

Apache Doris 在天翼云的最佳实践

李康 大数据总监

目录

01 整体介绍

02 案例速览

03 经典案例

04 展望未来

01

整体介绍

整体规模

20₊

落地项目

50₊

集群规模

3千₊

总节点数

应用场景



多维分析



日志检索



湖仓一体

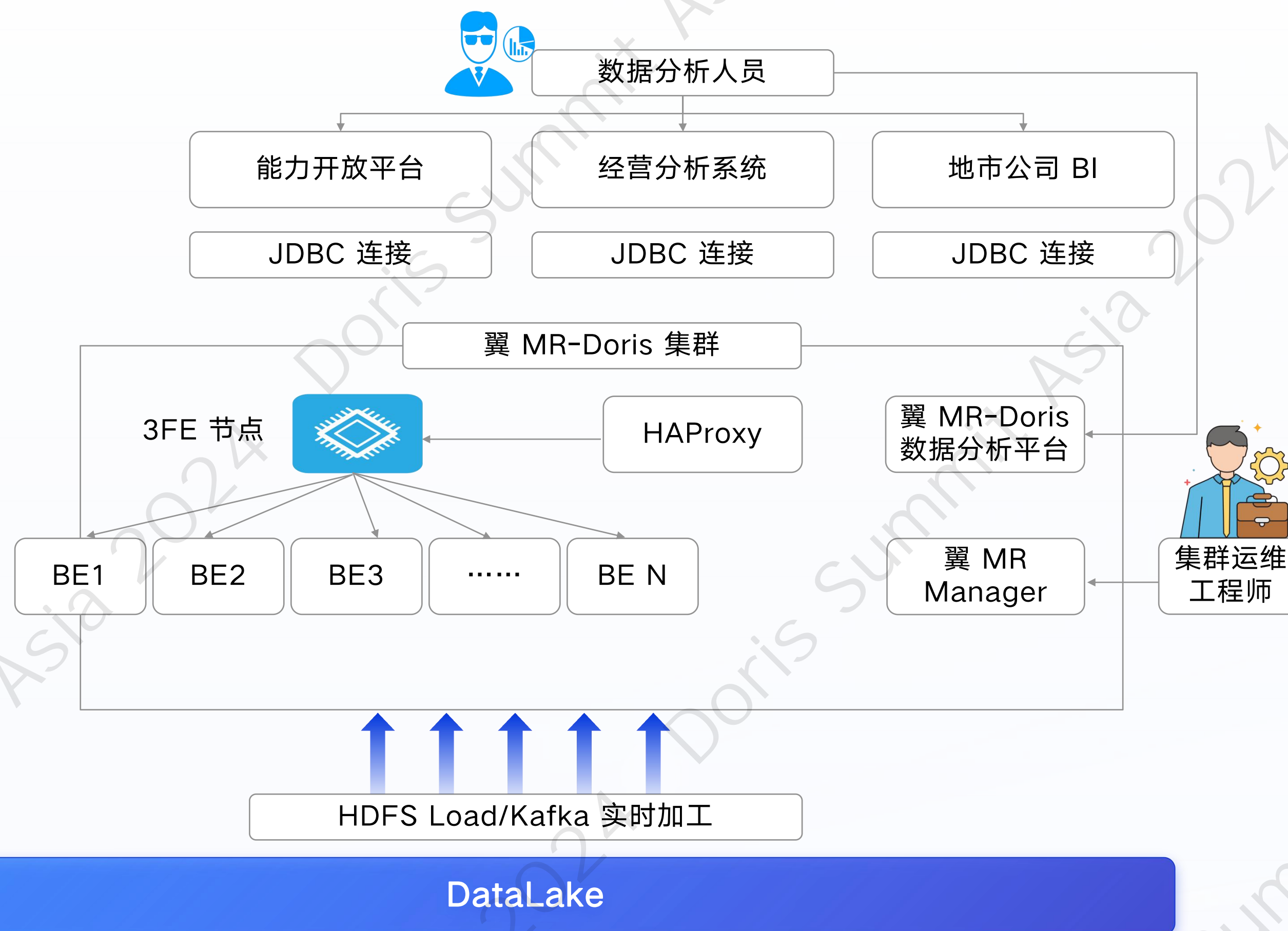


BI 报表

02

案例速览

即席多维分析案例



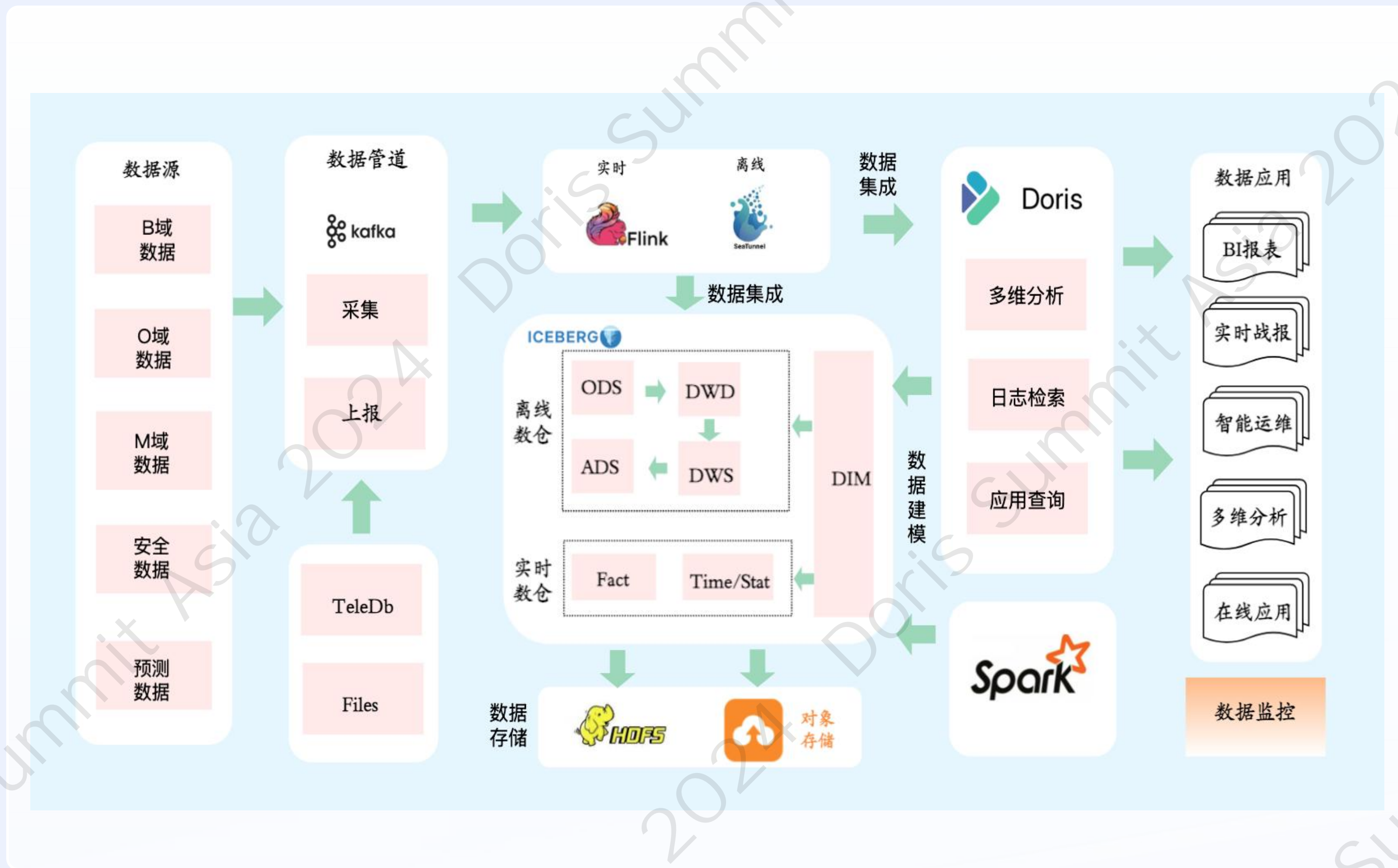
案例简述:

主要是通过 Doris 作为数据分析和即席查询工具提升各省数据平台的查询效率, 促进数字化转型。

案例成果:

