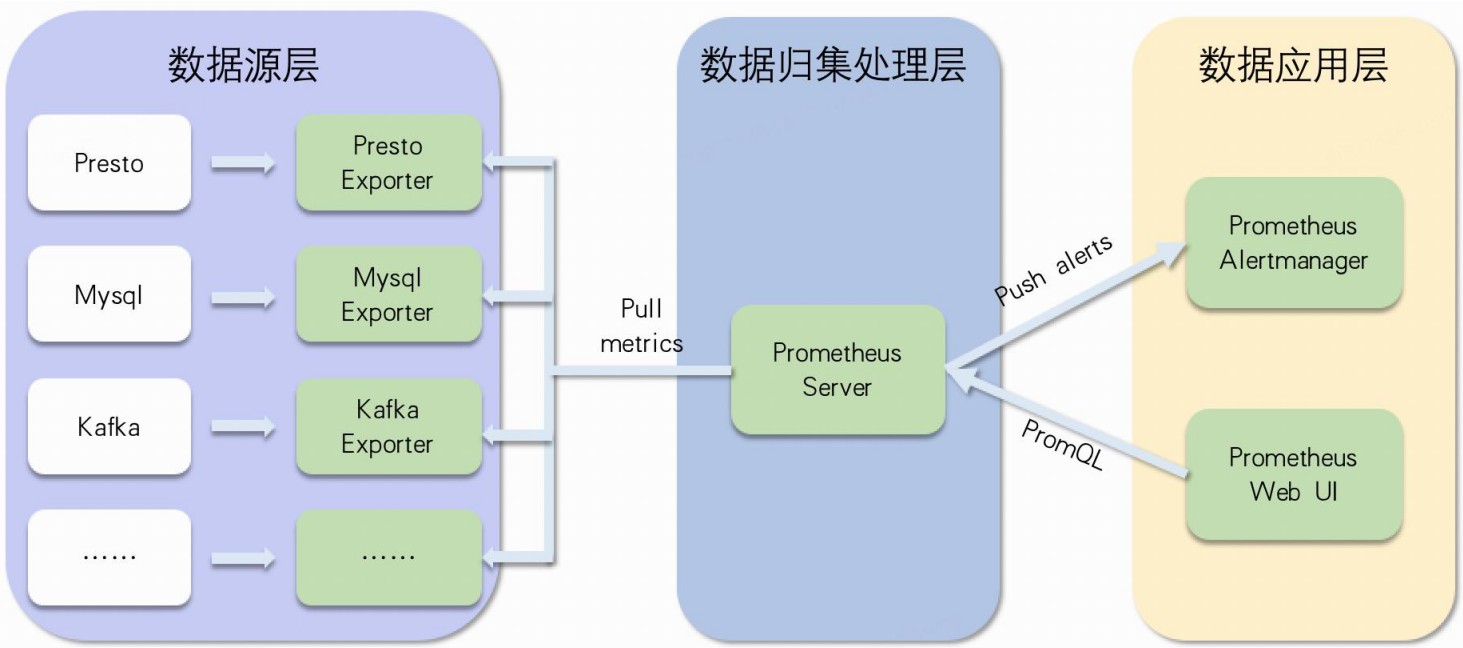
02

技术选型及方案

技术选型及方案

目前监控架构：

- 数据源层：通过exporter暴露集群指标
- 数据归集处理层：主要通过Prometheus server发送Http pull请求从数据源层拉取并存储监控的指标
- 数据应用层：展示指标数据及对异常数据做告警



技术选型及方案

Prometheus缺点：

- 存储数据量有限，不适合较久历史数据的回放
- 作为一个基于度量的系统，更多的是趋势性的监控，难以做到明细数据的存储与加工

大数据元仓特点：

- 能够实时观测集群指标数据
- 能够在多维分析中实现亚秒级查询返回
- 能够支持复杂计算逻辑
- 能够对半年甚至更久数据进行回放

技术选型及方案

技术选型-数据湖：

- 建于Hadoop体系上，具有更低的存储成本，更高的可靠性
- 支持ACID事务、支持Schema evolution，能够为用户提供更好的表格式
- 支持越来越多的索引及更好的文件组织格式，提升查询性能



技术选型及方案

技术选型-仓：

- 基于数据湖的远程IO成本很高，且缺少一系列数仓加速的手段
- 传统的数据湖实时性不够，在Hudi/Iceberg的支持下可能能解决分钟级别的时效性，但是无法解决秒级时效性的问题
- 对于高并发查询，不管是点查还是聚合类的查询，数仓是更擅长的

