Blink资源配置的三年双11(2016~2018)回顾

资源配置指的是配置作业运行时各个阶段的资源，包括并发度，CPU，内存（分为heap memory，native memory，directory memory）等资源，让作业能在最少的资源占用下达到最优的性能。资源配置的太多会造成浪费，配置的太少性能又不够。此外，资源配置的方式如果太复杂，对用户来说成本也比较高。

如何既方便快捷，又合理的配置资源对各种大数据处理系统来说一直都是一个难点。

目前业界的各种大数据处理引擎，比如Hive，JStorm，Spark一般都提供了如下几种方式来配置资源：
- 默认配置文件，如 storm.yaml
- 提交作业的命令行参数，如 ./bin/xxx-submit --executor-memory 1G --num-executors 1 --conf xxx.sql.shuffle.partitions=100
- 代码中设置，如：topologyBuilder.setBolt("2", new WordSpliter(), 4)

Flink中也提供了类似的方案：（1）flink-conf.yaml配置 （2）./bin/flink run命令行 （3）以及由阿里的同学在Flink 1.3版本加入社区的 setResource API接口（FLINK-5133）。通过这三种方式就能从粗粒度到细粒度来设置作业的资源。

接下来，通过回顾2016~2018这三年双十一来说下Blink资源配置的演变历程。

2016：文本配置

2016年的双十一，是Blink的第一年双十一，成功支持了主搜索离线计算，以及搜索和推荐全链路实时化。采用的Blink版本是 0.8.x，任务的类型以DataStream和TableAPI为主。

通过Java和Scala开发的Blink程序，如果把资源都通过API的方式写死在代码里面，显然不能满足快速调优和业务迭代的需求。因此业务方很多是自己定义了一套业务相关的配置文件，比如配置 taobao_source1, wireless_source2 的cpu和并发度，然后在代码里面解析后设置到对应的节点上。这样造成的问题是，每个业务方都得定义自己的一套格式，进行解析后设置，并不通用，重复工作较多。

因此Blink提供了通过JSON格式资源文件的方式来配置资源的方式。具体做法是：
- 通过 compile 命令，将代码预编译，输出各个transformation节点和默认资源配置到json文件中
- 用户修改json文件，可以设置各个transformation节点的并发度，cpu，memory等
- 用 run 命令提交任务的时候通过 -conf 指定这个资源配置的json文件

JSON文件的格式类似于：

```
"nodes": [
   {
     "id": 1,
     "uid": "1",
     "pact": "Data Source",
     "name": "Source: CustomTypeTT4TableSource-wireless_search_pv",
     "slotSharingGroup": "default",
     "chainingStrategy": "HEAD",
     "parallelism": 512,
     "vcore": 1,
     "heap_memory": 1024,
     "direct_memory": 64,
     "native_memory": 0,
     "state_size": 0
   },
   {
     "id": 2,
     "uid": "2",
     "pact": "Operator",
     "name": "SourceParser:EternalTTSourceCollector",
     "slotSharingGroup": "default",
     "chainingStrategy": "ALWAYS",
     "parallelism": 256,
     "vcore": 0.2,
     "heap_memory": 128,
     "direct_memory": 32,
     "native_memory": 0,
     "state_size": 0
   },
```

JSON资源配置文件的优点：
- 统一了资源配置的格式，业务方不需要在代码里面耦合资源设置的代码了，只需要关注业务逻辑就行了，减轻了很多工作量。

但同时也引入了新的问题：
- 复杂任务的JSON资源文件长达几千行，有几百个transformation，每次改动都是个艰巨的工作
- transformation节点和真正的物理执行节点的对应关系不是很明确，有时候需要多次反复修改才能确定

还记得2016年双十一，给搜索的珠峰项目做压测和调优的时候，和同事一起通过 vim 编辑json文件，批量替换，反复修改定位节点，耗时几小时才能完成一个稍复杂任务的调优。

2017：可视化配置

2017年的双11，Blink开始通过阿里实时计算平台Bayes面向全集团用户，任务类型以SQL为主，版本为1.4.x。社区的Flink SQL本身并没有提供资源设置的方式，Blink当时开发了SQLRunner/JobLauncher，提供通过JSON资源文件配置SQL任务的资源的方式。因为SQL任务的底层最终也是翻译成transformation，所以提供JSON文件的资源配置方式是比较自然的。

