晋升答辩

P序列

姓名：胡云海

部门：基础架构部/实时平台

目录

自我介绍

上海幻电信息科技

ASperger

huyunhai

开发工程师

教育经历

- **学校**：山东大学（2022）
- **专业**：数学

工作经历

- **实习时间**：2021年10月
- **正式入职**：2022年7月
- **岗位**：实时团队-开发工程师
- **主要成果**：
  - Flink RocksDB State 监控补全
  - Flink RocksDB 接口层压缩
  - Flink Batch 流批一体&云原生

主要成果

- **过去一年的主要成绩**
  1. **Flink RocksDB State 监控补全**
  2. **Flink RocksDB 接口层压缩**
  3. **Flink Batch 流批一体&云原生**

主要成果—Flink RocksDB State 监控补全

项目的背景——RokcsDB 缺少有效且直观的监控

在 Flink 中 RocksDB 被用作 StateBackend 的底层存储，适合大状态的作业。但是，社区版本的 Flink 使用 RocksDBStateBackend 时缺少实时监控，如果遇到性能问题，基本上是很难判断出问题原因。因此为了能够更好的对 Flink 性能进行优化，完善 RocksDB 监控具有重要意义。

下图是字节跳动分享中展示的 RocksDB 监控平台：

主要成果—Flink RocksDB State 监控补全

痛点问题分析

- 缺少监控无法及时检测到 RocksDB 性能问题
- 缺少监控无法有效检测到 RocksDB 的性能瓶颈
- State 操作数据量大 监控数据采集处理压力大

解决方案

- 对不同State 进行完善的采集
- 采集必要且全面 的数据指标
- 设置合适的采样率

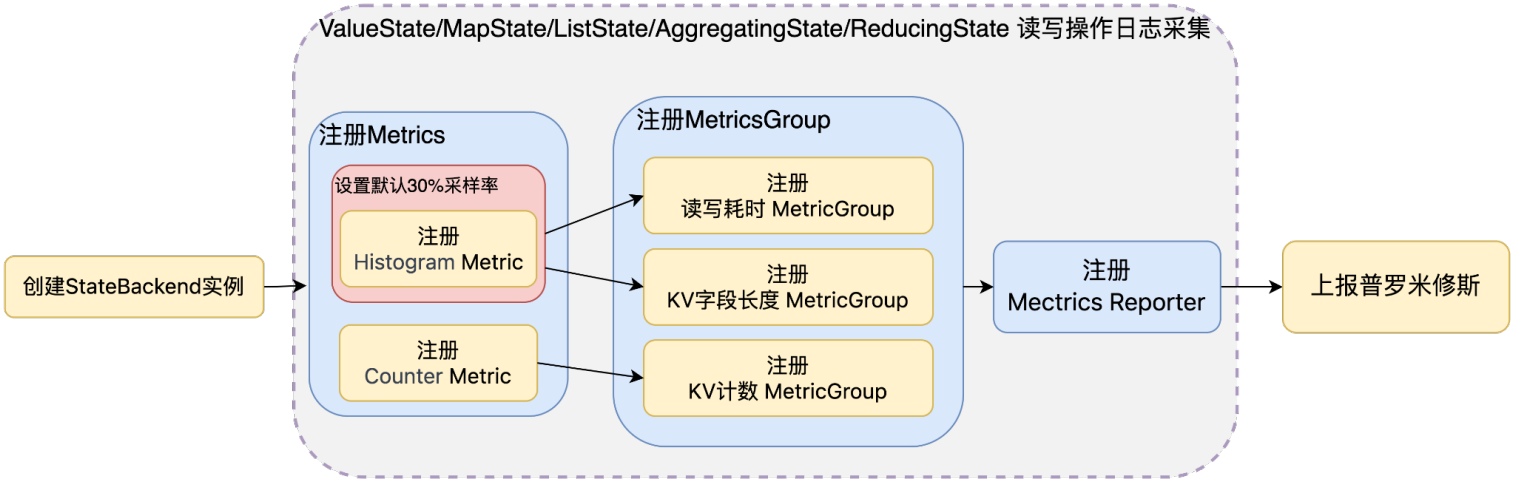
核心目标

- 实时监控
- 数据可视化
- 阈值和警报
- 全面监控

主要成果—Flink RocksDB State 监控补全

应对方法——监控采集架构设计

下图为针对上述痛点问题以及解决方案所设计的RocksDB StateBackend监控采集架构：



1. **数据量大**：**设置采样率**，防止数据过大打爆采集系统；而计数counter数据则可全量采集。
2. **多种监控指标**：**监控指标多样**，包括读写延迟、数据大小等,实现全面、准确的监控。