- 替代经分系统原 [Impala+Redis](#) 架构, 解决 Impala 稳定性差和 Redis 缓存量有限的问题, 提升了报表周期、优化数字化业务经营的手段和效果。
- 替代数据集市 [Oracle](#) 数据库, 通过能开平台直连 Doris, 满足一线 0.4s~0.7s 数据调用及查询的要求, 实现数据集市去 O 的改革目标。
- 替代地市公司 BI 系统使用的 [PG](#) 库, 在数据分析过程中的计数、聚合、单条快速检索运算等情况秒级可见。

湖仓一体案例



案例简述

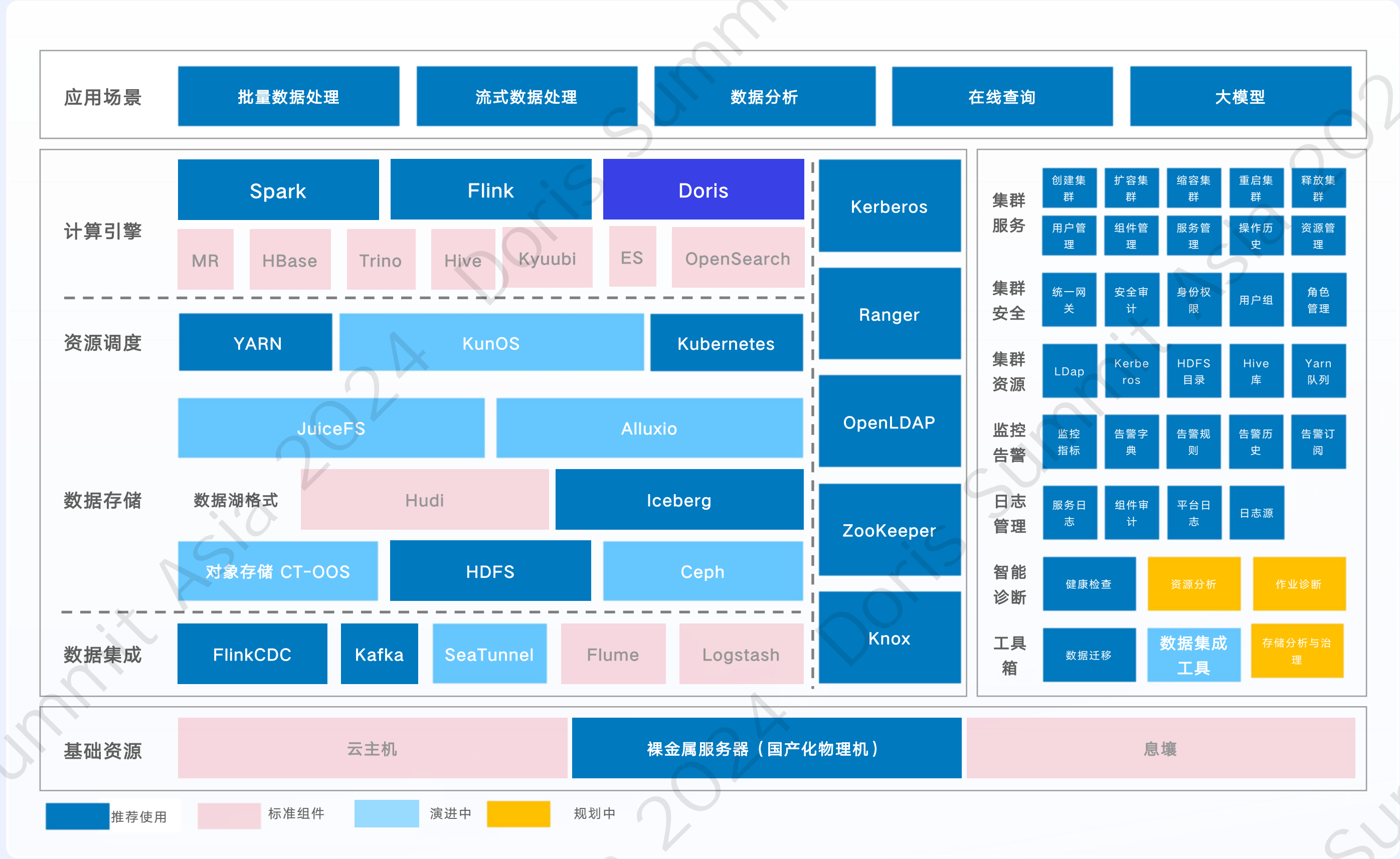
主要是使用 FlinkCDC+SeaTunnel 集成工具 Doris+Spark 计算引擎，以及 Iceberg 结合 Hdfs/对象存储等实现湖仓一体的统一架构。

