阿里巴巴大规模应用 Flink 的实战经验：常见问题诊断思路

整理：张宋庆（Flink 社区志愿者）
校对：李庆（Flink 社区志愿者）
作者：杨阳（时溪），阿里巴巴技术专家

> 摘要：本文由阿里巴巴高级运维工程师杨阳（时溪）分享，主要介绍阿里巴巴常见问题诊断模块与思路，内容涵盖以下几个方面：
>
> - 常见运维问题
> - 问题处理方式
> - 作业生命周期
> - 工具化经验
>
> Tips：点击「阅读原文」链接可查看作者原版 PPT 及分享视频～

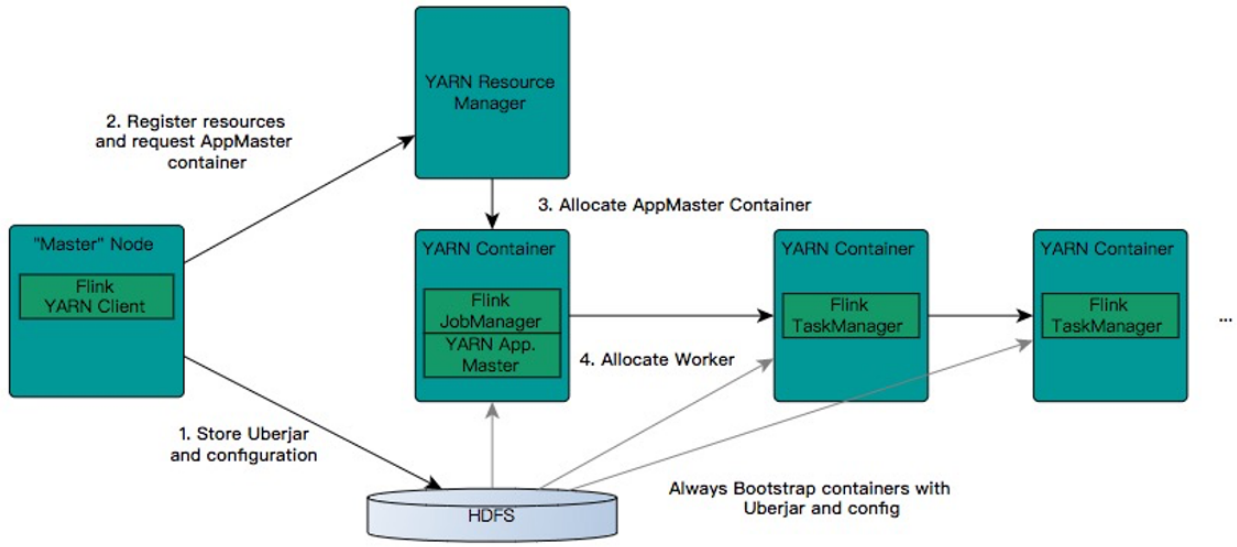
1.常见运维问题

1.1 作业运行环境

本文中介绍的作业运行环境主要是在阿里巴巴集团内，构建在 Hadoop 生态之上的 Flink 集群，包含 Yarn、HDFS、ZK 等组件；作业提交模式采用 yarn per-job Detached 模式。

作业运行环境

Yarn Per-Job Detached模式



- 第1步，作业提交是通过 Flink Yarn Client，将用户所写的作业代码以及编译好的 jar 包上传到 HDFS 上；
- 第2步 Flink Client 与 Yarn ResourceManager 进行通信，申请所需要的的 Container 资源；
- 第3步，ResourceManager 收到请求后会在集群中的 NodeManager 分配启动 AppMaster 的 Container 进程，AppMaster 中包含 Flink JobManager 模块和 Yarn 通信的 ResourceManager 模块；
- 第4步，在 JobManager 中根据作业的 JobGraph 生成 Execution Graph，ResourceManager 模块向 Yarn 的 ResourceManager 通信，申请 TaskManager 需要的 container 资源，这些 container 由 Yarn 的 NodeManger 负责拉起。每个 NodeManager 从 HDFS 上下载资源，启动 Container(TaskManager)，并向 JobManager 注册；JobManger 会部署不同的 task 任务到各个 TaskManager 中执行。