主要成果—Flink RocksDB State 监控补全

成果——提升开发运维效益

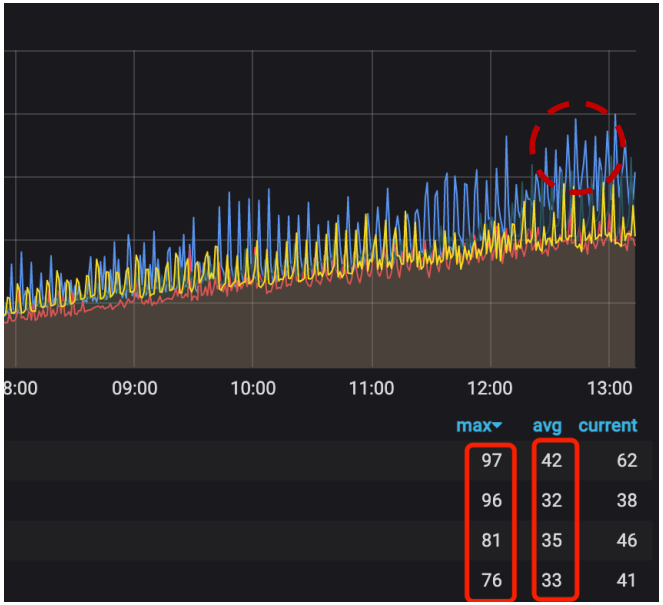
Flink RocksDB StateBackend监控实现了对Flink RocksDB实时的、全面的监控和警报，以确保系统的稳定性和性能，以下是具体成果：

1. **实时监控 & 数据可视化 &阈值和警报：**可以实时地监控Flink的RocksDB指标，在该监控系统的支持下，实时团队对线上任务出**RocksDB的性能问题的发现和排查速度从15min+提速到1min** 以内。
2. **全面性能监控：**监控RocksDB State 的关键指标，包括**读写耗时、KV字段长度和读写次数**，以提供全面的性能监控，全面的性能监控使后续RocksDB一系列调优提供了提供了基础支持。

主要成果—Flink RocksDB 接口层压缩

项目的背景——RokcsDB 性能调优

在实践过程和测试实验中观察到的大State任务在开启底层压缩总是会有比较大的cpu毛刺现象，火焰图确认是 RocksDB 引起的。同时如果完全关闭压缩会导致State文件膨胀数倍且在高峰期出现更为严重的毛刺。