- 通过自研 TaskFlow 集成工具把 FlinkCDC(实时)和 SeaTunnel(离线)融合在一起发挥各自的优势公共组成流批一体的数据集成体系。
- 利用 Doris 和 Spark 计算引擎的分析和加工能力, 实现数据建模、数据分析、数据查询等。
- 通过 Iceberg 把 Hdfs 和对象存储融合在一起作为数据统一的存储层。

案例成果：

- 自研了 TaskFlow 提供**流批一体的数据集成功能**，利用 Iceberg 提供了**统一的存储能力**。
- 利用 Doris 的多维分析和日志检索能力，提升了在线应用和 BI 报表、日志系统的**查询效率**。
- 通过 Amoro 对 Iceberg 小文件进行5分钟粒度的合并，**数据可见和性能较 Hive 两倍提升**。

国产化案例



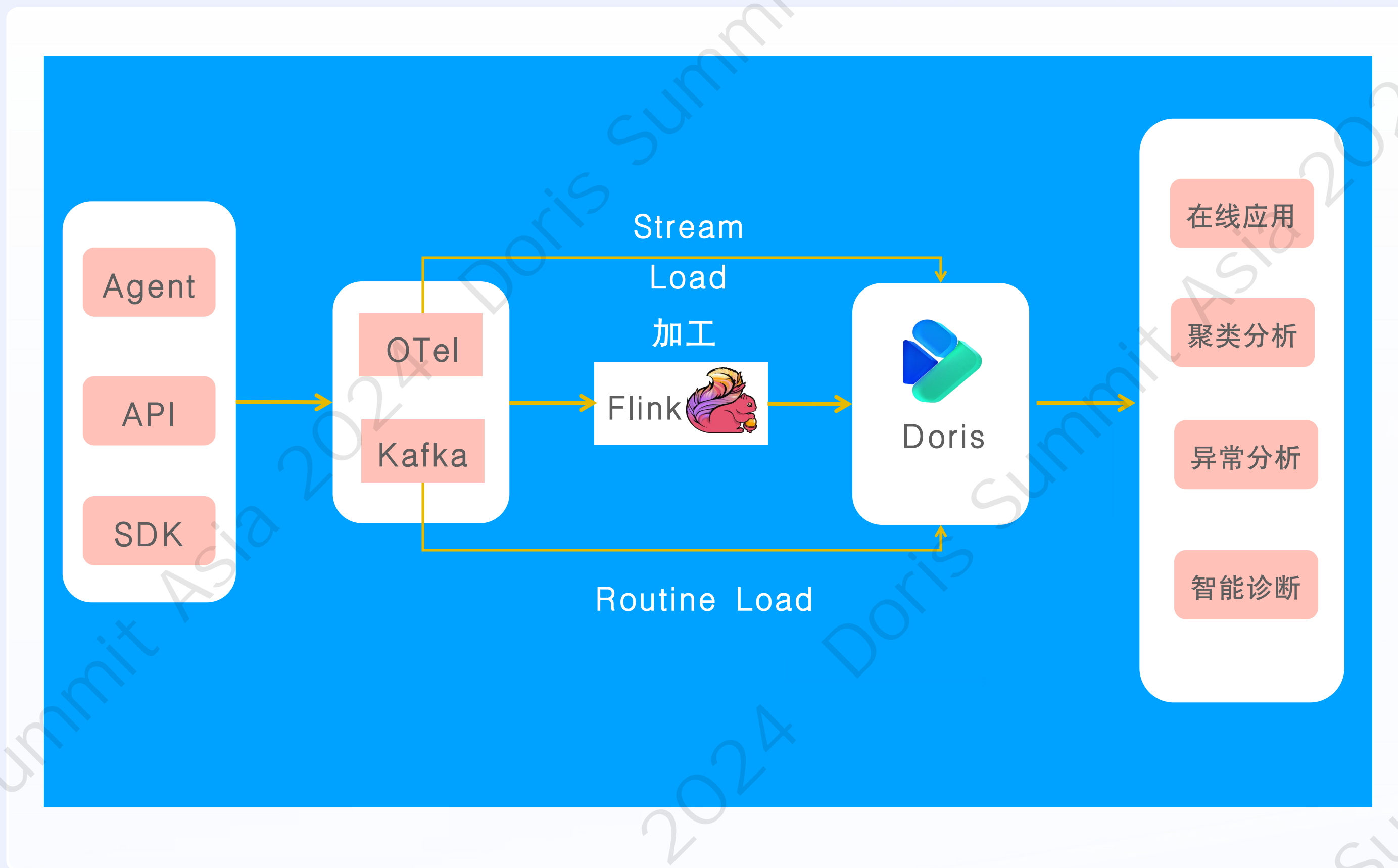
案例简述：

在国产化基础设施（CPU、OS）上，围绕采集、存储、调度、计算等四层核心业务构建国产化大数据平台，而 Doris 在其中主要承担大数据 **MPP 计算** 的重要角色。因此需要使 Doris 在鲲鹏芯片上落地并对 Doris 做了权限扩展和安全加固，为国产化大数据平台添砖加瓦。

