背景

前几天知识星球有人提问说：

> 星主你好，请教个问题，yarn 显示 flink job 还在运行中,但是实际上已经失败了,这个怎么判断 flink job 确实挂了呢?因为我需要监控 flink 程序是否正常在运行，好告警。

> 莫忘 提问： 星主你好，请教个问题，yarn显示flinkjob还在运行中,但是实际上已经失败了,这个怎么判断flinkjob确实挂了呢?因为我需要监控flink程序是否正常在运行，好告警，星主有什么好点的方式么?
> Job状态如图

> 依赖yarn的状态会被坑死，我就因为历史包袱导致因为这个背了几个故障
>
> 真实建议查到作业的task级别的状态去判断整个作业的运行状态（PS；有时候作业的状态为running，但task也有fail的），可以在控制台去看下这个接口
>
> 查看详情 >
>
> ithan 觉得很赞
>
> zhisheng：等这周末我写篇我们利用 flink UI 上的 rest api 做的一些工作
> 2020/5/19

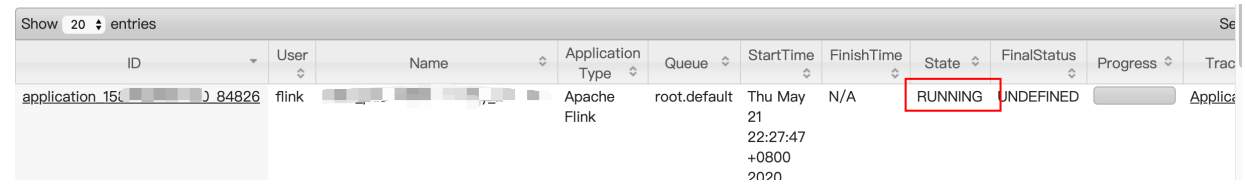
如果你没遇到这个问题，那么说明你遇到的场景还比较少，还得多历练；

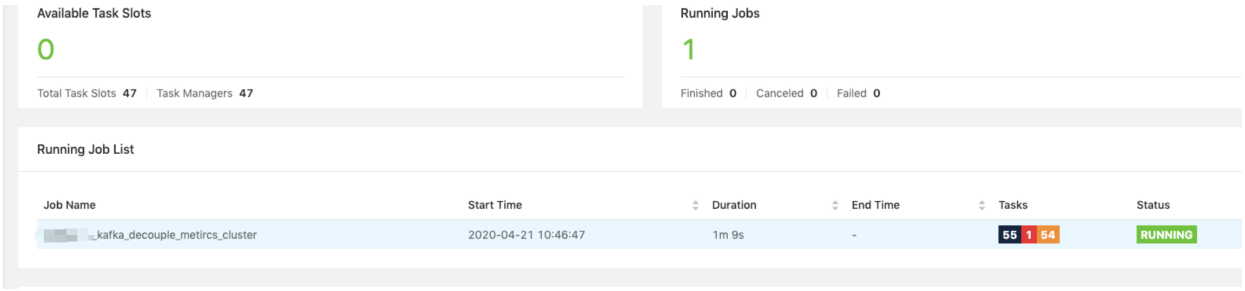
如果你也遇到这个问题，那么我们握握手，以示相互安慰，都是同命苦的小伙伴。

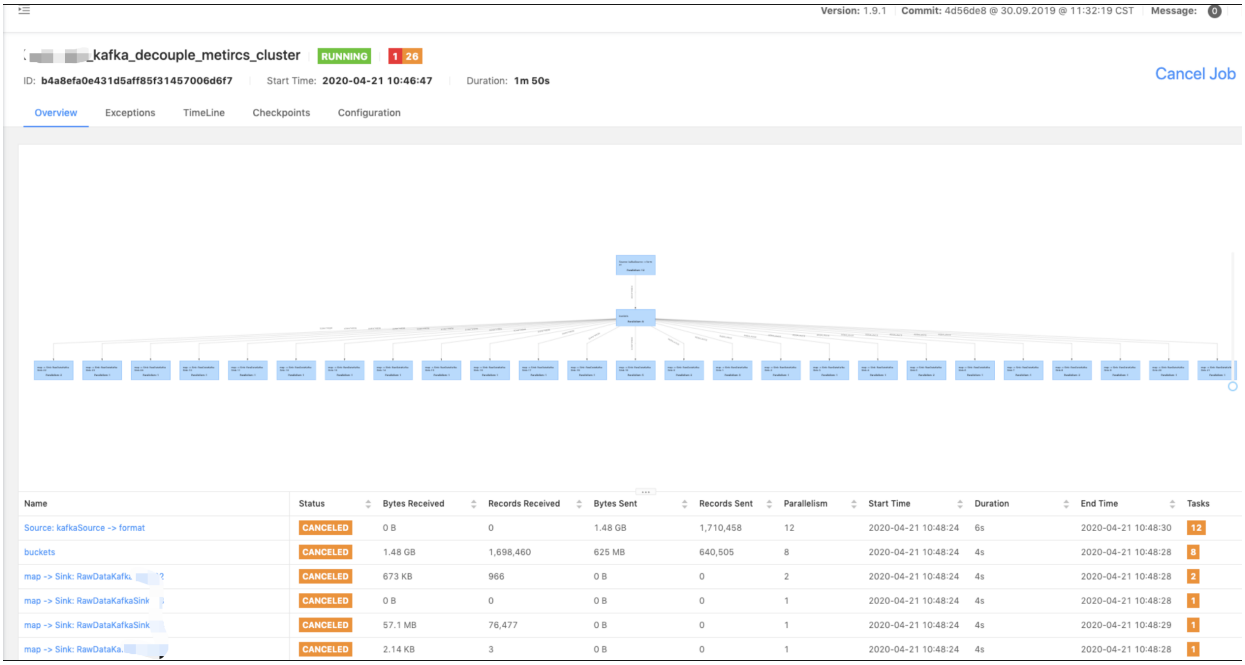
这里不得不又得提一下我就因为这个问题导致背过两个生产事故的故障。这个问题确实会在 Flink 里面出现，起码我现在碰到的次数就好些次了。