主要成果—Flink RocksDB 接口层压缩

痛点问题分析

开启RocksDB压缩
CPU毛刺现象严重

关闭压缩State膨胀

压缩算法的选择
压缩后数据一致性问题

解决方案

对接口层字段进行压缩

对比不同的压缩算法

针对不同State
设计压缩解压缩算法

核心目标

提升CPU利用率

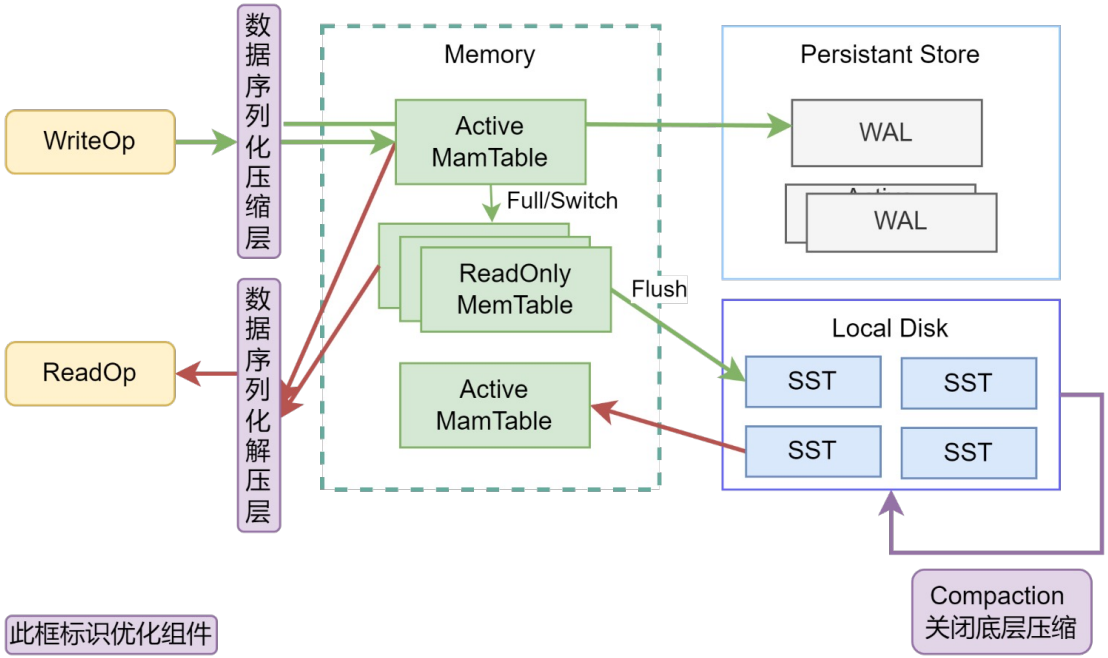
减少CPU毛刺

降低State大小

主要成果—Flink RocksDB 接口层压缩

应对方法——RocksDB 业务层压缩架构设计

针对上述问题，设计了如下图所示的RocksDB 业务层压缩架构，**在每次读写操作时对业务字段进行压缩/解压缩并且关闭底层压缩**，以获取较好的平衡节省磁盘内存和提高CPU利用率的需求。

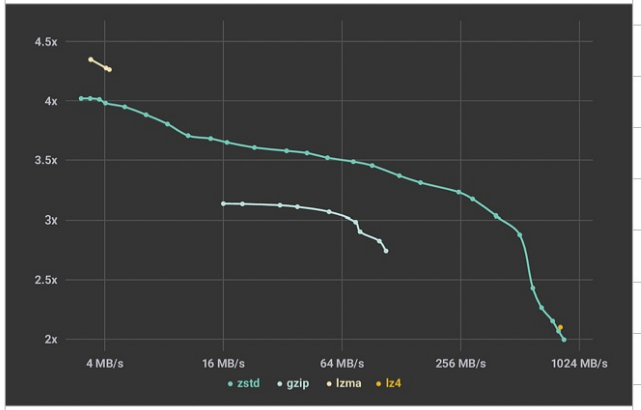


主要成果—Flink RocksDB 接口层压缩

应对方法——优秀的压缩算法选型

在实际使用中，需要根据实际业务需求选择合适的压缩算法，对比多种压缩算法包括LZ4、Snappy和ZSTD等的性能，在压缩比和压缩性能消耗的平衡下选择了综合性能最具优势的ZSTD，并且在实际测试中ZSTD实证存在综合显著优势。