案例成果：

- 利用毕昇编译器提升了 Doris 编译效率**30%**。
- 通过优化 Bitshuffle 提升运行效率**25%**。
- 通过安全加固提升了数据的安全性。

日志检索案例



案例简述:

通过 Doris 替换传统 ELK 的日志架构，提升日志系统的查询效率。满足省市在安全网关、系统运维等场景对日志检索，存储，以及智能分析的业务需求。

案例成果:

- 写入吞吐提升了5倍。
- 存储成本降低了80%。
- 百亿级日志检索秒级，查询效率提升3倍。

03

经典案例

高并发即席查询

背景：

基于天翼云3AZ多活的云资源池，应用云原生技术完成了电信内部某子公司的大数据平台(AIoT)架构升级。而Doris在这个项目中角色主要是为了给业务应该提供大规模应用系统的数据查询，替换掉原有的Orical数据库，助力公司内各行业应用规模发展。

业务特性：

- 高并发1W+。
- 单表数据规模庞大（总表百亿，单日几个亿）。
- 结果集小（大多数只有单行结果）。
- 表数据需要进行实时update操作。



如何实现海量数据的高并发查询

思考

通过业务特性可知业务单表数据量很大，但是有一个特点就是大多数的查询都是单行结果，是典型的点查场景。这相当于在大海中捞一根针但是庆幸的是 Doris 的数据分片粒度足够小（先分区在分桶）、索引丰富（内建智能、布隆、倒排等）因此我们需要好好利用这两个特性来尽量压缩数据的扫描范围。

● 分区分桶

我们先把数据按天分区，然后按 owner_id 字段进行分桶。理论来说是分桶数越大数据分片越小查询效率越高，但是 Meta 越大会随 Tablet 数的增加而增加，影响了数据管理同步和写入性能，因此我们需要通过压测选取合适的分桶数，经过多组实验和权衡100个分桶是比较合适当前场景的。

● 索引

智能索引（前缀索引和 ZoneMap ）是 Doris 的内建的在创建表时默认给每个字段都建立相应的索引，并当前只有是点查场景只需要建立布隆过滤器即可。

测试方案

使用 jmeter 发起300W的请求，每秒并发1W，持续压5分钟。owner_id 使用离散数据压测，样本数为5000

集群配置

节点类型	数量	cpu	内存	磁盘类型	单磁盘大小	磁盘数量
fe	3	48	376G	-	-	-
be	6	48	376G	SATA	10t	12
broker	3	48	376G			

测试数据

轮次	并发数	Tablet	95分位耗时ms	吞吐率
1	1W	300	22	3137
2	1W	200	17	4100
3	1W	100	8	4584
4	1W	50	9.7	4325
5	1w	30	7	4376
6	1W	20	8	3932

高并发查询措施&效果

解决措施

- 经过分区分桶+库表统计信息把上百亿大表在执行计划阶段确定目标的tablet，把数据缩小到百万级别。
- 在利用智能索引和布隆过滤器索引进一步过滤扫描的segment。

效果



容量评估

思考

数据存储和写入资源评估相对比较固定按需分配即可，但是计算资源评估是比较难的一个问题，往往和查询的业务复杂度、请求频率、数据规模与分布、任务的执行效率等因素有关。任务的执行耗时取决于计算引擎性能，其它因素主要取决于业务并且是动态变化的，这给资源评估带来了极大的困难。通过分析我们明显可知业务的场景(点查)和数据量是相对固定的，加上分区和分桶策略我们基本固化了数据分布，目前只需要明确执行引擎在现在场景中的执行耗时我们就能通请求频率计算出，因此我们需要对Doris的在点查场景进行一次压测。

测试方案&数据

使用jmeter发起180W的请求，持续压5分钟，每秒请求6000。

集群配置

节点类型	数量	cpu	内存	磁盘类型	单磁盘大小	磁盘数量
fe	1	48	376G	-	-	-
be	6	48	376G	SATA	10t	12
broker	3	48	376G			