■ 资源申请方式

1. 指定资源大小

提交时，指定每个 TaskManager、JobManager 使用多少内存，CPU 资源。

2. 细粒度资源控制

阿里巴巴集团内主要采用 ResourceSpec 方式指定每个 Operator 所需的资源大小，依据 task 的并发聚合成 container 资源向 Yarn 申请。

■ 环境高可用

1. **JM 高可用**，AppMaster(JobManager) 异常后，可以通过 Yarn 的 APP attempt 与 ZooKeeper 机制来保证高可用；
2. **数据高可用**，作业做 checkpoint 时，TaskManager 优先写本地磁盘，同时异步写到 HDFS；当作业再次启动时可以从 HDFS 上恢复到上次 checkpoint 的点位继续作业流程。

1.2 为什么我的作业延时了？

■ 时间类型

- **Processing time**

  Processing time 是指 task 处理数据时所在机器的系统时间

- **Event time**

  Event time 是指数据当中某一数据列的时间

- **Ingestion time**

  Ingestion time 是指在 flink source 节点收到这条数据时的系统系统时间

■ 延时定义

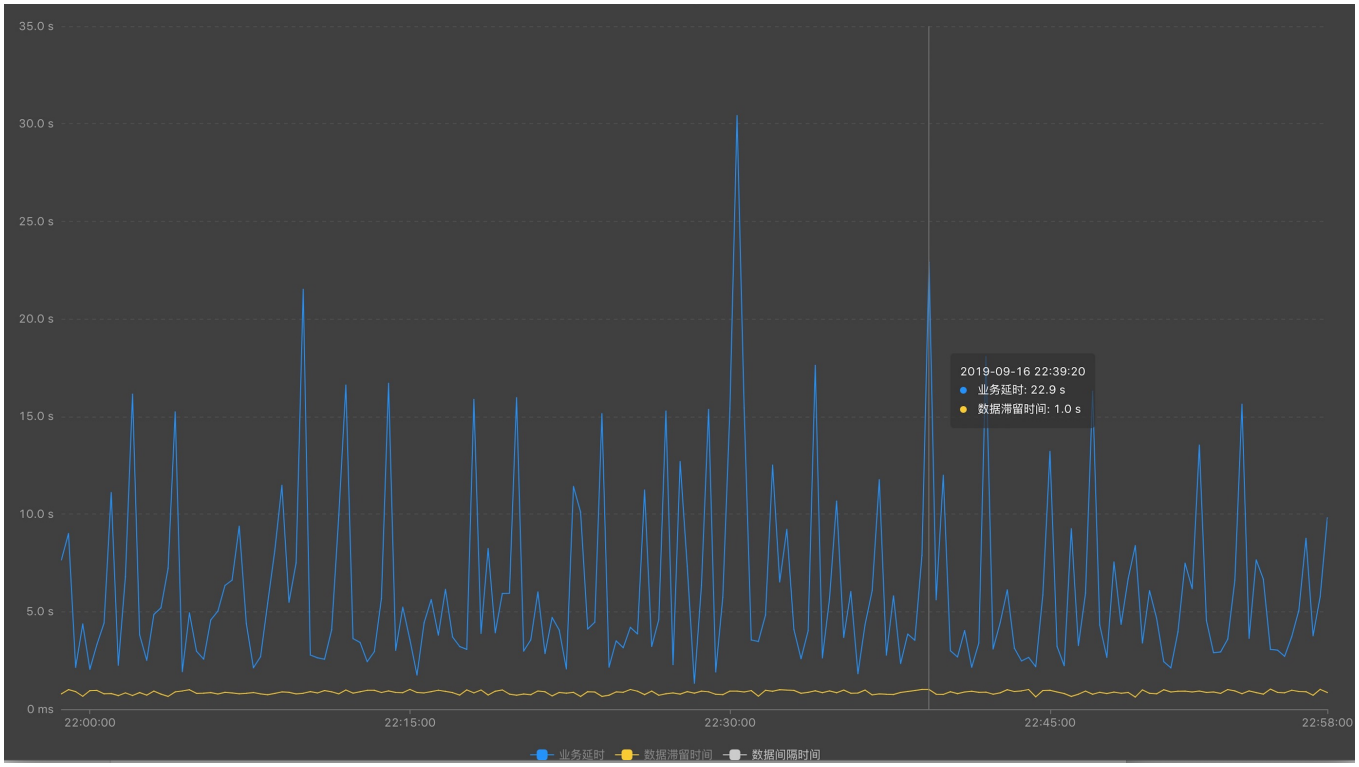
自定义 Source 源解析中加入 Gauge 类型指标埋点，汇报如下指标：

1. 记录最新的一条数据中的 event time，在汇报指标时使用当前系统时间 - event time。
2. 记录读取到数据的系统时间-数据中的 event time，直接汇报差值。