zstd 与 gzip lzma lz4 对比

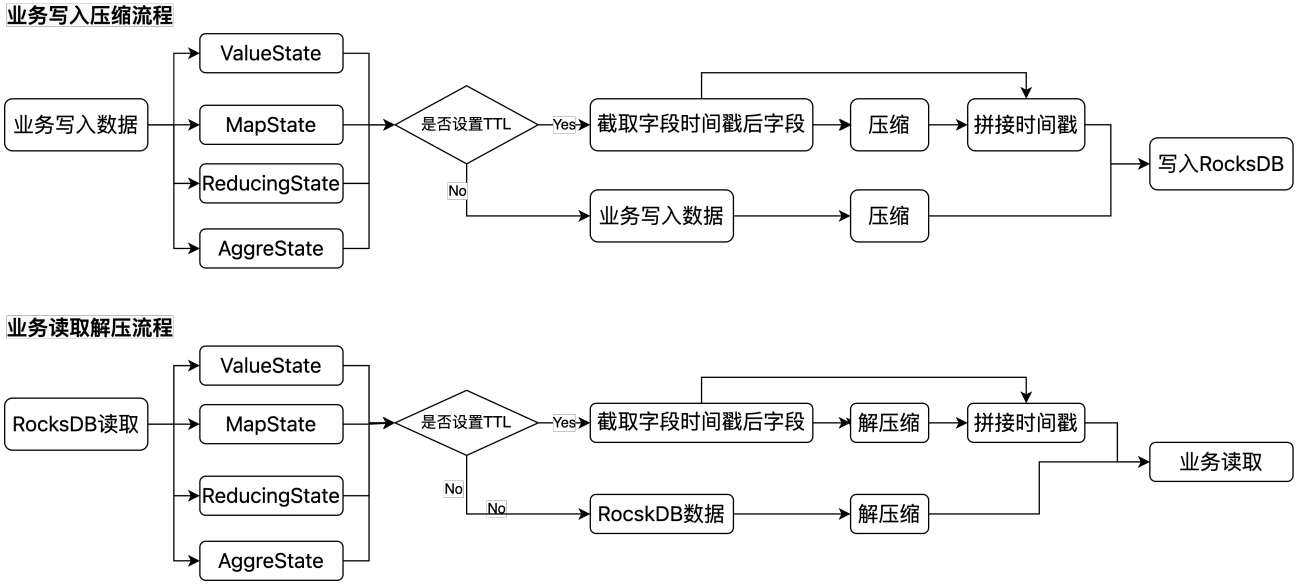


Compressor name	Ratio	Compression	Decompress
Zstd 1.4.5 -1	2.884	500 MB/s	1660 MB/s
zlib 1.2.11 -1	2.743	90 MB/s	400 MB/s
brotli 1.0.7 -0	2.703	400 MB/s	450 MB/s
Zstd 1.4.5 --fast=1	2.434	570 MB/s	2200 MB/s
Zstd 1.4.5 --fast=3	2.312	640 MB/s	2300 MB/s
Zstd 1.4.5 --fast=5	2.178	700 MB/s	2420 MB/s
lzo1x 2.10 -1	2.106	690 MB/s	820 MB/s
lz4 1.9.2	2.101	740 MB/s	4530 MB/s
snappy 1.1.8	2.073	560 MB/s	1790 MB/s

主要成果—Flink RocksDB 接口层压缩

应对方法——自定义的压缩流程确保数据一致性

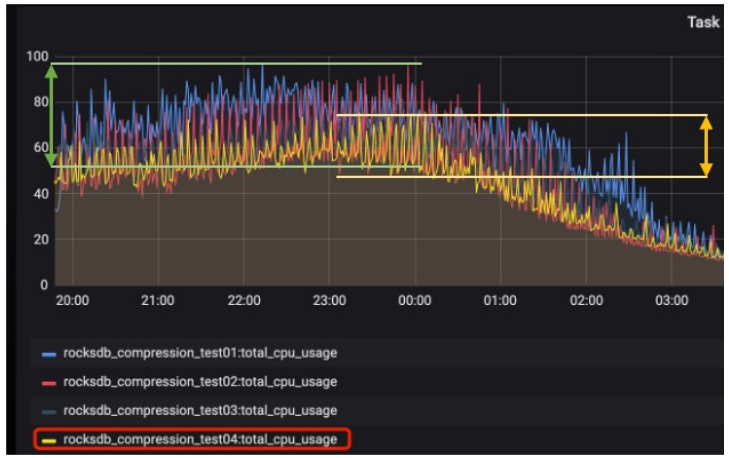
Flink State的种类较多，在读写数据时，需要保证数据的正确性和一致性。采用自定义的压缩流程。在业务数据压缩中，我们对几种主要的State设计了针对性的流程优化，保证数据的一致性和正确性。