测试数据

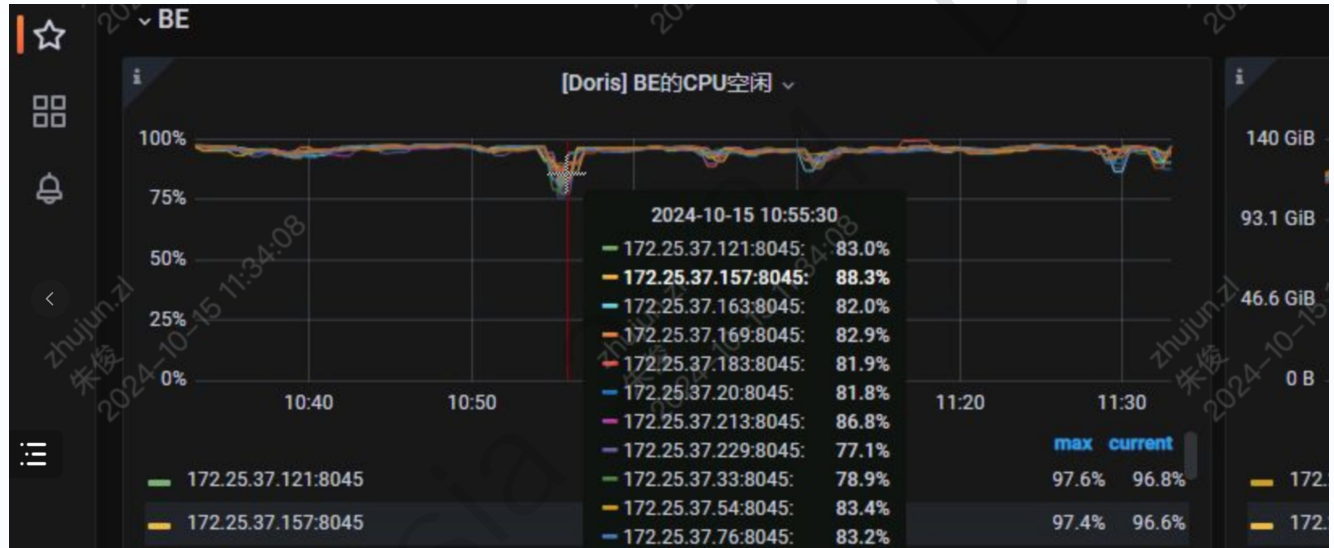
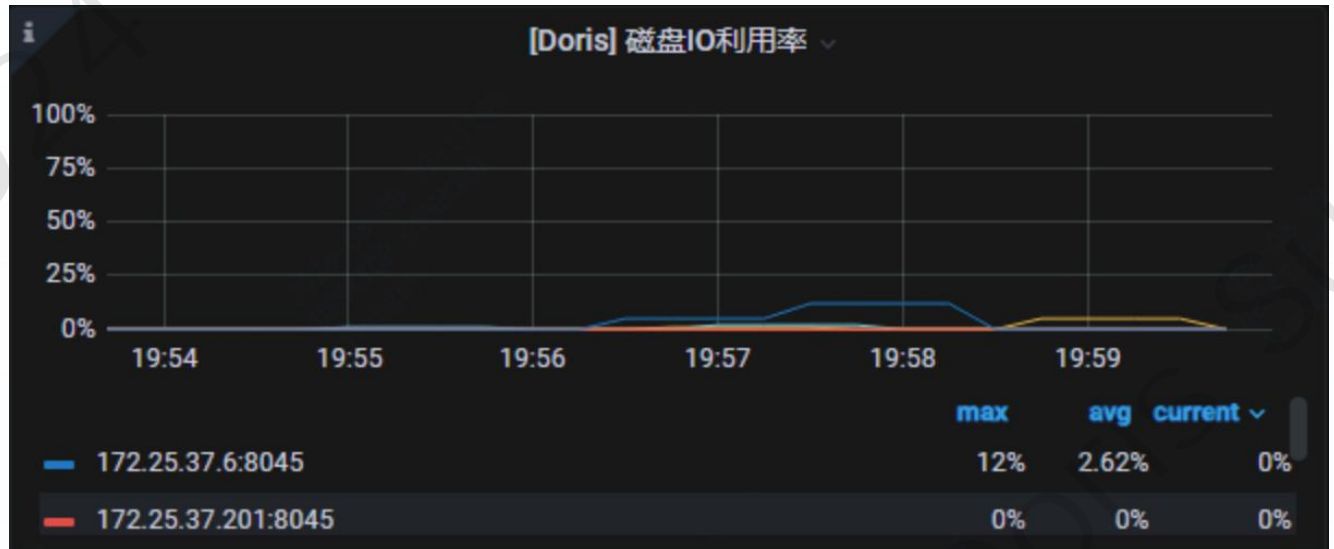
轮次	线程数	80分位 耗时ms	95分位 耗时ms	99分位 耗时s	吞吐率
1	512	500	820	2	4137
2	1024	550	912	5	4609
3	2048	536	789	3	4984
4	4096	512	873	2,8	5125
5	8192	1200	3500	12	3332

QPS压测现象&结论

- Fe和Be节点的负载(CPU/IO)都比较低，并没有随任务数的增加的加大。
- 出现这个现象是因为业务场景是点查场景结果集小，并通过分区分桶+索引策略数据过滤率很高，因此IO和CPU都不是一很高。
- QPS到达5000千多之后调整各级线程数和队列大小、执行并发度等参数QPS都升不上去，并线程和队列资源使用率并不高。
- 执行并发度无效是因为当前查询命中的基本上是一个tablet，任务基本只有一个instance来执行，并行度基本起不到作用。
- 线程和队列资源的使用率不高但是QPS上不去可能是系统各级锁竞争引起。
- 任务耗时在任务5000以下任务95分位的任务执行耗时都很低ms级，在5000+95分位的任务执行耗时会逐步增加。

实验结论：

综上所述点查场景下单节点的QPS可以达到5000，因此当前fe节点数据就是所需QPS/5000，而BE是可以横向扩展的，因此可以是可以按照QPS/(5000/3/节点使用率)。