delay = 当前系统时间 – 数据事件时间(event time)
说明：反应处理数据的进度情况。

fetch_delay = 读取到数据的系统时间- 数据事件时间(event time)
说明：反应实时计算的实际处理能力。

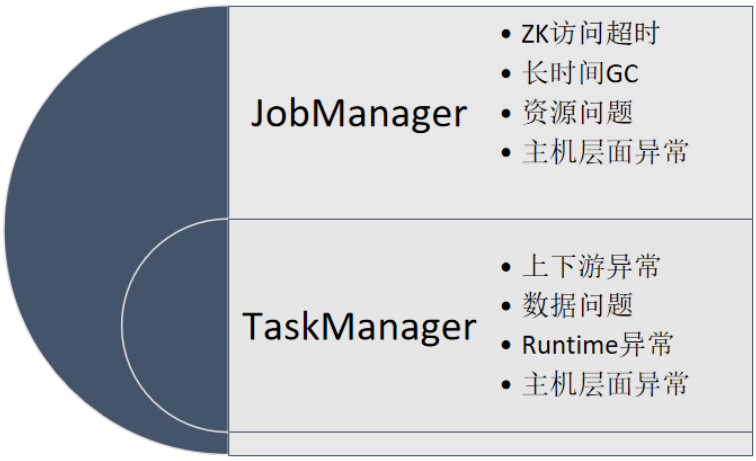
■ 延时分析



- 从上游源头，查看每个源头并发情况
- 是否上游数据稀疏导致
- 作业性能问题

1.3 为什么我的作业 failover 了?

■ 作业 failover 主要分为两大类



Flink Failover 主要有两类，一类是 Job Manager 的 Failover，还有一类是 Task Manager 的 Failover。

1.4 作业无法提交、异常停止

■ 无法提交

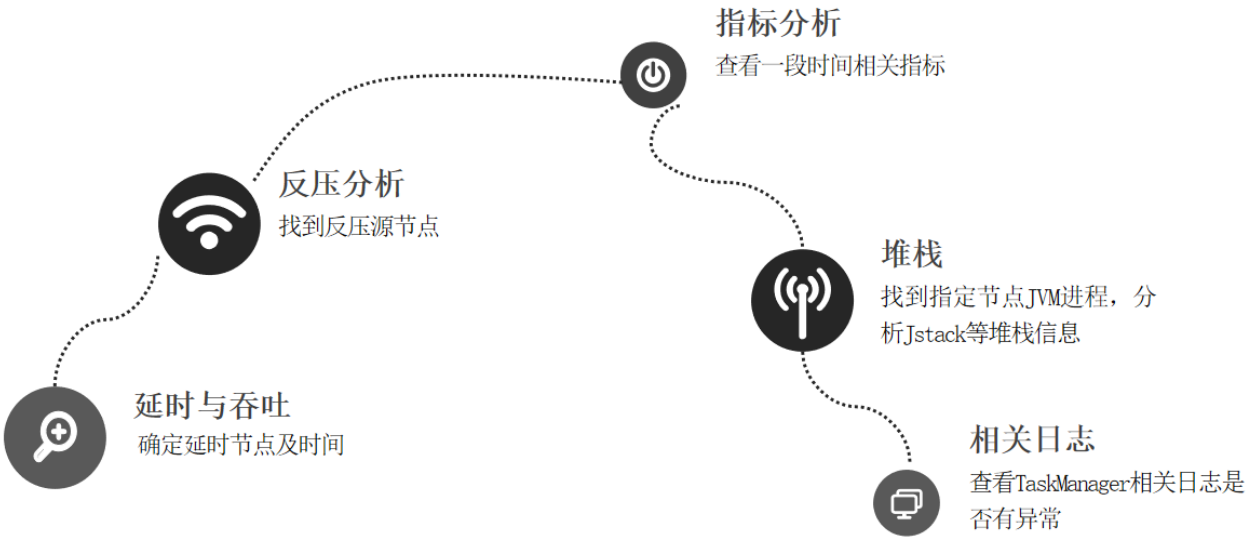
- Yarn 问题 – 资源限制
- HDFS 问题 - Jar 包过大，HDFS 异常
- JobManager 资源不足，无法响应 TM 注册
- TaskManager 启动过程中异常