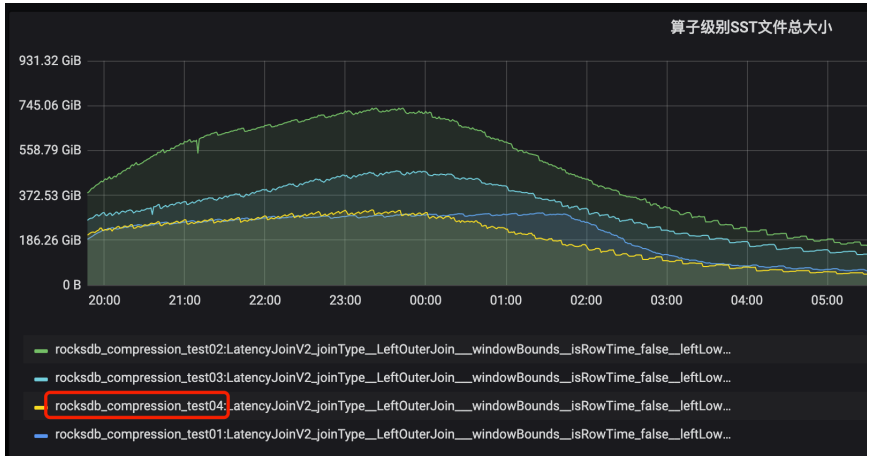
主要成果—Flink RocksDB 接口层压缩

成果——最终实现有效的性能提升

CPU毛刺 & 使用均值降低20%



State文件压缩率达到近 60%



test01：state接口层不压缩，rocksdb底层默认压缩（lz4）

test02：state接口层不压缩，rocksdb底层不压缩

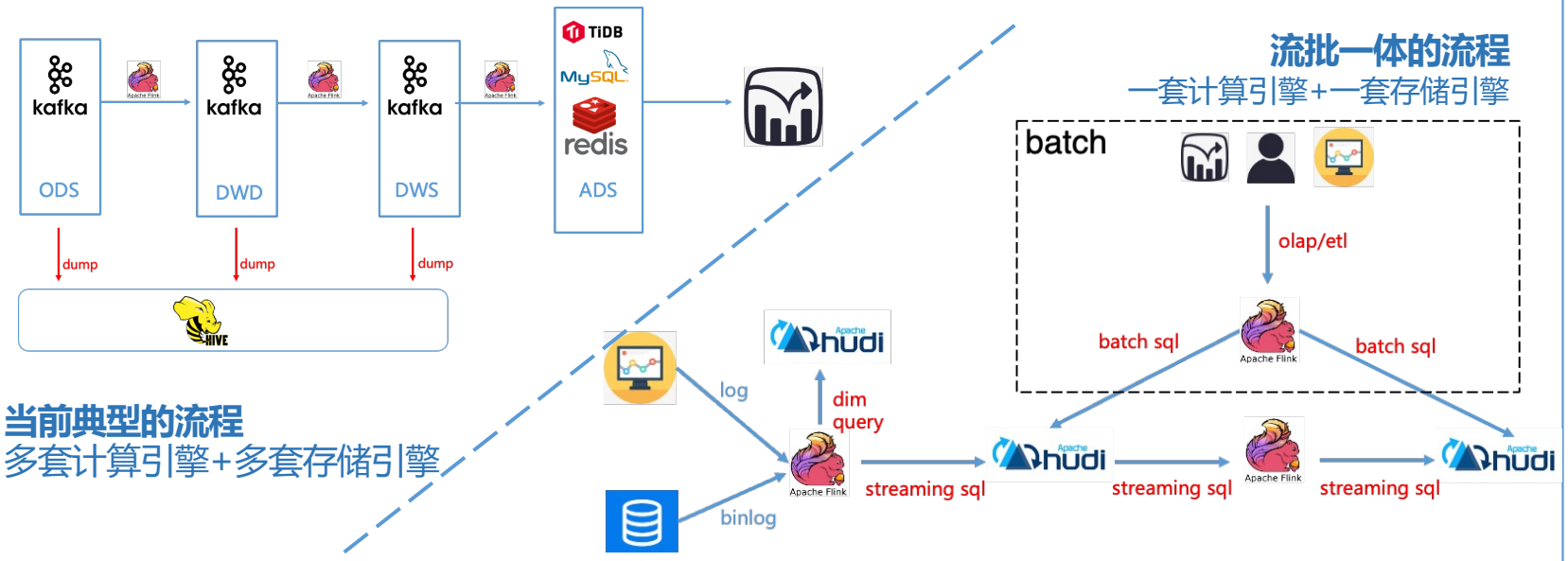
test03：大字段压缩，rocksdb底层不压缩 ，读写rocksdb前后，直接在大字段上套zstd压缩/解压缩的udf来实现。

test04：state接口层压缩，rocksdb底层不压缩，这是test03的进阶，对于label join等场景，存储的是row，字段级别的压缩效率不高，state接口层可以做到更充分的压缩。