问题剖析

这个问题主要的意思是表示通过 YARN 看到作业 AppID 的状态是 Running 的，点进去查看 Flink UI 的状态也是 Running 的，但是 Flink 作业的 Task 其实是有挂（failed）了的。







上面的问题可以用上面三个图来描述，总而言之就是三者的状态不同步。

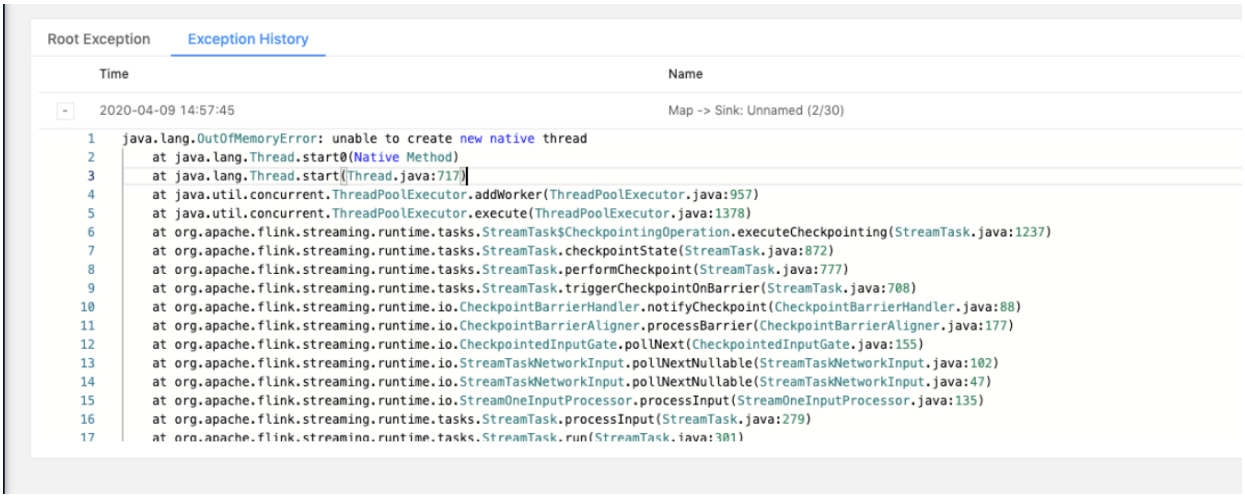
问题影响

如果对作业的状态进行告警只关注 Yarn 或者 Flink 显示的状态，那么自然会错漏掉这种情况，从而导致最终这个作业挂了而一直没被发现。作为作业的 Owner 来说，肯定不能接受公司实时计算平台针对这种情况就不告警了。当遇到大故障的时候，如果对于这种作业不能及时的发现，那么就很可能会出很严重的故障，尤其是这个作业还是非常重要的情况下（数据延迟不能超过公司制定的故障级别）。所以，我就因此问题背过两次该问题的故障，两次相隔没几天，都没来得及修复上线。

复盘一下这两次出现的故障：

- 第一次是因为一个业务方在 IDEA 测试完功能后就直接去生产测试，因为测试环境无数据，他的作业里面引入公司的一个日志中间件，然后他的作业代码里面有个 HBase 的依赖和公司的日志

中间件的依赖有冲突，在消费 Kafka 的数据后写到 HBase 之前会利用公司的日志中间件把一份数据打到另一个 Kafka Topic 里面，因为冲突的问题，导致中间件频繁的创建连接，且数据消费的模式是 startFromEarliest，最后导致创建非常多的线程，然后这个作业 Task 所在的机器都遭殃了，所有的机器线程都被打满了，导致这些机器上面部署调度到的其他作业 Task 也都受到了影响（这个影响是导致 Checkpoint 失败，如下图所示）。



同时伴随着部分作业会一直做 failover，当找到这个导致故障的根本作业后（这个找的技巧后面可以单独说），直接把那个作业 kill 掉了，后面慢慢部分作业的状态都 OK 了，但是就有上面提到的这种作业（状态是 OK，但是 Task 其实已经 failed 的）。针对这种如果没及时发现，就会酿成大故障，在此也提醒大家遇到故障后要及时梳理全排查受影响的范围和严重性，切勿匆匆打扫战场！！！