■ 异常停止-指标监控无法覆盖

- 重启策略配置错误
- 重启次数达到上限

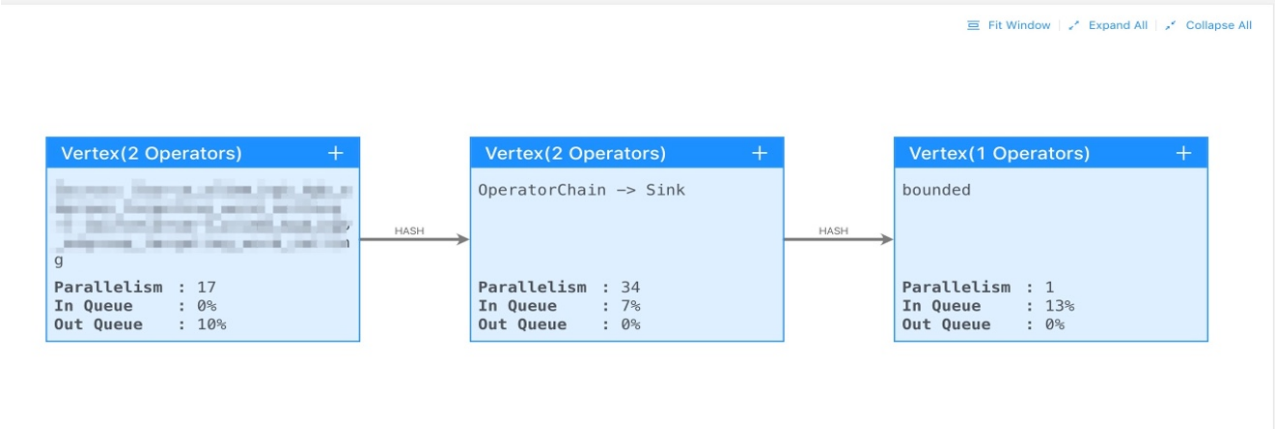
2.处理方式

2.1 延时问题处理方式



- 通过 delay、fetch_delay 判断是否上游稀疏导致延时或者作业性能不足导致延时
- 确定延时后，通过反压分析，找到反压节点
- 分析反压节点指标参数
- 通过分析 JVM 进程或者堆栈信息
- 通过查看 TaskManager 等日志