主要成果—Flink 流批一体&云原生

项目背景——流批一体增益降本增效

在新的流批一体架构下，只需要一套计算引擎+一套存储引擎即可，学习和维护成本大大降低，统一带来的一些隐性价值也难以估量



主要成果—Flink 流批一体&云原生

痛点问题分析

- 流批两套计算调度系统
- 流批多套存储系统
- 业务方需要开发流批两套任务

解决方案

- 构建一个可靠、高效的 Flink 流批一体计算引擎
- 实现并优化 Flink on Kubernetes
- 设计和构建一套高效的开发协同流程

核心目标

降本增效

提高资源利用率

降低开发成本

主要成果—Flink 流批一体&云原生

应对方法与成果——可靠、高效的 Flink 流批一体计算引擎优化

稳定性调优：调优后实现与传统批计算引擎的近似稳定性能表现（测试任务中 **Flink 2.8H:Spark 3H**）；支持remote shuffle技术，使Flink 批计算可以处理数据量高达**100T及以上**数据的任务 。

流批一体兼容性：批计算**完全兼容**流计算/Hive UDF/UDTF ，可以有效支持流批一体，业务方批计算任务和流计算任务**SQL相似度达98%，实现广告部门原有流计算任务一天转换批计算上线的速度。**

批计算SQL

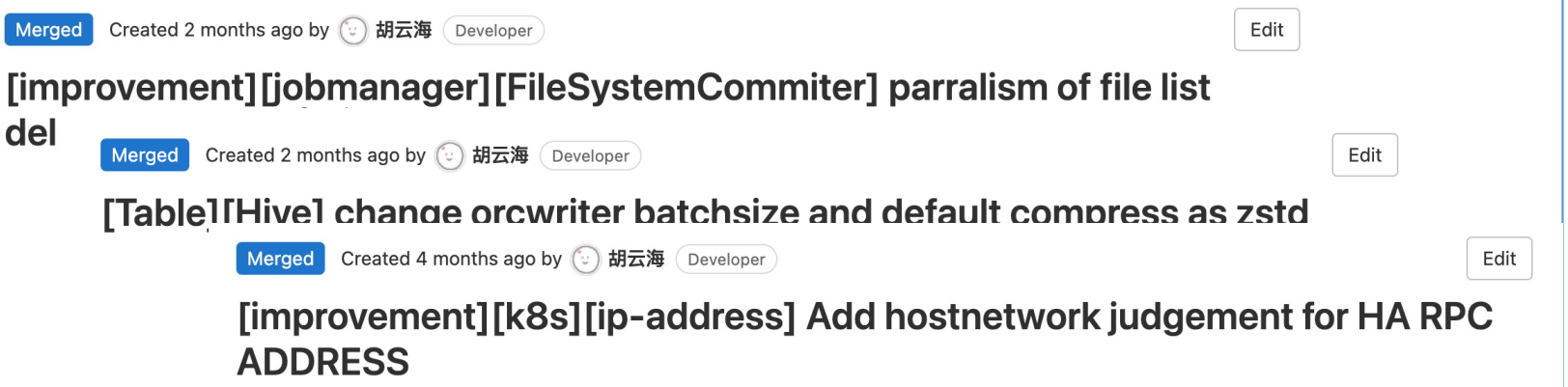
流计算SQL

主要成果—Flink 流批一体&云原生

应对方法与成果——可靠、高效的 Flink 流批一体计算引擎优化

性能优化：社区版Flink的批计算引擎中存在不少欠考虑的细节，我们对此进行排查和优化。

1. JobManager在任务结束后对HDFS文件使用单线程处理，导致JM在测试中会在任务结束后无响应较长时间。因此对其多线程化，**处理时间从20min降低到5min以内，提升了400%的性能。**
2. Hive在sink时的位置batch size不合理导致TM容易在sink阶段OOM，通过反射调整参数并且使用ZSTD压缩字段，**节省了近50%的内存消耗。**
3. Flink on K8S的实际应用中存在一系列问题并对其进行**持续的优化。**