技术选型及方案

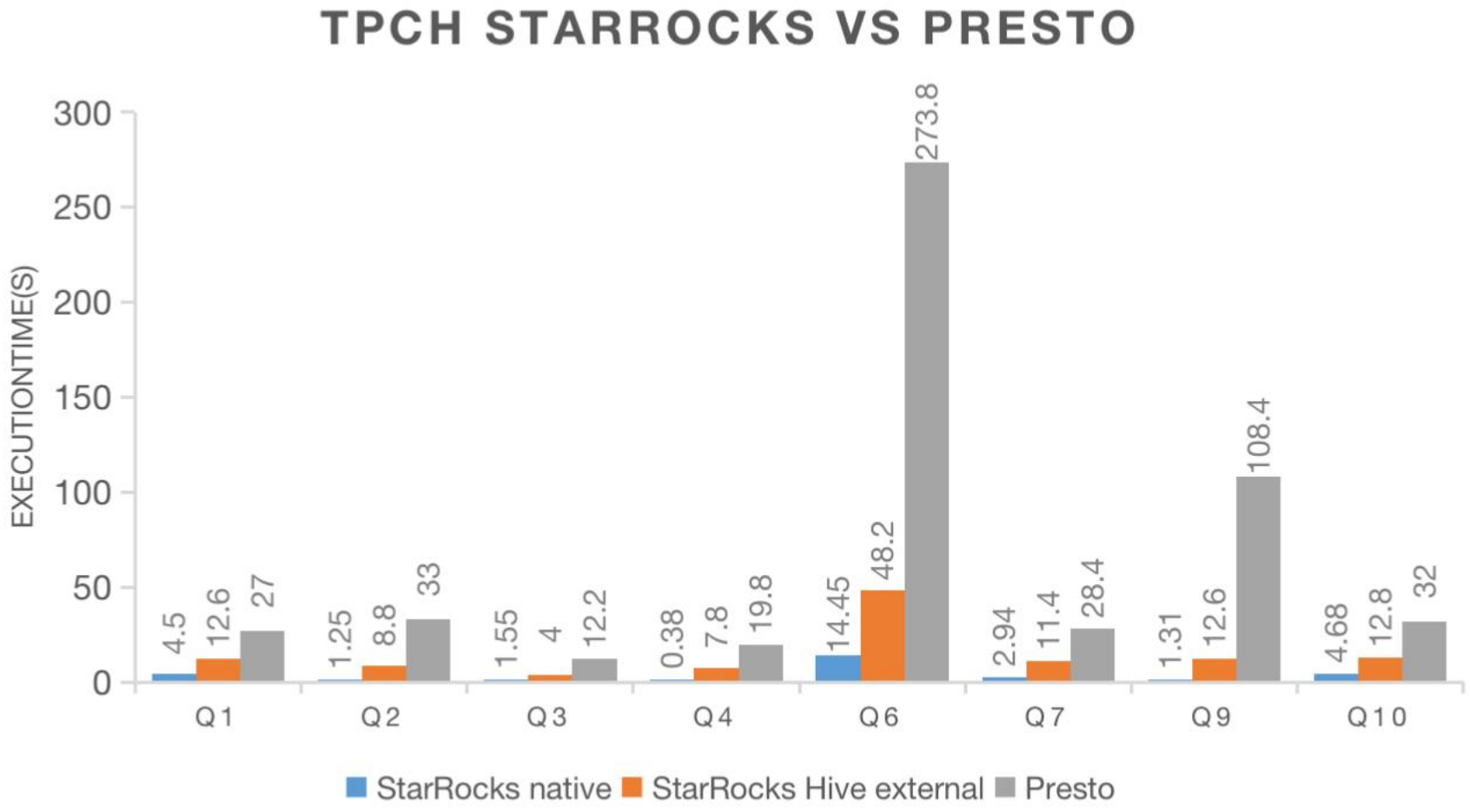
技术选型：StarRocks VS Clickhouse

功能	StarRocks	Clickhouse
标准SQL	支持标准SQL，兼容MySQL协议	不完全支持
性能	读写性能好	单机性能强悍
并发能力	支持高并发查询	并发能力弱，官方推荐QPS为100
Join能力	支持	Join能力弱，一般使用大宽表
运维	不依赖第三方组件，运维容易	依赖Zookeeper，运维成本高
社区	社区活跃度高，问题回复快	开源社区较好

技术选型及方案

StarRocks内外表与Presto性能测试：

效果：StarRocks（外表）相对于Presto缩短了约70%～80%的查询时间



技术选型及方案

StarRocks内外表与Presto资源消耗对比：

效果：StarRocks内表SQL查询资源消耗较小

Presto查询	q1	q2	q3	q4	q6	q7	q9	q10
memory	5.98g	12.9g	692m	1.29g	13.8g	29.4g	30.1g	5.57g
cpu	29.65m	30.72m	5.50m	11.54m	5.67h	25.23m	1.89h	30.08m

StarRocks内表	q1	q2	q3	q4	q6	q7	q9	q10
memory	2.5g	380m	880m	410m	8g	10g	2g	5.4g
cpu	6m57s	1s183ms	27s53ms	4s735ms	11m30s	24s297ms	37s84ms	1m41s

技术选型及方案

元仓架构图：

技术选型及方案

Kafka数据导入：

Routine Load

- 适用于原数据结构简单，无复杂逻辑处理
- 操作简便，只需配置相关参数
- 缺少监控与告警

Flink

- 适用于复杂的多表关联和ETL预处理
- 需要编写Flink代码处理业务逻辑
- 有监控与告警（公司流计算平台提供）

技术选型及方案

Flink导入遇到的问题：

背景：

- 鉴于部分数据（如日志）存在复杂的逻辑处理，原生作业导入方式（如Routine load）难以处理，因此需要采用Flink做transform后落库。
- 之前部分同学有使用Flink消费Kafka落Tidb的例子，再加上StarRocks兼容Mysql协议，不难想到稍做处理也可以落StarRocks。