■ 延时与吞吐

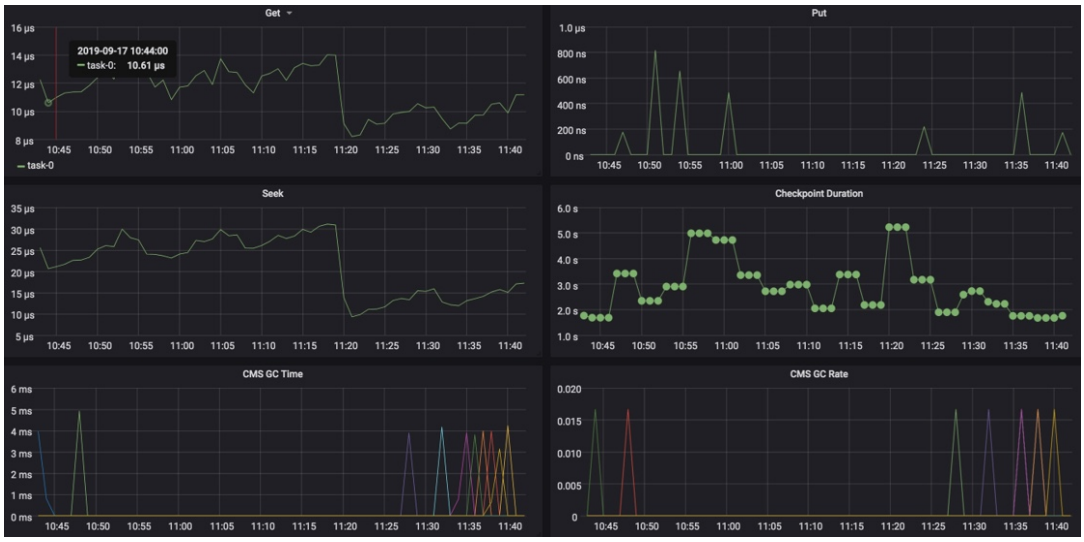


观察延时与 tps 指标之间关联，是否由于 tps 的异常增高，导致作业性能不足延时

■ 反压

- 找到反压的源头。
- 节点之间的数据传输方式 shuffle/rebalance/hash。
- 节点各并发的吞吐情况，反压是不是由于数据倾斜导致。
- 业务逻辑，是否有正则，外部系统访问等。IO/CPU 瓶颈，导致节点的性能不足。

■ 指标



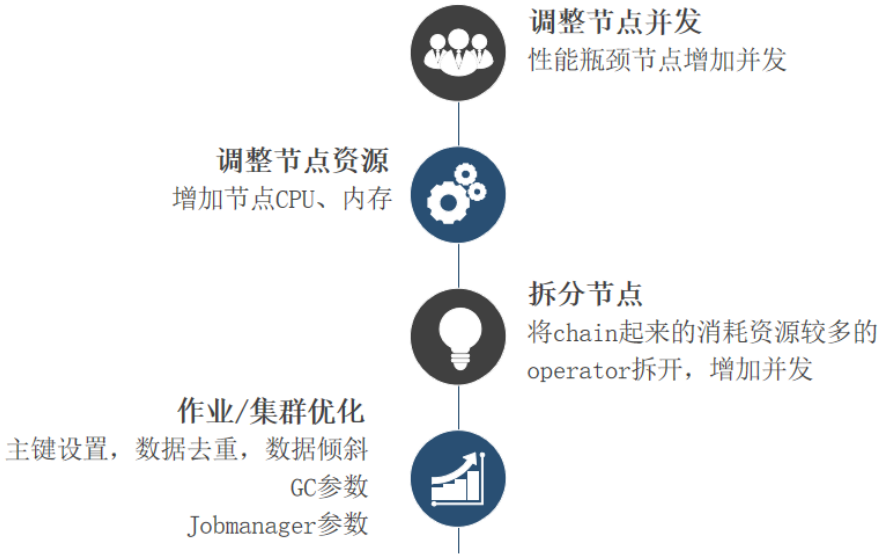
- GC 耗时多长
- 短时间内多次 GC
- state 本地磁盘的 IO 情况
- 外部系统访问延时等等

■ 堆栈

在 TaskManager 所在节点,查看线程 TID、CPU 使用情况，确定是 CPU，还是 IO 问题。

```
ps H -p ${javapid} -o user,pid,ppid,tid,time,%cpu,cmd

#转换为16进制后查看tid具体堆栈
jstack ${javapid} > jstack.log
```

■ 常见处理方式



1. 增加反压节点的并发数。

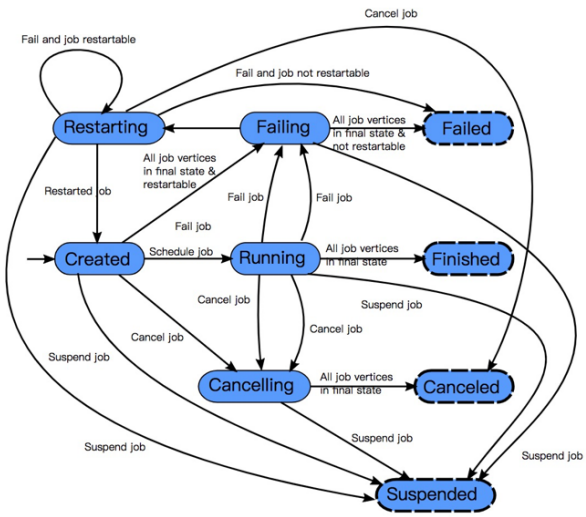
2. 调整节点资源，增加 CPU，内存。
3. 拆分节点，将 chain 起来的消耗资源较多的 operator 拆分。
4. 作业或集群优化，通过主键打散，数据去重，数据倾斜，GC 参数，Jobmanager 参数等方式调优。

2.2 作业 failover 分析

- 查看作业 failover 时打印的一些日志信息
- 查看 failover 的 Subtask 找到所在 Taskmanager 节点
- 结合 Job/Taskmanager 等日志信息
- 结合 Yarn 和 OS 等相关日志