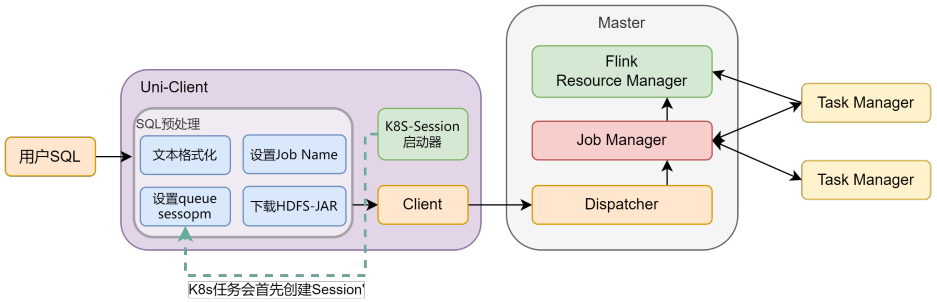


主要成果—Flink 流批一体&云原生

应对方法与成果——可靠、高效的 Flink 流批一体计算引擎优化

批任务提交架构设计：设计一套能够支持 **Yarn 与 K8S** 的任务提交框架，支持如：指定任务名，提交队列，支持远程Jar包的调用以及任务运行状态监控等功能。

Flink Batch 交付使用：2023 H1 Flink Batch 已经正式在公司离线平台 **Archer 上线** 提供用户使用，并且也**交付了AI部门** 一套完整的Flink Batch on K8S的方案。