- 第二次是在上一次故障的后几天，果然验证了那句老话：祸不单行！！！故障也是类似问题，只不过这次是换成写 Redis 了，也是创建了大量的线程导致的。

总结一下两次故障：

因为作业的问题，导致机器线程打满，从而影响了机器上其他的 Flink 作业。暴露的问题主要有两点：

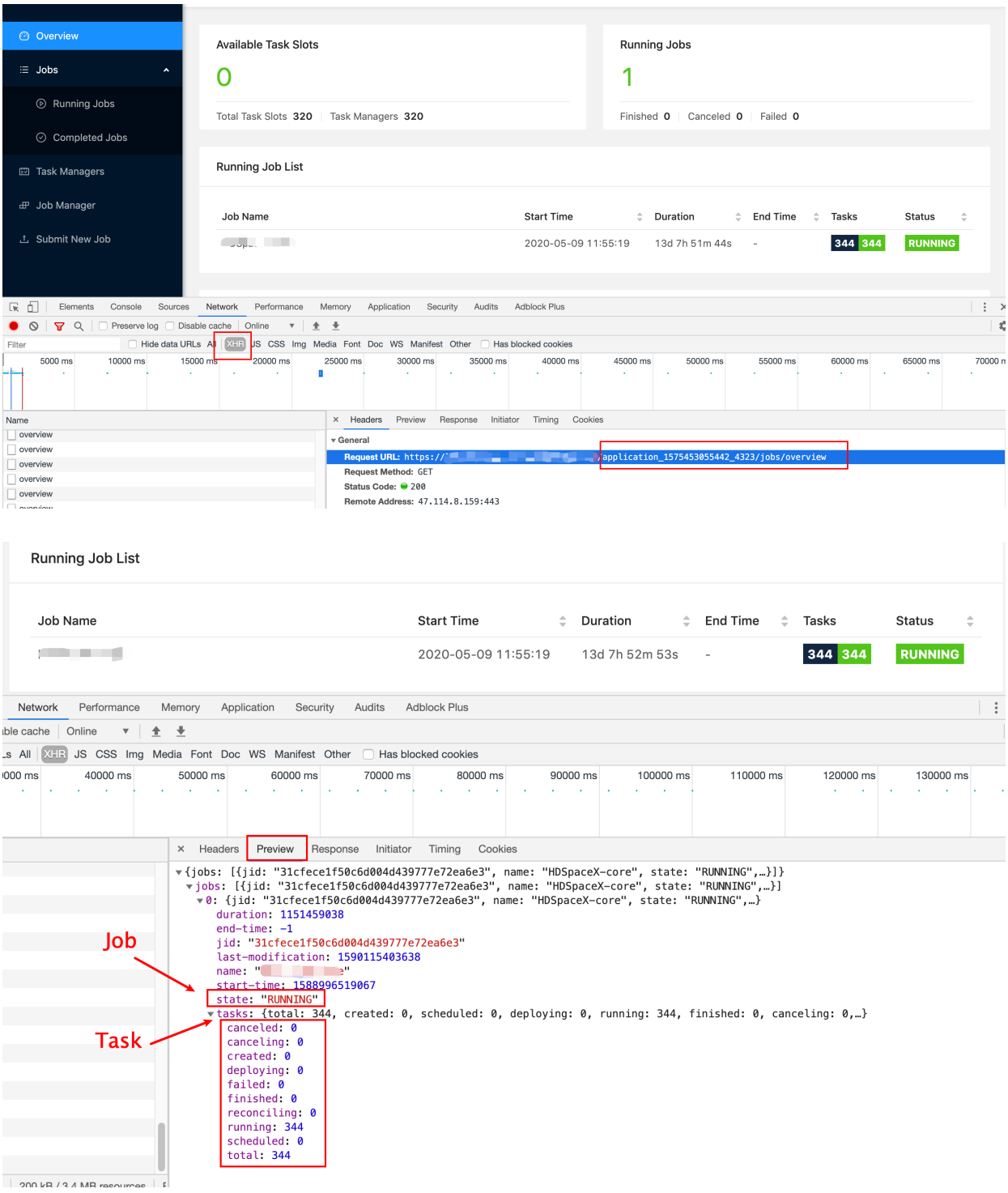
- 监控不够深，对于 Task failed 的未及时告警
- 作业未做到资源相互隔离（导致后面强烈要上 Cgroup）

解决方法

这里 Cgroup 的解决方案就不在这篇文章讲了，其实公众号前几天已经发过一篇文章。

这里讲一下，作业要做到 Task 级别的告警方法（主要利用 Flink Rest API）。

你可以通过该 `https://xxx/application_1575453055442_4323/jobs/overview` 接口拿到作业的状态和 Task 状态信息，如下两图所示：