3.作业生命周期

3.1 作业状态变化-JobStatus



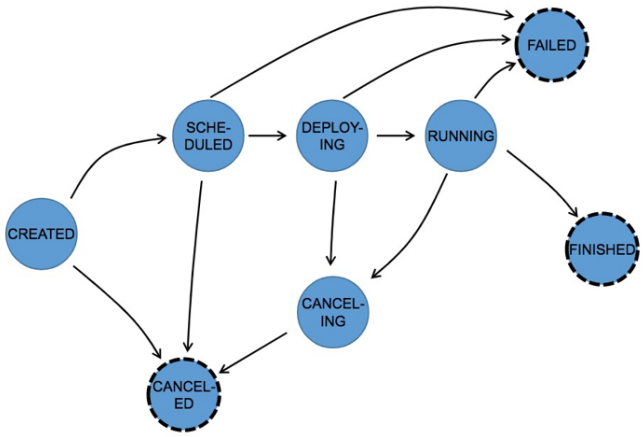

```java
/**
 * Possible states of a job once it has been accepted by the job manager.
 */
public enum JobStatus {

    /** Job is newly created, no task has started to run. */
    CREATED(TerminalState.NON_TERMINAL),

    /** Some tasks are scheduled or running, some may be pending, some may be finished. */
    RUNNING(TerminalState.NON_TERMINAL),

    /** The job has failed and is currently waiting for the cleanup to complete. */
    FAILING(TerminalState.NON_TERMINAL),

    /** The job has failed with a non-recoverable task failure. */
    FAILED(TerminalState.GLOBALLY),

    /** Job is being cancelled. */
    CANCELLING(TerminalState.NON_TERMINAL),

    /** Job has been cancelled. */
    CANCELED(TerminalState.GLOBALLY),

    /** All of the job's tasks have successfully finished. */
    FINISHED(TerminalState.GLOBALLY),

    /** The job is currently undergoing a reset and total restart. */
    RESTARTING(TerminalState.NON_TERMINAL),

    /**
     * The job has been suspended which means that it has been stopped but not been removed from a
     * potential HA job store.
     */
    SUSPENDED(TerminalState.LOCALLY),

    /** The job is currently reconciling and waits for task execution report to recover state. */
    RECONCILING(TerminalState.NON_TERMINAL);

    // ------------------------------------------------------------------------
```

上图中可以看到作业的整个状态转换。从作业创建、到运行、失败，重启，成功等整个生命周期。

这里需要注意的是 reconciling 的状态，这个状态表示 yarn 中 AppMaster 重新启动，恢复其中的 JobManager

模块，这个作业会从 created 进入到 reconciling 的状态，等待其他 Taskmanager 汇报，恢复 JobManager 的 failover，然后从 reconciling 再到正常 running。