如何实现数据快速导入01

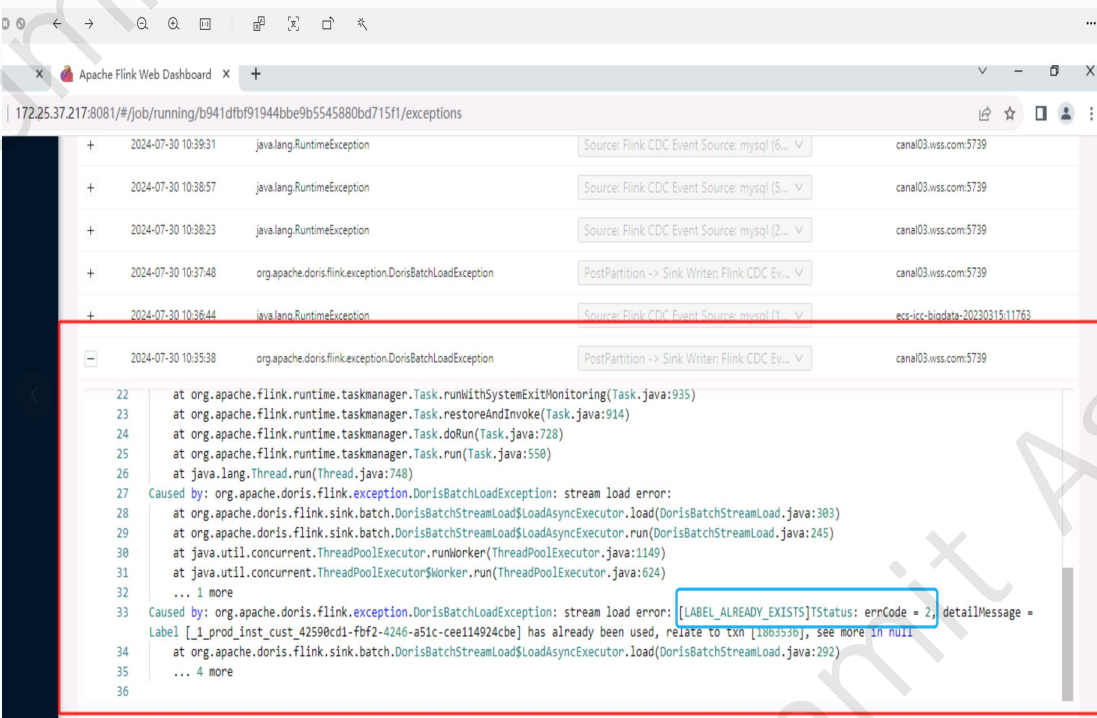
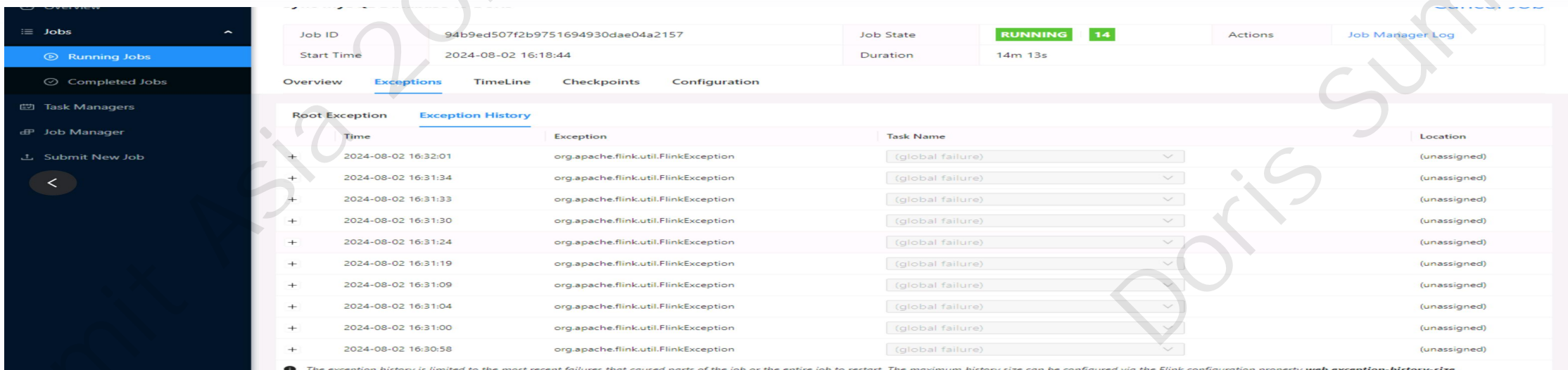
思考

在数据导入上首先要考虑的是如何把数据从Oracle导入到doris中，因为我们大数据产线FlinkCDC的集成产品也已经比较成熟了，所以FlinkCDC的集成方案就成为我们的首选。于是我们对该方案做了压测，在压测中发现如下问题：

- FlinkCDC的挂掉、频繁重启。
- Doris的经常抛出Lable已存在的问题。
- 数据导入性能比较低，只能写入几百条每秒。
- 集群的BE资源IO占用过高。

测试方案&现象

使用Flink全量的导入Oracle的10个Instance的数据，数据量300G+。



解决Doris抛出Lable已存在的问题

思考

经过对Doris的Lable标签设计原理和flink-doris-connector的分析，最后发现是因为flink-doris-connector使用steamload把数据写入。而steamload是通过http协议把数据写入到Doris中的，在http的类库中默认是开启了重试策略的。如果网络异常或延迟都是触发客户端的重试，而Http的重试是不保证数据ExactlyOnce语义的。因此我们关掉了http的重试，并在connetor层做重试，每次重试的时候会改变Label，进而解决了再不丢数的情况下避免因Label重复导致的flink任务挂掉的问题。

```
putBuilder.setLabel(label);
putBuilder.addCommonHeader()
putBuilder.setEntity(entity)
putBuilder.addHiddenColumns(executionOptions.getDeletable())
putBuilder.addProperties(executionOptions.getStreamLoadProp());

int retry = 0;
String usedLabel = label;
while (retry <= executionOptions.getMaxRetries()) {
    LOG.info("stream load started for {} on host {}", usedLabel, hostPort);
    try (CloseableHttpClient httpClient = httpClientBuilder.build()) {
        try (CloseableHttpResponse response = httpClient.execute(putBuilder.build())) {
            int statusCode = response.getStatusLine().getStatusCode();
            LOG.info("receive stream load status code : {}", statusCode);
            if (statusCode == 200 && response.getEntity() != null) {
                String loadResult = EntityUtils.toString(response.getEntity());
                LOG.info("load Result {}", loadResult);
                RespContent respContent =
                    LOG.error("stream load error with {}, to retry, cause by", hostPort, ex);
            }
        }
        retry++;
        // stream load失败进行重试时，重新设置标签，避免be标签重复错误
        usedLabel = label + "_" + retry;
        putBuilder.setLabel(usedLabel);
        LOG.info("stream load new label name : {}", usedLabel);
        // get available backend retry
        refreshLoadUrl(buffer.getDatabase(), buffer.getTable());
        putBuilder.setUrl(loadUrl);
    }
    buffer.clear();
}
```

如何解决Doris写入慢的问题01

思考

我们对flinkCDC写入Doris流程做了比较深入的分析，flinkCDD当前主要是通过streamload的方式进行写入，streamload提供了一种可以直连BE进行数据传输的方式，Doris-flink-connector也是采用这种方式进行数据导入，在这种情况下影响写入性能主要是由tablet数量，数据模式（Unique、Duplicate、Aggregate），表存储方式（MoW、MoR），streamload的数据块大小，硬件资源（网络、IO）5个方面的因素影响的。