现象：频繁Full GC（堆内存：-Xmx8192m）

```
2023-03-10T11:16:42.616+0800: 54484.963: [Full GC (Allocation Failure) 2023-03-10T11:16:42.616+0800: 54484.963: [CMS2023-03-10T11:16:45.335+0800: 54487.682: [
CMS-concurrent-preclean: 4.934/4.938 secs] [Times: user=24.83 sys=0.63, real=4.94 secs]
2023-03-10T11:17:07.681+0800: 54510.028: [Full GC (Allocation Failure) 2023-03-10T11:17:07.682+0800: 54510.028: [CMS2023-03-10T11:17:08.940+0800: 54511.287: [
CMS-concurrent-preclean: 5.200/5.204 secs] [Times: user=36.89 sys=0.98, real=5.20 secs]
2023-03-10T11:17:30.208+0800: 54532.554: [Full GC (Allocation Failure) 2023-03-10T11:17:30.208+0800: 54532.554: [CMS2023-03-10T11:17:32.331+0800: 54534.677: [
CMS-concurrent-preclean: 4.842/4.853 secs] [Times: user=27.46 sys=0.69, real=4.86 secs]
2023-03-10T11:17:54.140+0800: 54556.486: [Full GC (Allocation Failure) 2023-03-10T11:17:54.140+0800: 54556.487: [CMS2023-03-10T11:17:56.608+0800: 54558.954: [
CMS-concurrent-preclean: 4.718/4.722 secs] [Times: user=24.16 sys=0.59, real=4.72 secs]
2023-03-10T11:18:18.294+0800: 54580.640: [Full GC (Allocation Failure) 2023-03-10T11:18:18.294+0800: 54580.640: [CMS2023-03-10T11:18:20.194+0800: 54582.540: [
CMS-concurrent-preclean: 5.068/5.072 secs] [Times: user=32.37 sys=0.84, real=5.07 secs]
2023-03-10T11:18:40.626+0800: 54602.973: [Full GC (Allocation Failure) 2023-03-10T11:18:40.626+0800: 54602.973: [CMS2023-03-10T11:18:43.184+0800: 54605.531: [
CMS-concurrent-preclean: 4.818/4.822 secs] [Times: user=24.57 sys=0.61, real=4.82 secs]
2023-03-10T11:19:05.741+0800: 54628.088: [Full GC (Allocation Failure) 2023-03-10T11:19:05.741+0800: 54628.088: [CMS2023-03-10T11:19:06.721+0800: 54629.068: [
CMS-concurrent-preclean: 4.893/4.897 secs] [Times: user=33.12 sys=0.59, real=4.89 secs]
2023-03-10T11:20:06.999+0800: 54689.345: [Full GC (Allocation Failure) 2023-03-10T11:20:06.999+0800: 54689.345: [CMS2023-03-10T11:20:10.005+0800: 54692.351: [
CMS-concurrent-preclean: 5.049/5.053 secs] [Times: user=20.29 sys=0.37, real=5.05 secs]
2023-03-10T11:20:31.750+0800: 54714.096: [Full GC (Allocation Failure) 2023-03-10T11:20:31.750+0800: 54714.096: [CMS2023-03-10T11:20:34.549+0800: 54716.895: [
CMS-concurrent-preclean: 5.034/5.041 secs] [Times: user=23.21 sys=0.62, real=5.04 secs]
2023-03-10T11:20:54.696+0800: 54737.042: [Full GC (Allocation Failure) 2023-03-10T11:20:54.696+0800: 54737.042: [CMS2023-03-10T11:20:57.391+0800: 54739.738: [
CMS-concurrent-preclean: 4.859/4.863 secs] [Times: user=22.79 sys=0.42, real=4.87 secs]
2023-03-10T11:21:18.090+0800: 54760.436: [Full GC (Allocation Failure) 2023-03-10T11:21:18.090+0800: 54760.436: [CMS2023-03-10T11:21:20.742+0800: 54763.088: [
CMS-concurrent-preclean: 4.869/4.875 secs] [Times: user=18.34 sys=0.26, real=4.88 secs]
```

技术选型及方案

Flink导入遇到的问题：

解决方案：

- 对JVM参数调优（如调大堆内存），可能能缓解这种情况，但这种导入作业一多还是会出现频繁gc的情况
- 需从本质上解决Flink导入频繁的问题（代码层面）

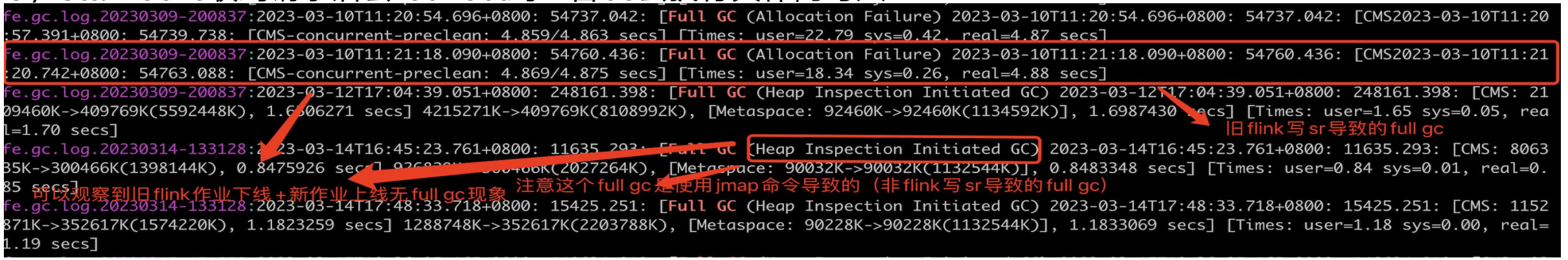
> StarRocks 提供 flink-connector-starrocks，导入数据至 StarRocks，相比于 Flink 官方提供的 flink-connector-jdbc，导入性能更佳。 flink-connector-starrocks 的内部实现是通过缓存并批量由 Stream Load 导入。