返回的数据如下 JSON 所示：

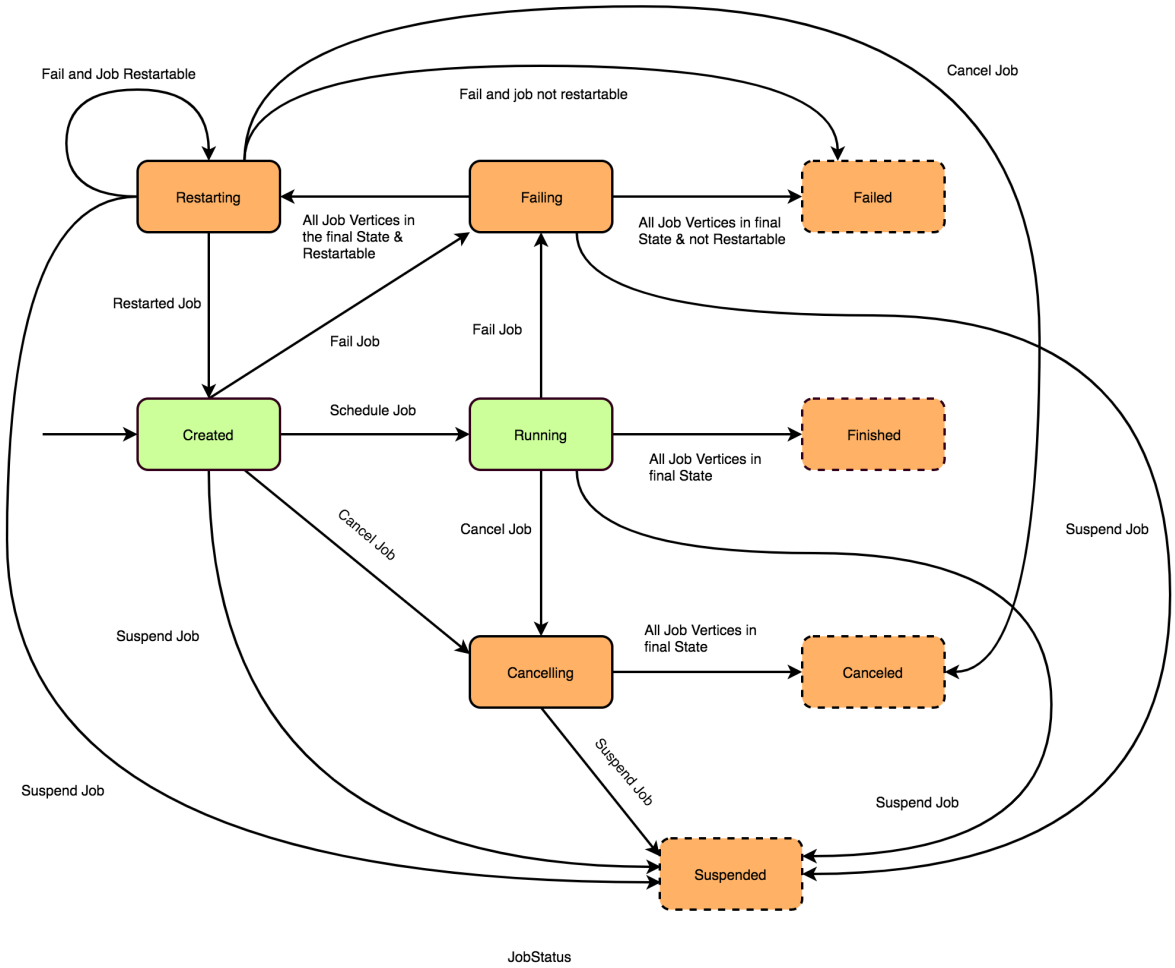
```
{
    "jobs": [{
        "jid": "31cfece1f50c6d004d439777e72ea6e3",
        "name": "xxx",
        "state": "RUNNING",
        "start-time": 1588996519067,
        "end-time": -1,
        "duration": 1151459038,
        "last-modification": 1590115403638,
        "tasks": {
            "total": 344,
            "created": 0,
```

```
            "scheduled": 0,
            "deploying": 0,
            "running": 344,
            "finished": 0,
            "canceling": 0,
            "canceled": 0,
            "failed": 0,
            "reconciling": 0
        }
    }]
}
```

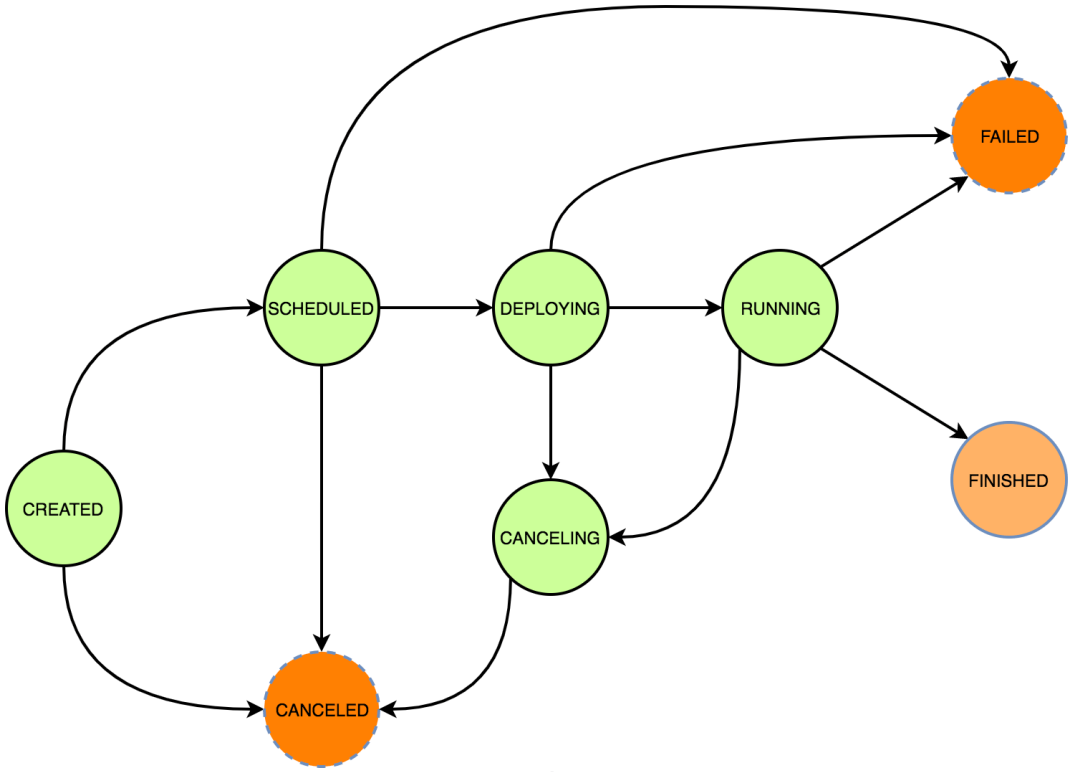
我这边检测的时候是会既判断作业的状态和 Task 的 running 个数是否和 total 个数一致，因为我们生产还有的作业是批和流一起用的，所以只判断 running 个数是否和 total 个数一致还不能判断作业真正的状态，还得再加一个判断是否 failed 的个数是为大于 0。如果大于 0 则表示作业有 Task 失败，然后会给作业的 Owner 告警。