- Tablet数量：

该因素主要是影响到写入数据块在Doris侧的数据分发，进而影响到磁盘的IOPS和网络。因为tablet数多大数据比较分散，一次streamLoad的数据就被分得比较散写入磁盘的次数比较多，不仅消耗磁盘的IOPS同时增加了随机写磁盘的次数，因此tablet应该比较小。但是过小的tablet数又影响查询效率，因此我们需要在写入性能和查询性能上做一个权衡。

- 数据模式：

这个主要是和业务场景相关，不同数据模型下数据block的写入性能不一样，通常来说Unique性能是最好的，单tablet写入性能可以达到磁盘上限，Aggregate这个性能是和业务统计关联性比较大性能不固定。

如何解决Doris写入慢的问题02

- 表存储方式：

和数据模型类似，主要是影响单block的写入性能，包括LTM和compaction的速度。

- streamload的数据块大小：

这个因素主要是和connector的参数相关，也就是可以调整connector的单个batch的buffer大小和攒批的时间来调整数据块大小。

- 硬件（网络、IO）：

硬性这个就是可以通过提升硬件来加速的。

综合上面的因素考虑，当前场景下Tablet数量是100，数据模式因存在更新需要Unique，表存储方式选择MoR这个三者都是已经确定只需按需调整即可。当前剩余streamload的参数和硬件影响未明确，因此我们对这两个方面做了相关实验。

StreamLoad的Batch实验

测试环境

使用FlinkCDC导入Oracle的10个Instance的数据，数据量300G+。

测试环境

- 在客户端，通过stream load在不同批次大小的方式下进行压测。
- 使用每批2w的数据量，分别从两个全量和增量维度，做不同tablet规模下的压测。

测试结论：

- sl每批条数调大后，单次sl的耗时其实是基本一致的，但flink cdc的同步性能得到巨大提升。
- 降低tablet数可以较大提升写性能，但是会影响查询效率。

序号	sl每批条数	攒批时间（单位：s）	sl单次耗时（单位：ms）	作业运行时长	同步数据量
1	1w	30	7356	15m23s	4.94G
2	2w	30	8870	15m40s	7.54G
3	5w	30	7408	15m10s	18.15G

全量同步：

序号	tablet数	sl条数	攒批时间（单位：s）	sl耗时（ms）	作业运行时长	同步数据量
1	100	20000	30	11710	20m26s	14.88G
2	50	20000	30	3556	20m28s	23.04G
3	20	20000	30	1845	23m10s	43.8G

增量同步：

序号	tablet数	sl条数	攒批时间（单位：s）	sl耗时（ms）	作业运行时长	同步数据量
1	100	20000	30	5440	20m36s	8.6G
2	50	20000	30	2888	20m12s	11.98G
3	20	20000	30	1904	20m56s	15.76G
4	10	20000	30	1783	20m59s	12.68G

HDD磁盘的实验

测试环境

机器配置： CPU 96 内存：128G

磁盘： 1块8T的sata盘

测试方案

- 通过相同的数据量不同数据块的大小测试磁盘在不同数据块情况下的性能差异。
- 通过fio对磁盘进行随机写和顺序写来测试两种情况下的磁盘的性能差异。

测试数据:

随机写

块大小	iops	吞吐量	await (ms)
4k	624	2.5M/s	228
16k	524	33.5M/s	337
20k	743	14.8M/s	277
64k	504	33M/s	304
128k	429	55M/s	310
218k	340	74M/s	3.0
1024k	254	260M/s	139
20m	239	238M/s	1241
100m	244	248M/s	1344

顺序写

块大小	iops	吞吐量	await
4k	44000	176M/s	0.79
16k	16500	264M/s	2.19
20k	13115	262M/s	2.53
64k	3600	261M/s	7.83
128k	1892	257M/s	1.55
1024k	254	260M/s	139
20m	257	263M/s	1040
100m	256	262M/s	984

测试结论:

- HDD磁盘随机写IPOS是有限的仅仅是几百，数据块大小对影响影响很大。
- HDD磁盘顺序写IPOS是可以达到万级的，块大小对吞吐的上限影响不大。
- HDD磁盘的吞吐上限大概在250M左右。

SSD磁盘的实验

测试环境

机器配置： CPU 96 内存：128G

磁盘： 1块普通SSD， 1块MVME磁盘

测试方案

- 通过fio对普通磁盘和MVME两种磁盘进行测试，评估普通SSD和MVME磁盘的性能差异。
- 通过相同的数据量不同数据块的大小测试磁盘在不同数据块情况下的性能差异。
- 通过fio对磁盘进行随机写和顺序写来测试两种情况下的磁盘的性能差异。

测试数据:

随机写性能数据:

块大小	IOPS	吞吐量	wait(ms)
4k	67321	269M	1.52
16k	24498	376M	5.57
64k	8073	516	15
128k	4525	523	31
1M	594	526	192
20M	26	534	100
100M	5	574	101

顺序写性能数据:

块大小	IOPS	吞吐量	wait(ms)
4k	129000	506	0.22
16k	32600	510	1
64k	8187	511	4.1
128k	4089	511	8.25
1M	485	511	65.91
20M	25	511	520
100M	5	511	505

MVME性能数据:

块大小	IOPS	吞吐量	wait(ms)
4K	259K	1011M	-
16K	147K	2304M	-
128K	34.6K	4321M	-
1M	4266	4277M	-

测试结论:

- 普通SSD 随机写和顺序写性能差不多 500MB。
- 普通的SSD在数据块很小< 4k, 性能衰减比较严重（250M左右）。
- MVME 貌似没有想象的那么强，极限吞吐在4G，IOPS在极限在25W。