大致思路为：

1）Flink sink数据到StarRocks的时候，并非来一条处理一条会先进行攒批，写到一个内存的map结构中

2）启动一个异步线程，会去扫这个map（通常设置为5s），发起http请求（http://$fe:${http_port}/api/$db/$tbl/_stream_load）

3）StarRocks收到请求后会redirect到一台be去执行具体的写入



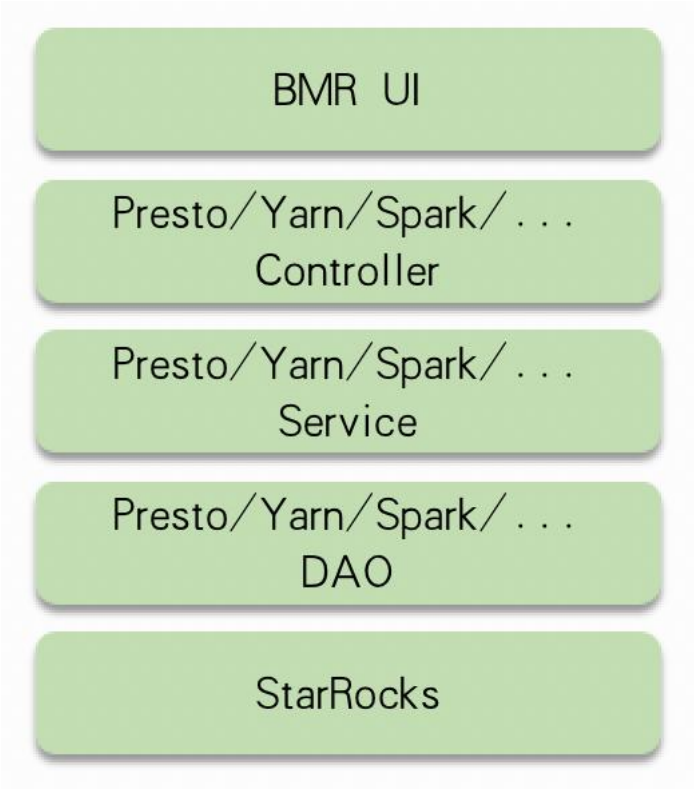
技术选型及方案

数据服务：

- 避免用户直连StarRocks，大sql带来系统稳定性问题
- 解耦用户与StarRocks，规范用户使用姿势
- 增强安全性，收归访问路径