加了这个检测后，作业的状态告警更准，更符合我们的需求，另外我还写成了接口统计所有异常状态的作业，方便故障后的及时梳理工作。

这里再拿出我之前画的关于作业和 Task 的运行状态变换图：



JobStatus

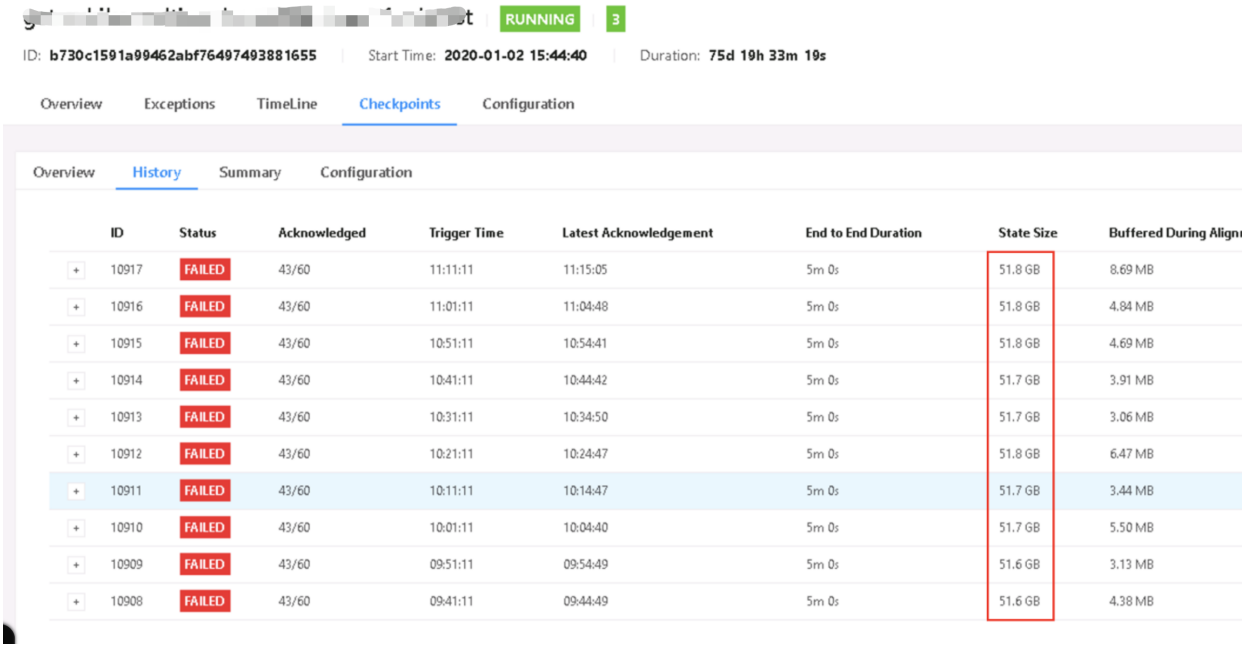


Task State

除了利用上面的这个接口去做作业状态的告警，另外我们还利用了下面这些 API 来做一些日常的统计。

统计大状态作业

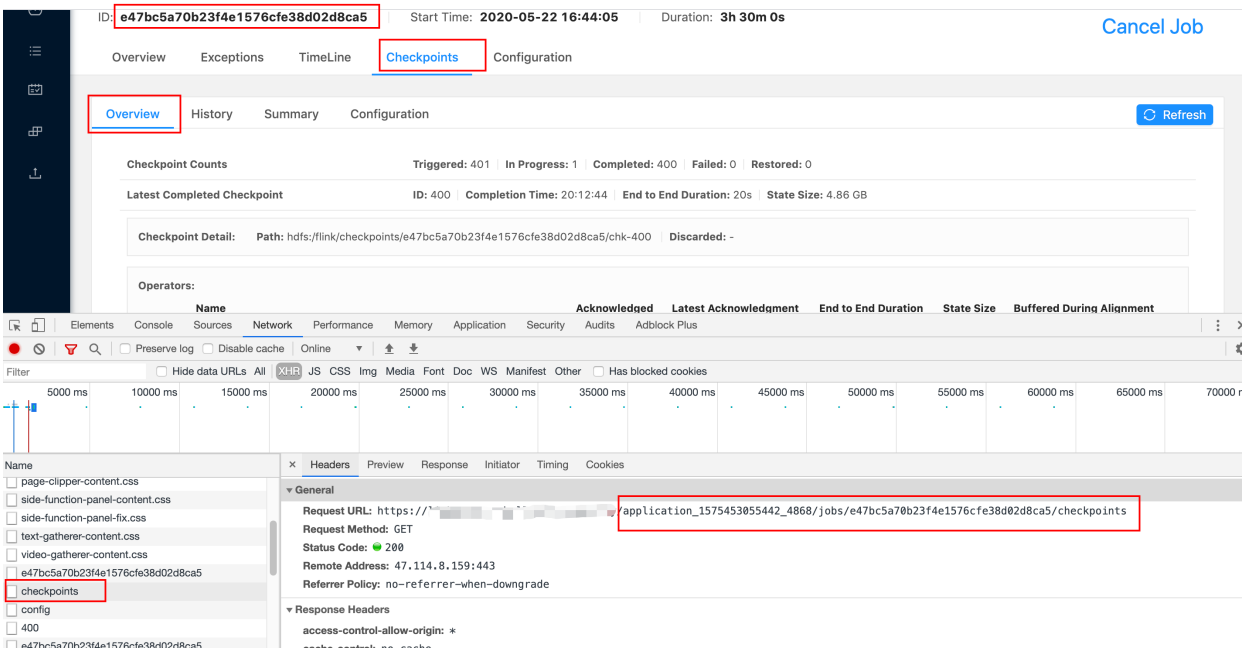
我们生产中也有不少的大状态作业，刚接手实时平台底层的负责的时候，碰到的就是这种大状态作业，做 Checkpoint 会有很多问题，我们就想每隔一段时间统计一下集群的这种大状态作业，然后我这边自己有个心理准备，以防 Checkpoint 的时候打爆机器网卡带宽（也碰到过这种故障）。