快速写入的解决措施&效果

解决措施

从上面多个实验可以得出Doris写入的结论如下：

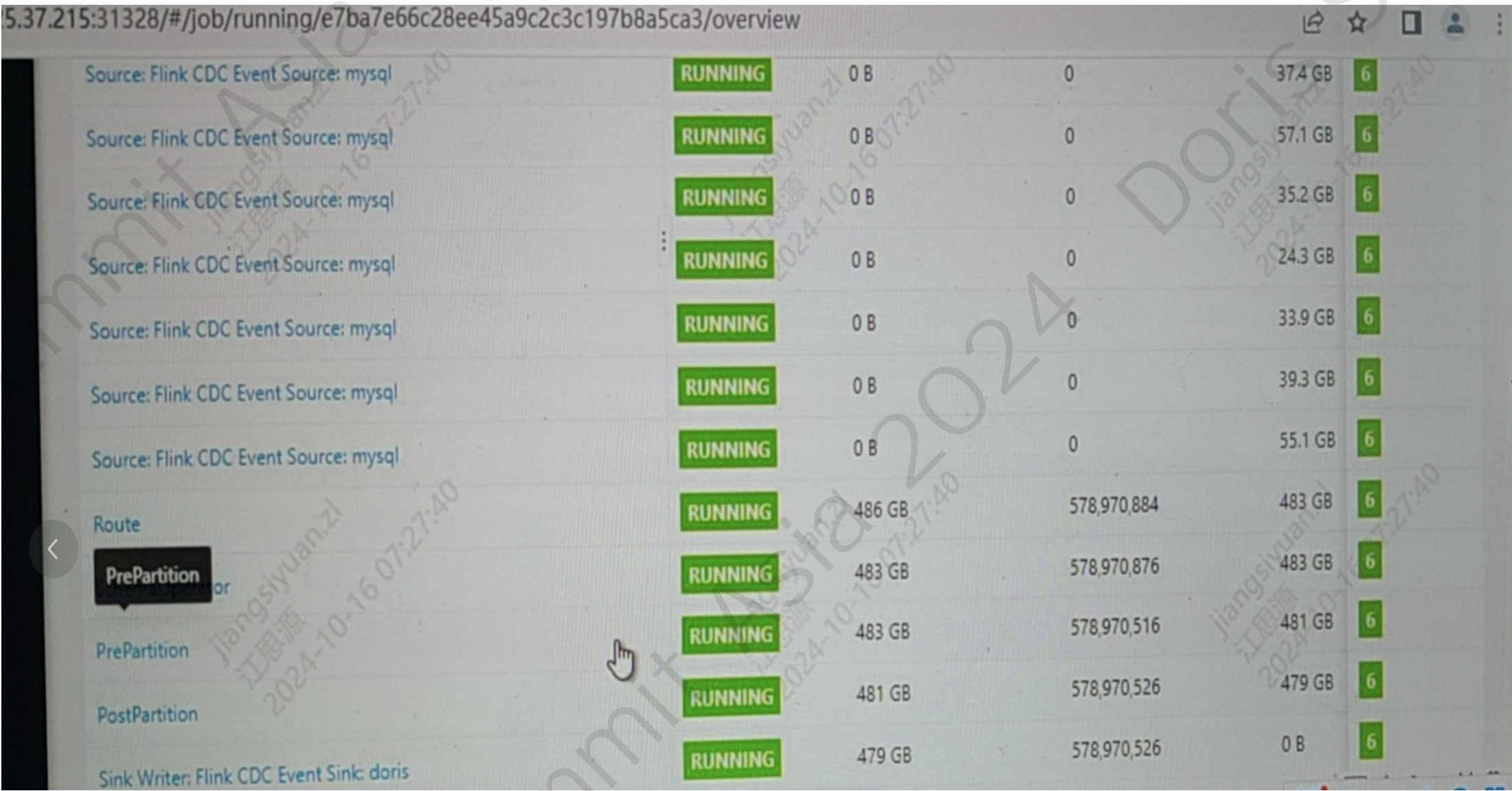
- 目前Doris的BE集群所使用的磁盘是HDD，不适合写小批数据（因为HDD在随机写场景下IOPS低）。
- 通过**攒大批**方式可以有效提高StreamLoad的当次写入的数据块大小进而提升写入性能。
- 通过**降低Tablet的数量**也能提高StreamLoad单次写入的数据块大小进而提升写入性能。
- 多cdc任务场景下，小任务之间会出现io竞争（IOPS有限），引起性能下降。

总体来说：需要从查询性能、写入性能成本三方面做平衡，当前问题解决方式有两种：

- 1.更改磁盘为SSD，会增加成本。
- 2.降低Tablet数，会影响查询效率。

效果：

6并行度，每个并行度2core:4G，1小时导入500G左右



The screenshot shows a Doris job overview page with the URL 5.37.215:31328/#/job/running/e7ba7e66c28ee45a9c2c3c197b8a5ca3/overview. It displays a table of running tasks. The first 6 tasks are 'Source: Flink CDC Event Source: mysql' and are in 'RUNNING' status. The last task is 'Sink Writer: Flink CDC Event Sink: doris' and is also in 'RUNNING' status. The table includes columns for task name, status, progress, and other metrics.

Task Name	Status	Progress	Other Metrics
Source: Flink CDC Event Source: mysql	RUNNING	0 B	0
Source: Flink CDC Event Source: mysql	RUNNING	0 B	0
Source: Flink CDC Event Source: mysql	RUNNING	0 B	0
Source: Flink CDC Event Source: mysql	RUNNING	0 B	0
Source: Flink CDC Event Source: mysql	RUNNING	0 B	0
Source: Flink CDC Event Source: mysql	RUNNING	0 B	0
Route	RUNNING	486 GB	578,970,884
PrePartition	RUNNING	483 GB	578,970,876
PrePartition	RUNNING	483 GB	578,970,516
PostPartition	RUNNING	481 GB	578,970,526
Sink Writer: Flink CDC Event Sink: doris	RUNNING	479 GB	578,970,526

04

展望未来

未来规划

存算分离

深入分析 Doris 存算分离架构利用存算分离解决业务场景中数据共享、冷热分离，资源隔离等难题

业务落地

积极推广 Doris 在数据分析、日志检索、即席查询等场景的落地。助力客户实现数字化转型的改革。

拥抱社区

积极参与社区共建工作，合作完善元数据、Catalog、CRC、存算分离等模块，推动 Doris 的发展。

国产化

继续优化 Doris 在 ARM 平台的性能问题，推动国产化进程



Thanks for Watching!