实现：

- 基于SpringBoot构建元仓数据服务
- 鉴于StarRocks支持Mysql协议，采用Mybatis框架访问



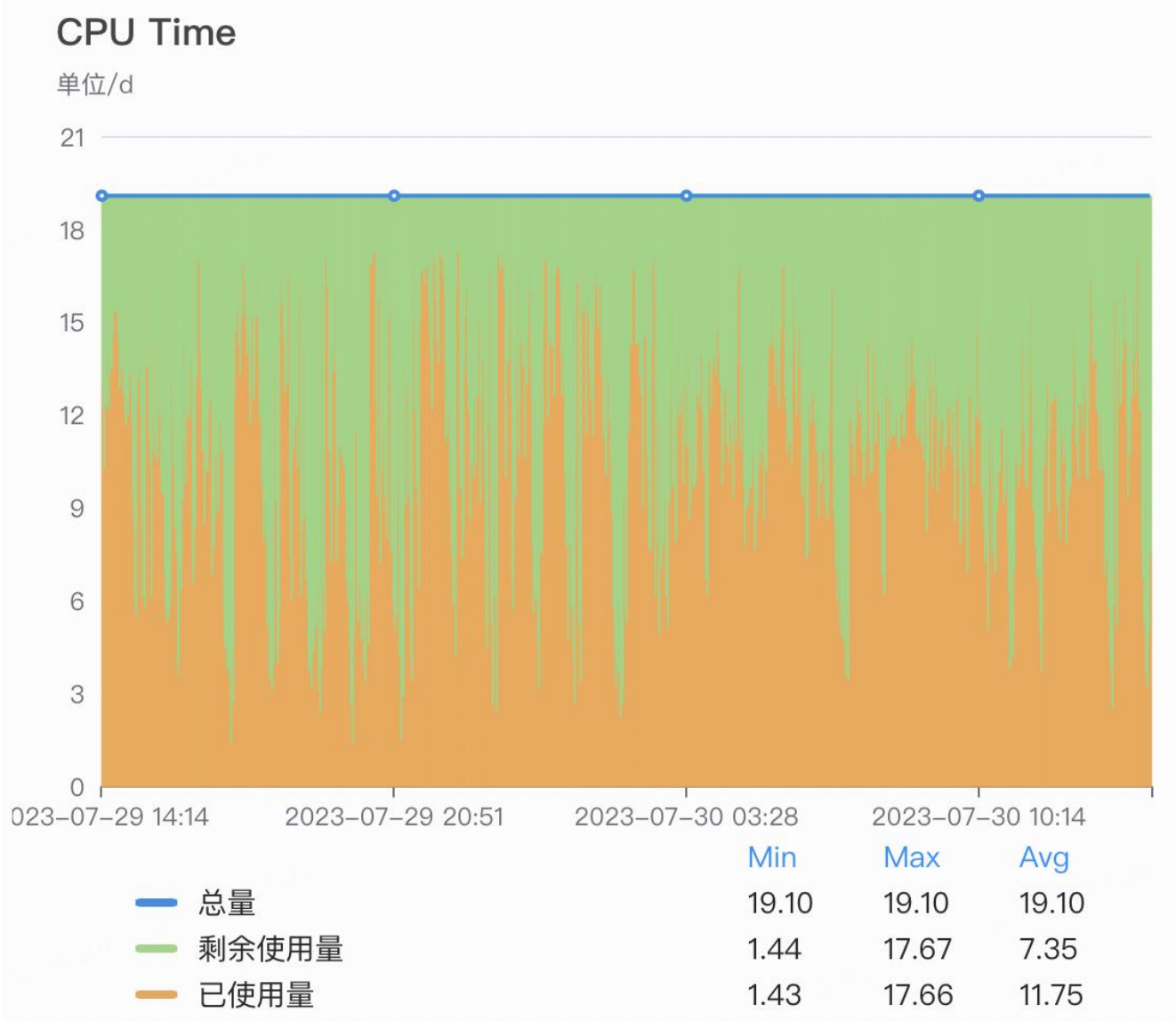
03

元仓与诊断效果

元仓与诊断效果

Presto元仓-CPU使用情况：

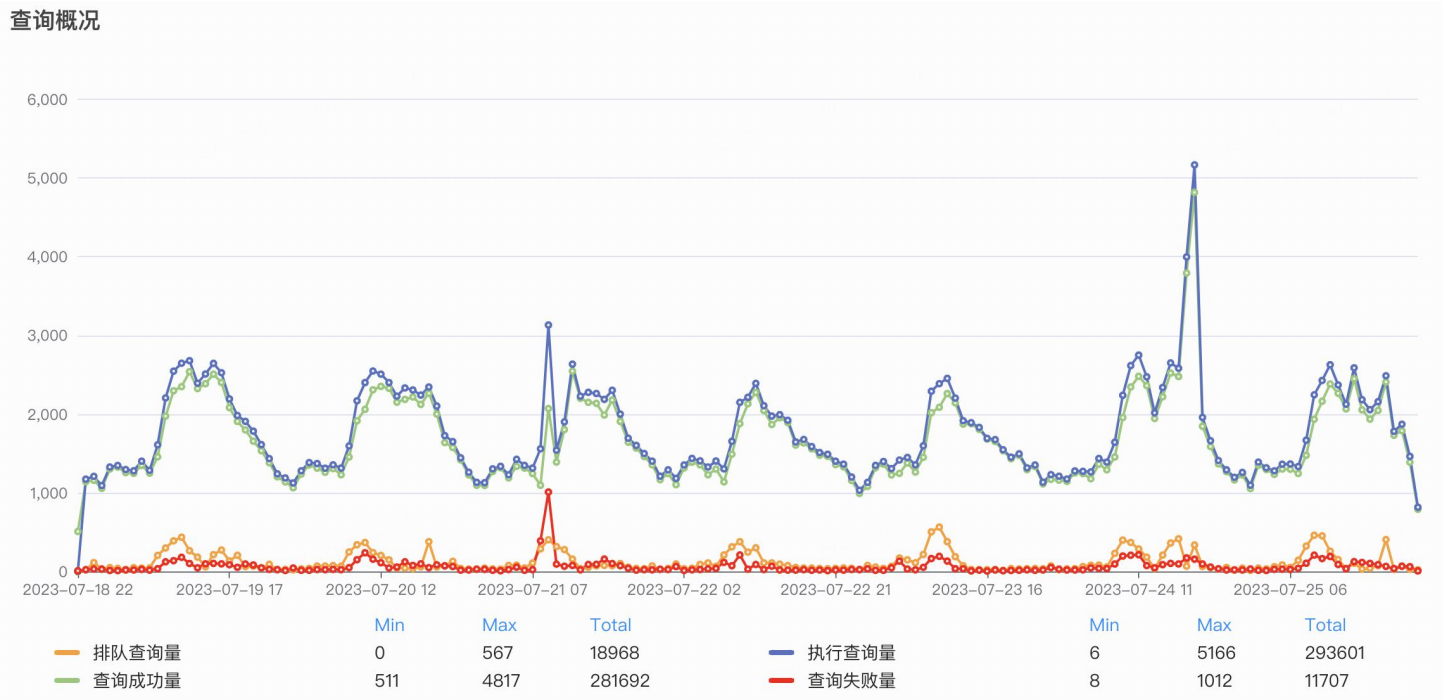
- 数据来源于Presto Worker节点的CPU Metrics
- 根据时间粒度+时间范围先按照时间字段分组，再进行每组的数据SUM聚合
- 图表上每个时间点的数据即为该时间点所有Worker进程的CPU Time总量、使用量、空闲量的SUM数值（单位为天）