那么我们也是利用 Rest API 来检测的，帮助我们不少问题。

上图已经是我们经过几次联系业务开发进行作业治理后的效果了，大状态作业明显降低了很多，针对这个大状态作业的优化问题也是比较复杂的，后面可以单独讲一篇。

你可以通过 `https://xxx/{application_id}/jobs/{jobid}/checkpoints` 接口查看到 Checkpoint 的信息:



接口返回的数据如下:


```
{
    "counts": {
        "restored": 0,
        "total": 401,
        "in_progress": 1,
        "completed": 400,
        "failed": 0
    },
    "summary": {
        "state_size": {
            "min": 281401159,
            "max": 7645131875,
            "avg": 4184641844
        },
        "end_to_end_duration": {
            "min": 5543,
            "max": 47550,
            "avg": 17686
        },
        "alignment_buffered": {
            "min": 0,
            "max": 0,
            "avg": 0
        }
    },
    "latest": {
        "completed": {
            "@class": "completed",
            "id": 400,
            "status": "COMPLETED",
            "is_savepoint": false,
            "trigger_timestamp": 1590149544874,
            "latest_ack_timestamp": 1590149564924,
```

```
            "state_size": 5220376693,
            "end_to_end_duration": 20050,
            "alignment_buffered": 0,
            "num_subtasks": 192,
            "num_acknowledged_subtasks": 192,
            "tasks": {},
            "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-400",
            "discarded": false
        },
        "savepoint": null,
        "failed": null,
        "restored": null
    },
    "history": [{
        "@class": "in_progress",
        "id": 401,
        "status": "IN_PROGRESS",
        "is_savepoint": false,
        "trigger_timestamp": 1590149574874,
        "latest_ack_timestamp": 1590149579446,
        "state_size": 3920009423,
        "end_to_end_duration": 4572,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 127,
        "tasks": {}
    }, {
        "@class": "completed",
        "id": 400,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149544874,
        "latest_ack_timestamp": 1590149564924,
        "state_size": 5220376693,
        "end_to_end_duration": 20050,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-400",
        "discarded": false
    }, {
        "@class": "completed",
        "id": 399,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149514874,
        "latest_ack_timestamp": 1590149520970,
        "state_size": 4678903399,
        "end_to_end_duration": 6096,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-399",
        "discarded": true
```

```
    }, {
        "@class": "completed",
        "id": 398,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149474586,
        "latest_ack_timestamp": 1590149512772,
        "state_size": 4269715948,
        "end_to_end_duration": 38186,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-398",
        "discarded": true
    }, {
        "@class": "completed",
        "id": 397,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149444586,
        "latest_ack_timestamp": 1590149462786,
        "state_size": 4688939894,
        "end_to_end_duration": 18200,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-397",
        "discarded": true
    }, {
        "@class": "completed",
        "id": 396,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149414586,
        "latest_ack_timestamp": 1590149421011,
        "state_size": 4872715569,
        "end_to_end_duration": 6425,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-396",
        "discarded": true
    }, {
        "@class": "completed",
        "id": 395,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149376379,
        "latest_ack_timestamp": 1590149412150,
        "state_size": 4344812828,
        "end_to_end_duration": 35771,
        "alignment_buffered": 0,
        "num_subtasks": 192,
```

```
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-395",
        "discarded": true
    }, {
        "@class": "completed",
        "id": 394,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149346379,
        "latest_ack_timestamp": 1590149364845,
        "state_size": 5322745287,
        "end_to_end_duration": 18466,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-394",
        "discarded": true
    }, {
        "@class": "completed",
        "id": 393,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149316379,
        "latest_ack_timestamp": 1590149322623,
        "state_size": 3382116678,
        "end_to_end_duration": 6244,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-393",
        "discarded": true
    }, {
        "@class": "completed",
        "id": 392,
        "status": "COMPLETED",
        "is_savepoint": false,
        "trigger_timestamp": 1590149279721,
        "latest_ack_timestamp": 1590149314164,
        "state_size": 4695354056,
        "end_to_end_duration": 34443,
        "alignment_buffered": 0,
        "num_subtasks": 192,
        "num_acknowledged_subtasks": 192,
        "tasks": {},
        "external_path":
"hdfs:/flink/checkpoints/e47bc5a70b23f4e1576cfe38d02d8ca5/chk-392",
        "discarded": true
    }]
}
```