3.2 Task 状态变化 -ExecutionState



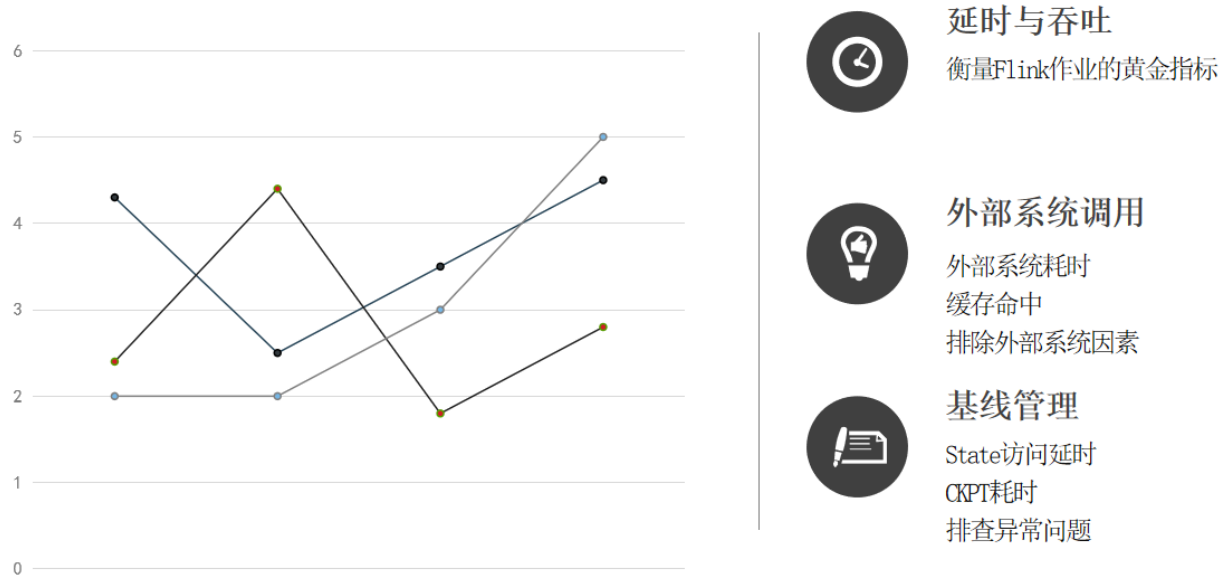

```
/**
 * An enumeration of all states that a task can be in during its execution.
 * Tasks usually start in the state {@code CREATED} and switch states according to
 * this diagram:
 * <pre>{@code
 *
 *     CREATED  -> SCHEDULED -> DEPLOYING -> RUNNING -> FINISHED
 *        |            |            |          |
 *        |            |            |   +------+
 *        |            |            |   V  V
 *        |            |            CANCELLING -----+----> CANCELED
 *        |            |                           |
 *        |            +---------------------------+
 *        |
 *        |                                  ... -> FAILED
 *        V
 *    RECONCILING  -> RUNNING | FINISHED | CANCELED | FAILED
 *
 * }</pre>
 *
 * <p>It is possible to enter the {@code RECONCILING} state from {@code CREATED}
 * state if job manager fail over, and the {@code RECONCILING} state can switch into
 * any existing task state.
 *
 * <p>It is possible to enter the {@code FAILED} state from any other state.
 *
 * <p>The states {@code FINISHED}, {@code CANCELED}, and {@code FAILED} are
 * considered terminal states.
 */
```

上图是作业的 Task 状态转换，需要注意的是，作业状态处于 running 状态时，并不意味着作业一定在运行消费信息。在流式计算中只有等所有的 task 都在 running 时，作业才算真正运行。

通过记录作业各个阶段的状态变化，形成生命周期，我们能很清楚地展示作业是什么时候开始运行、什么时候失败，以及 taskmanager failover 等关键事件，进一步能分析出集群中有多少个作业正在运行，形成 SLA 标准。

4.工具化经验

4.1 指标



如何去衡量一个作业是否正常?

- **延时与吞吐**

对于 Flink 作业来说，最关键的指标就是延时和吞吐。在多少 TPS 水位的情况下，作业才会开始延时.

- **外部系统调用**
  从指标上还可以建立对外部系统调用的耗时统计，比如说维表 join，sink 写入到外部系统需要消耗多少时间，有助于我们排除外部的一些系统异常的一些因素。
- **基线管理**
  建立指标基线管理。比如说 state 访问耗时，平时没有延时的时候，state 访问耗时是多少？每个 checkpoint 的数据量大概是多少？在异常情况下，这些都有助于我们对 Flink 的作业的问题进行排查。

4.2 日志

- 错误日志
  JobManager 或者 TaskManager 的关键字及错误日志报警。
- 事件日志
  JobManager 或者 TaskManager 的状态变化形成关键事件记录。
- 历史日志收集
  当作业结束后，想要分析问题，需要从 Yarn 的 History Server 或已经采集的日志系统中找历史信息。
- 日志分析
  有了 JobManager，TaskManager 的日志之后，可以对常见的 failover 类型进行聚类，标注出一些常见的 failover，比如说 OOM 或者一些常见的上下游访问的错误等等。

4.3 关联分析

1. 作业指标/事件 - Taskmanager，JobManager
2. Yarn 事件 - 资源抢占，NodeManager Decommission
3. 机器异常 - 宕机、替换
4. Failover 日志聚类

在做了这些指标和日志的处理之后，可以对各组件的事件进行关联，比如说当 TaskManager failover 时，有可能是因为机器的异常。也可以通过 Flink 作业解析 Yarn 的事件，关联作业与 Container 资源抢占，NodeManager 下线的事件等。

作者简介：

杨阳（时溪），阿里巴巴技术专家，目前就职于阿里巴巴计算平台事业部，负责实时计算中 Flink 运维开发。