元仓与诊断效果

Presto元仓-集群作业概况：

- 数据来源于Presto Cordinator节点的Query信息
- 图中展示为按小时粒度聚合一周的Query各种状态的查询数



元仓与诊断效果

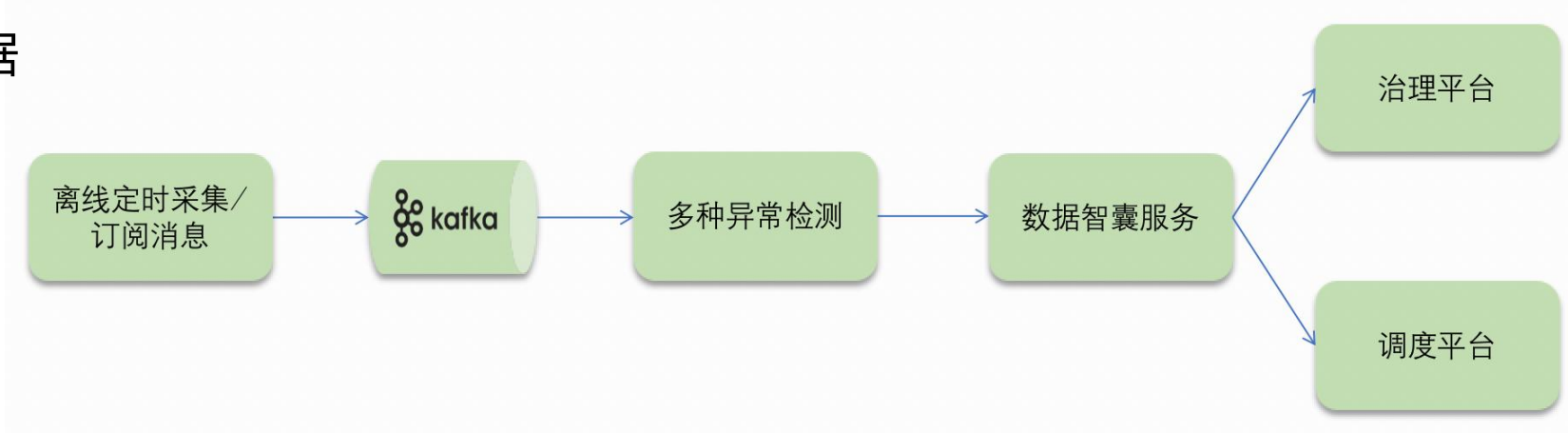
Presto元仓-作业失败异常概况

- 对Presto异常进行分类统计，划分为4大类
- 大类异常可以下钻到具体的异常类型

元仓与诊断效果

诊断系统：（基于OPPO开源项目罗盘改造）

- 完善元仓中任务异常指标数据
- 推动任务治理减少资源浪费
- 减轻值班人员压力



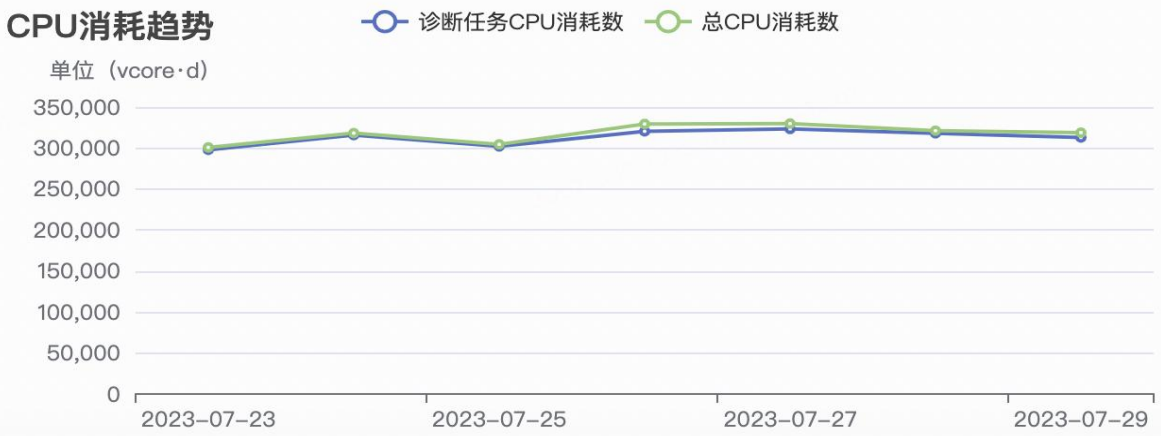
元仓与诊断效果

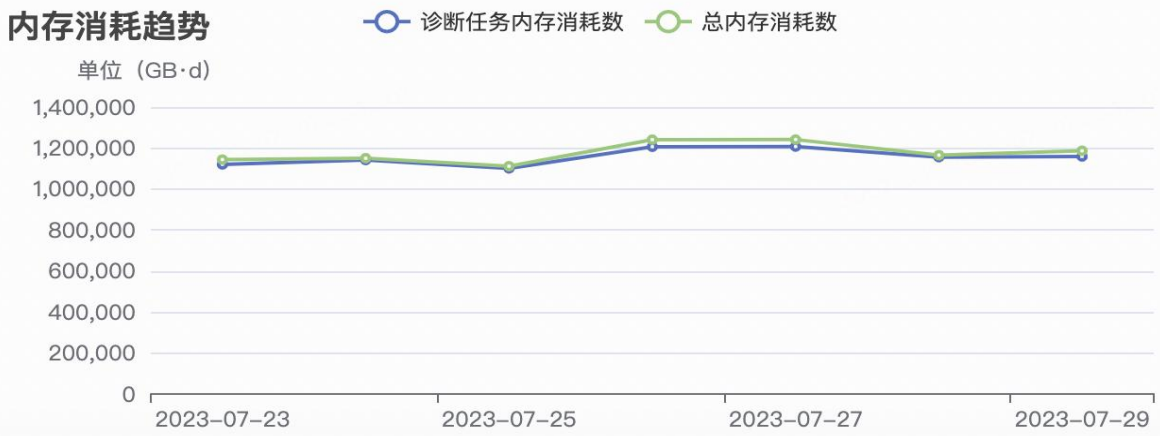
诊断系统-主页面

诊断异常任务数	占比	诊断异常实例数	占比	任务CPU消耗数	占比
13989	30.41%	28121	24.33%	142843 vcore·d	65.14%
诊断总任务数 45994	↑ 1.77% 环比 ↓ −8.48% 同比	诊断总实例数 115581	↑ 0.57% 环比 ↓ −13.64% 同比	总CPU消耗数 219287	↓ −3.15% 环比 ↓ −1.96% 同比