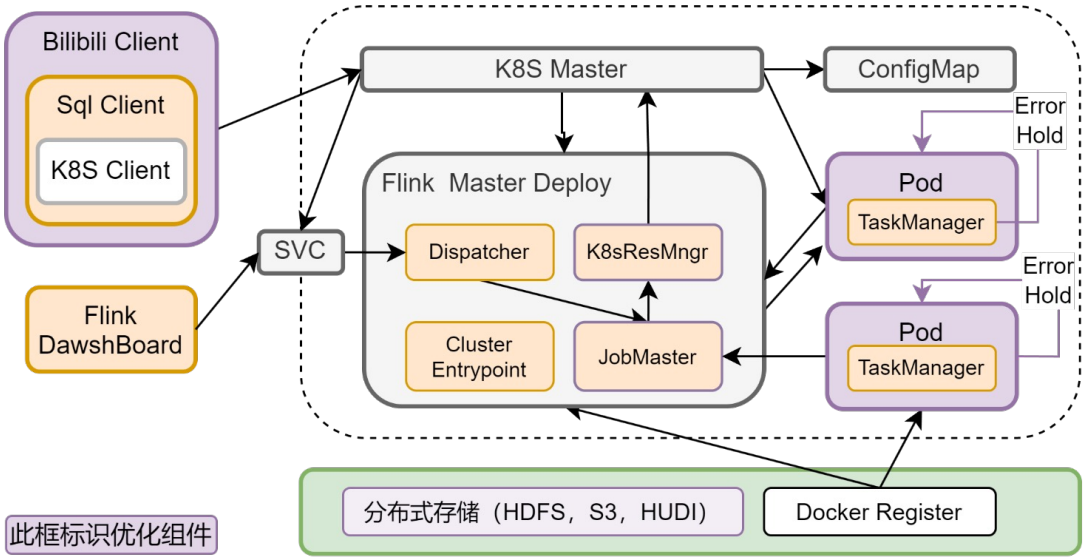


主要成果—Flink 流批一体&云原生

应对方法与成果——实现 Flink on Kubernetes

在使用社区版Flink on K8S 的过程中，发现了一些问题以及功能性不足，因此在其基础上进行了优化设计。

Flink on K8S 优化后架构图：



设计要点：

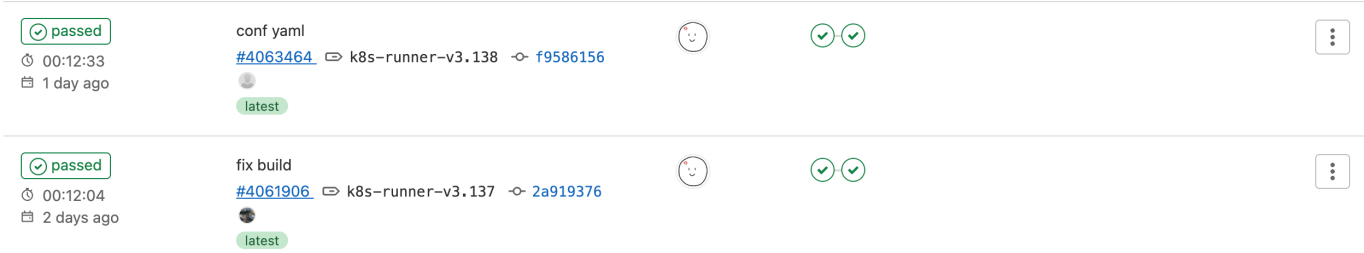
- 实现了自定义的**Bilibili Client作为任务提交入口**；
- Flink Master可以按配置**设置Pod的个性化参数，**包括磁盘挂载、标记任务名等；
- Pod在任务出错时可以设定清除阻滞时间，给开发者**保留更多日志**；
- 逐步支持HUDI 数据源；
- **解决部分Flink社区代码存在的Bug，**如K8S存在LocalHost 绑定错误等。

主要成果—Flink 流批一体&云原生

应对方法与成果——Flink 流批一体开发辅助工具

Flink Batch正式上线后，发现流批出现两套 Flink 分支，导致迭代时出现管理上的困难和低效，因此构建了一套适合目前F流批一体的代码打包流程和更新流程，实现了：

1. 支持GitLab完整、可靠的CICD流程，且K8S支持提交镜像和运行镜像的**同步迭代**。
2. 通过info文档记录和CICD的分支信息，比手动打包上传节省了**80%**的时间，为测试提供了有效的技术保障。
3. 并且CICD记录避免出现镜像版本混乱和丢失的问题，效率提升了***100%***



自我评估

➢ 成长与收获

在我从事Flink相关的开发的工作中，我取得了十足的成长和收获：

1. **技术能力提升：**通过不断学习和实践逐渐掌握了Flink的使用和优化技巧，提高了自己的技术能力。
2. **解决问题的能力提高：**在实践中我逐渐掌握了解决问题的方法技巧，提高了自己的解决问题的能力。
3. **团队合作能力提高：**在优化Flink作业的过程中，提高了自己的沟通和协作能力，更好地合作和协调工作。
4. **自我学习能力提高：**不断地学习和尝试，逐渐提高了自己的自我学习能力，更好地应对复杂问题和挑战。

自我评估

➢ 请阐述自己需要提高和努力（未来待发展）的领域/方面

1. **技术能力的提高**：不断提升自己的技术能力，更好地解决复杂的技术问题，提升自己的技术竞争力。
2. **团队协作和沟通的能力提高**：提高自己的沟通和协作能力，合理分配和协调工作等方面的能力。
3. **业务和行业知识的提高**：深入了解业务和行业知识，更好地了解公司的战略和发展方向。

晋升后岗位职责与目标

➢ 晋升后的岗位职责和目标

1. **你认为晋升后的岗位职责与目前的岗位职责相比，有什么变化?**

   随着晋升，我的岗位职责将会更加重要和复杂。相比于现在，我将会负责更多的开发任务，并且需要承担更多的技术方面的职责，包括技术选型、架构设计和性能优化等方面的工作。

2. **你对晋升后岗位的构想和规划是什么，希望在新岗位实现什么目标?**

   - **提升技术水平**：希望能够持续提升技术水平，成为一个更加专业和有能力的开发工程师。
   - **推动项目进展**：负责更多的开发任务，更好的推动项目的进展。
   - **提升团队效率**：提升个人的工作效率和质量，优化开发流程，使工作更加高效和顺畅。

建议和意见

- 建议我们可以定期举办跨部门的会议，分享每个部门的工作进展，互相学习，为公司的整体发展注入新的动力。
- 此外，我们也可以定期组织员工活动，比如下午茶、运动会、团队建设等，这有助于加强团队凝聚力和员工之间的交流，同时也能够缓解工作压力，提高员工的工作积极性和生产力。
- 最后，我还建议公司可以鼓励员工参加行业内的技术交流活动和会议，以扩大员工的技术视野和提高技术能力，进而促进公司技术的发展。

Thanks