你其实就可以发现上面可以拿到很多信息，比如每次 Checkpoint 的大小、路径、时间、最新的一次、summory 的信息。通过这些你可以拿到作业的状态信息，对于检测大状态（这个判定为大状态可以根据你们公司的场景来定，我目前是根据多个指标关联起来才判断是大状态）特别有效。

上面的 Checkpoint 状态的路径也是一个非常重要的数据，我们也会存储作业最新的一个路径，方便用户作业挂的时候能够从上一个新的 Checkpoint 路径恢复。

Checkpoint 时间配置

针对这个是因为想梳理一下公司目前所有作业的 Checkpoint 的配置，来看到底是否合理，也是有接口的，你可以通过 `https://xxx/{application_id}/jobs/{jobid}/checkpoints/config` 来查看，返回的结果如下：

```
{
    "mode": "exactly_once",
    "interval": 120000,
    "timeout": 600000,
    "min_pause": 0,
    "max_concurrent": 1,
    "externalization": {
        "enabled": false,
        "delete_on_cancellation": true
    },
    "state_backend": "RocksDBStateBackend"
}
```

这几个字段的含义：

- mode 模式：有 EXACTLY_ONCE、AT_LEAST_ONCE、NONE
- interval：单位是毫秒，如果是 9223372036854776000 则表示没开启 Checkpoint
- timeout：单位是毫秒
- state_backend：有 RocksDBStateBackend、MemoryStateBackend、FsStateBackend

另外还可以看到这些作业是否开启了 Checkpoint。

作业 Flink 版本

因为现在我们集群有两个 Flink 版本，所以我们也有接口去统计集群上所有作业 Flink 版本的统计，你可以看看 `http://xxx/{application_id}/config` 该接口，返回的数据如下：

```
{
    "refresh-interval": 3000,
    "timezone-name": "China Time",
    "timezone-offset": 28800000,
    "flink-version": "1.9.1",
    "flink-revision": "4d56de8 @ 30.09.2019 @ 11:32:19 CST"
}
```

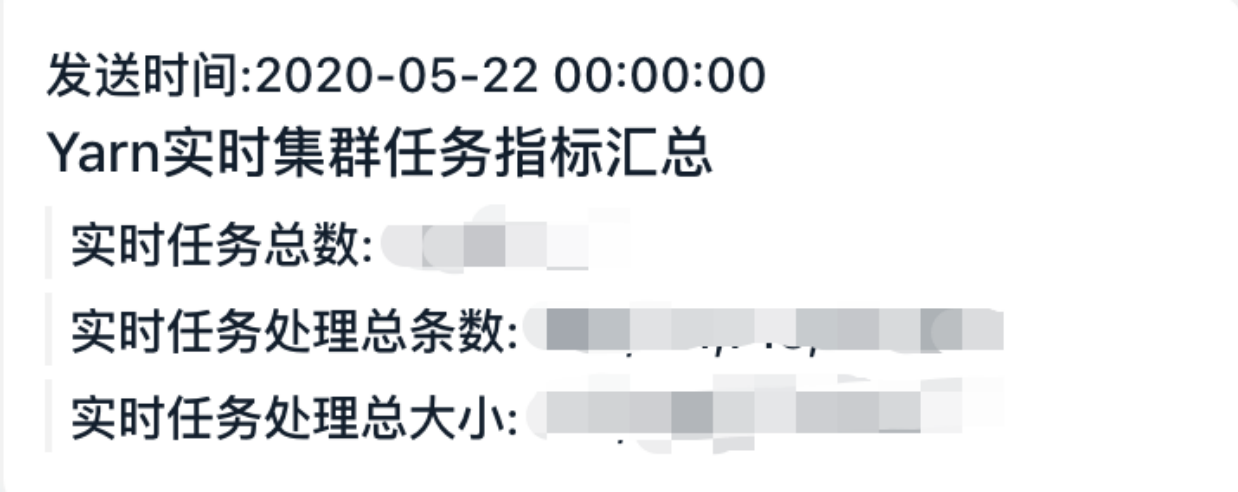
这里可以拿到 `flink-version` 字段的值，但是我们发现有的版本竟然是 `<unknown>` 。