资源趋势　数量趋势

2023-07-23 – 2023-07-30





元仓与诊断效果

诊断系统-大表扫描分析

- 解析计划数分析scan数据量
- 对大表scan给出诊断建议

大表扫描分析(指标异常)

扫描的hive表名称	扫描行数	扫描行数阈值	扫描文件数	文件数阈值	分区数	分区数阈值	文件大小	文件大小阈值
[redacted]	7928.21亿	500.00亿	-1	5.00万	-1	5.00万	-1	5.0TB
[redacted]	107.48万	500.00亿	-1	5.00万	-1	5.00万	-1	5.0TB
[redacted]	107.49万	500.00亿	-1	5.00万	-1	5.00万	-1	5.0TB

分析结论：

当前[redacted]的扫描量 7928.21亿超过阈值500.00亿已超过。该实例可能存在以下情况

1.大表或全表扫描的情况，大表扫描或者全表扫描耗时长、浪费资源，也会给集群带来一定压力，且容易导致内存溢出任务失败，建议检查执行sql ，确认逻辑，添加分区条件，过滤不需要的数据，避免不必要全表扫描；

2.多次扫描大表的情况，建议将所有的目标数据进行一次大表扫描收集到临时表（或者通过global temp view缓存到内存中，视数据量而写，一般小于几十G的数据量可以缓存），从而将N次扫描大表转换为1次扫描大表+N次扫描小表，减少大消耗的操作。

元仓与诊断效果

诊断系统-全局排序异常分析

- 全局排序数据量过大导致
- 建议改成局部排序减少数据量

全局排序异常分析(指标异常)

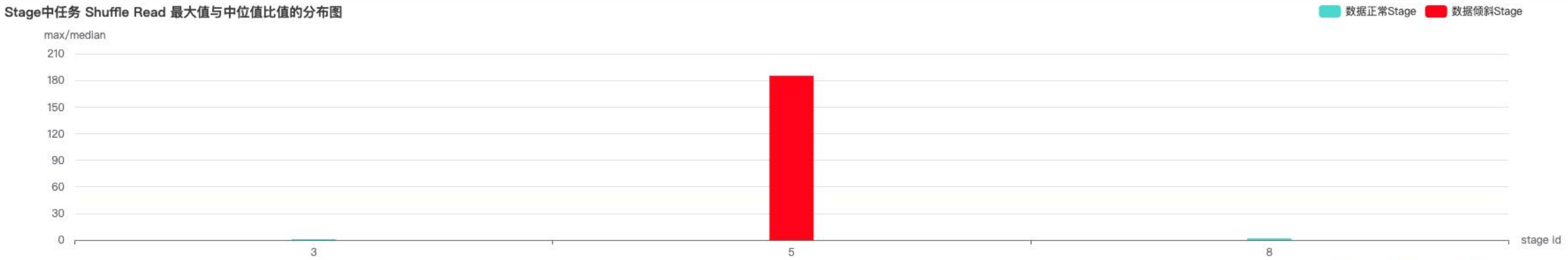
JobId	StageId	任务个数	处理数据量(行)	执行耗时
6	9	1	58.44亿	48.93m

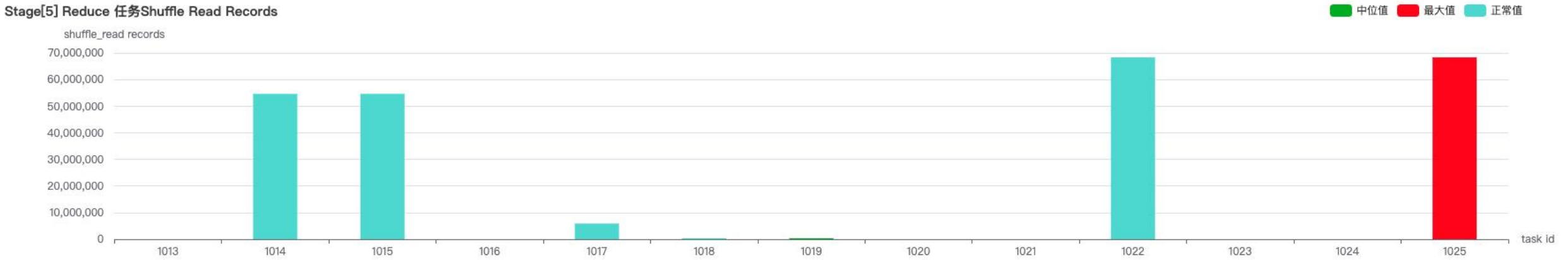
分析结论：	当前数据处理量过大, 发生全局排序异常。您可以将SQL改写成局部排序,下面举例说明:如存在表【CREATE TABLE test table (id INT,value STRING)】, 需对每个分区内的数据进行排序, 可使用【SELECT * FROM (SELECT *, ROW_NUMBER() OVER (PARTITION BY id ORDER BY value) AS row_num FROM test_table) tmp】对整个数据集先使用ROW_NUMBER进行分区内排序, 减少全局排序的数据量,从而提高排序性能。

元仓与诊断效果

诊断系统-数据倾斜分析

数据倾斜分析(指标异常)



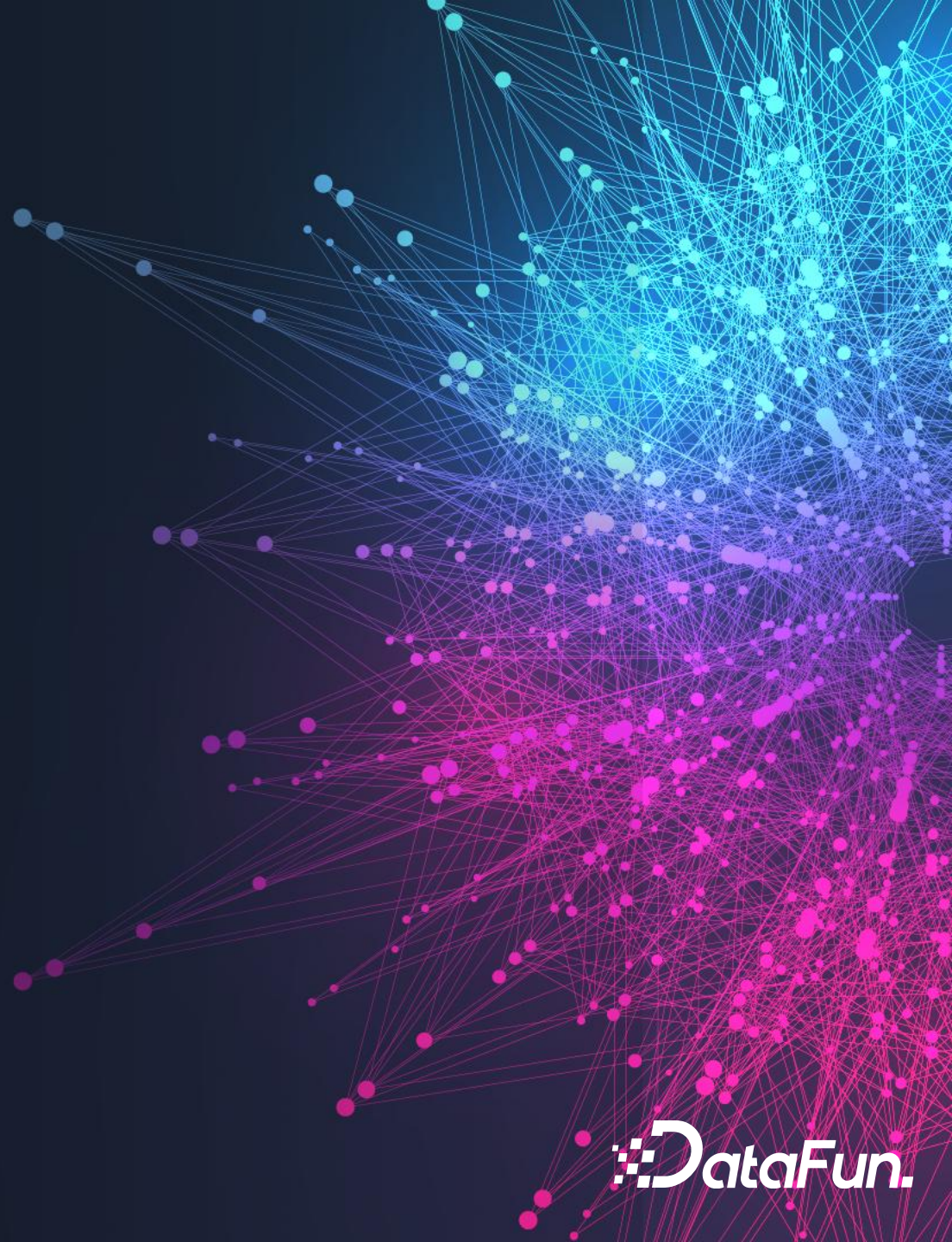


分析结论：

job[3].stage[5].task[1025]shuffle read的数据量为6831.80万 中位值为36.87万，读取的数据量严重超过Stage下的中位值，发生数据倾斜。
具体解决方法可参考如下建议：
1、找到倾斜阶段对应执行的sql
2、查看关联字段是否有倾斜（join on，group by，partition by 的字段）
3、根据任务需要，如果异常值不需要可以过滤（字段！=异常值）；如果异常值也需要，可以单独处理（第一段sql排除异常值，union all 只有异常值的数据）

04

总结与未来规划



DataFunSummit # 2023

DataFun.

总结与未来规划

总结：

- 尝试StarRocks初步落地公司应用场景
- 构建大数据元仓为用户提供资源消耗实时观测能力
- 通过诊断系统推动用户治理异常作业

未来规划：

- 尝试StarRocks接入公司更多业务（如BI、DQC）
- 解决StarRocks权限、UDF等问题，与其他引擎对齐
- 开启StarRocks物化视图、索引等为查询提速
- 接入更多组件如HDFS、Kyuubi等至大数据元仓
- 升级大数据元仓架构从仓至湖仓一体
- 支持Presto与Flink的智能诊断
- 打通内部其他平台，提供专业诊断建议

欢迎关注

DataFunSummit # 2023

DataFun.


感谢观看
DataFun.