作业处理数据量

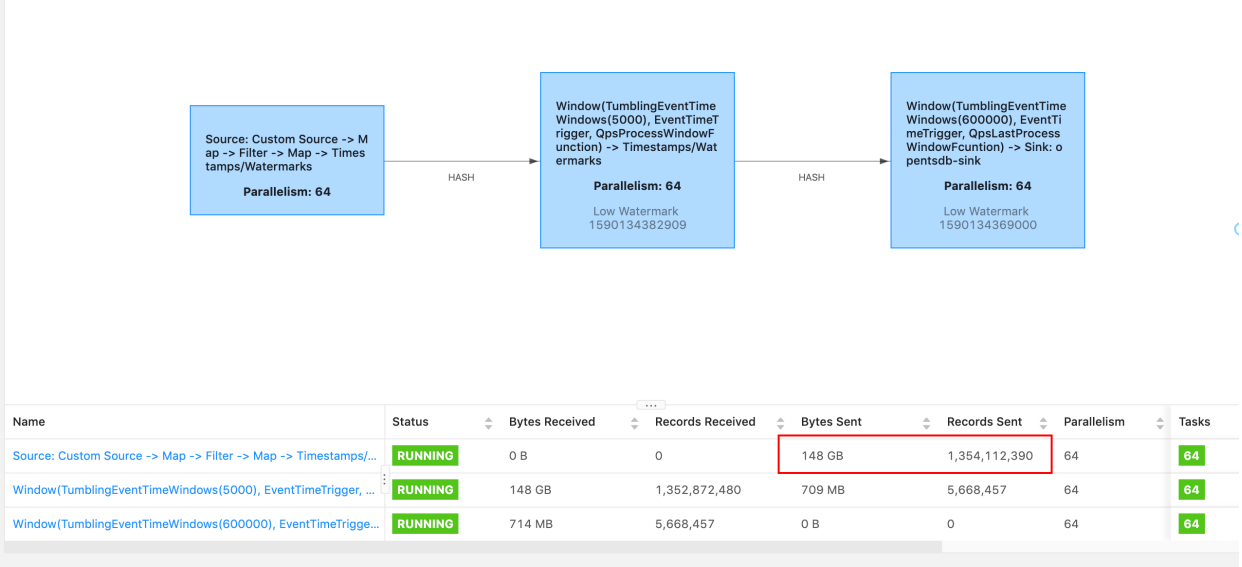
还有一个比较重要的就是统计每天我们集群所有的 Flink 作业处理的流数据的数据总量，供我们领导查看，这个就很关键了。

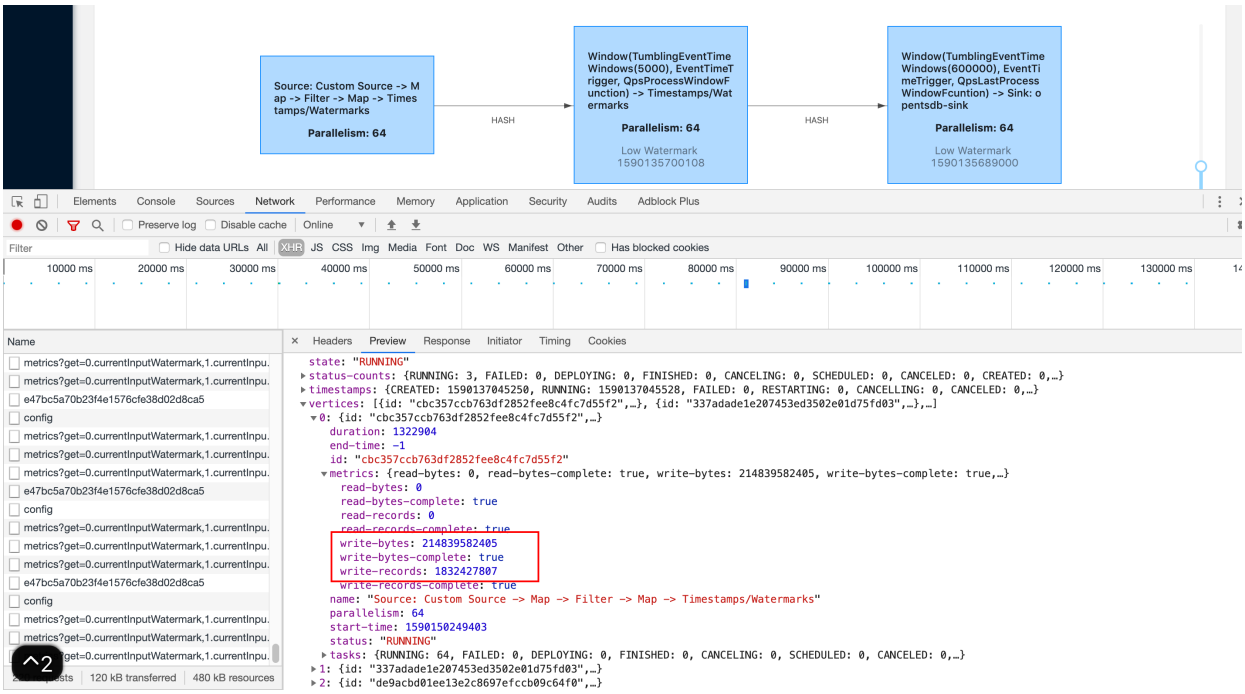
目前也是通过 UI 上的 Rest API 信息查看到的信息，然后统计每个作业的，最后汇总成一个总数。

效果如下：



你可以通过单个作业的查看，比如下图：





但是光从这个 API 里面拿到的值其实有的作业是不准确的，比如作业 task 都是 chain 在一起的，那么是统计不到这个信息的，我们是通过 metric 那边（有点复杂）获取到的值。

其他

上面列举了我们常用的几个很刚需的场景，完全就是利用了 Flink Rest API 来做的一个监控告警。另外还有些其他的指标，这里就没有讲了。除了利用 API 来统计，我们强烈建议给你们集群加上 Flink Metric Reporter 来统一收集所有的作业指标信息，然后统一的监控告警和指标大盘。这块自己也在知识星球发过文章的，不清楚的可以去找找。

总结

没有非常完善的监控系统，只有更完善，不断完善的监控系统。

出现问题，及时排查，及时解决，及时排查影响范围，及时修复问题。

多维度充分利用现有的信息，以达到我们最刚需的需求。