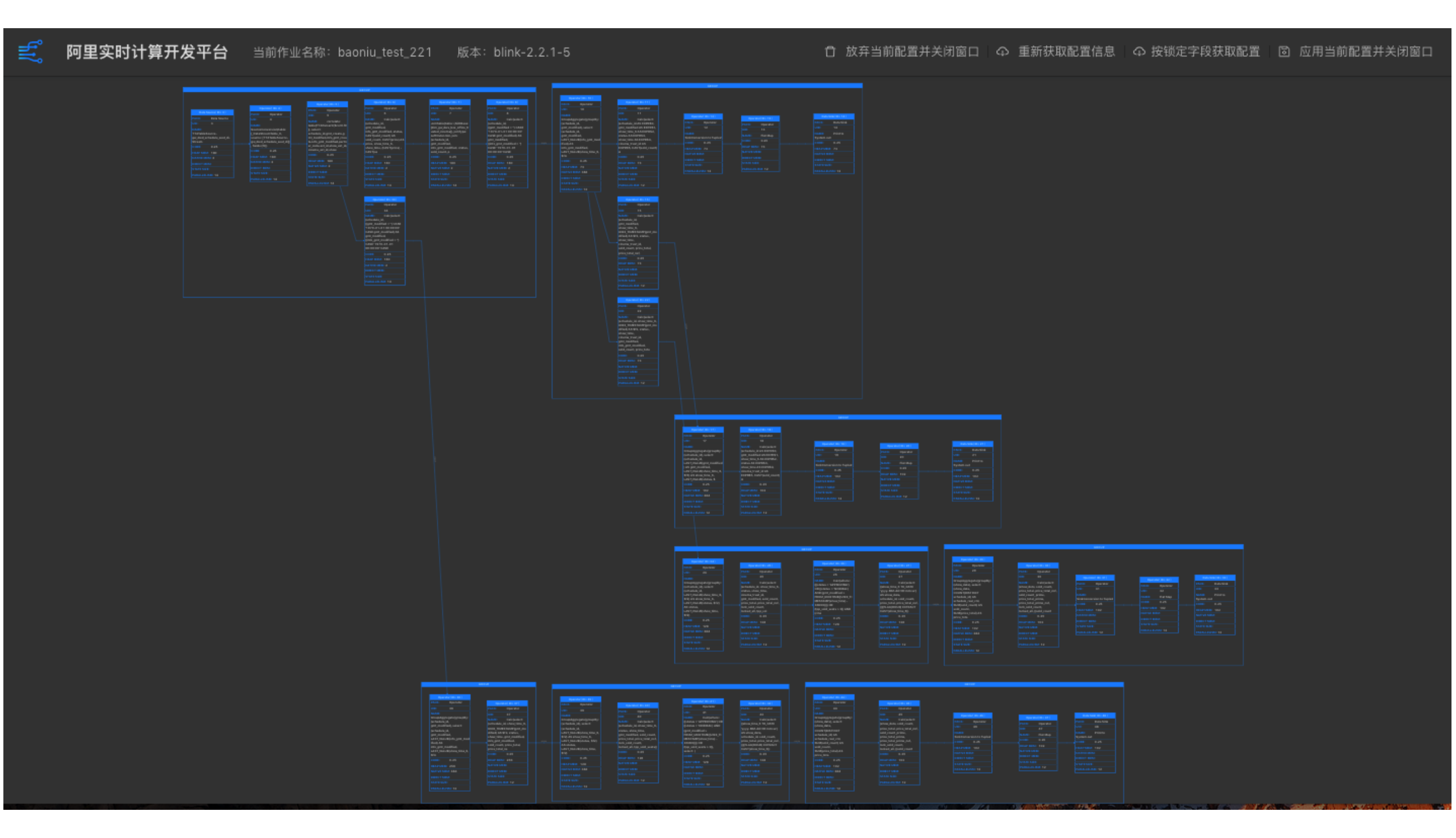
具体使用流程是：用户在Bayes平台上写完SQL语句，通过『获取资源配置』就能生成一份默认的JSON资源文件。然后用户可以在Bayes页面上直接修改json文件，或者拷贝出来修改，保存后上线和运行。

通过Bayes平台进行资源配置的优点是：流程更简单，资源配置生成和修改在一个界面，鼠标点击就能完成。
但缺点也一样存在：对于长达几千行的复杂json资源文件，每次修改都很繁琐，费时费力。

因此当时开发了一个『可视化资源配置』的功能：

用户只需要将json文件拷贝到这个系统中，就能以可视化的方式展示出来，这样用户可以设置Vertex（对应图中的Group）或者单个Operator的资源。配置完成后，需要将这个json文件拷贝到Bayes系统中。

可视化资源配置系统在双十一之后就直接集成到Bayes平台了。



可视化资源配置的出现大大减轻了用户手工修改JSON文件的工作量，降低了出错概率。对Blink SQL任务来说，用户一般只需要配置Vertex级别的资源就可以了，不需要精确配置每个Operator的资源。

但可视化配置也存在同样的问题：对于复杂的任务即使可视化展示出来，拓扑图也很复杂；很多节点名字一样，仍然难以配置。此外，对于流式任务来说，任务的性能瓶颈并不是一下子就能定位到，往往需要多次运行，看哪里有反压确定系统的瓶颈，然后调大对应的资源。用户还是需要花不少时间才能配置好一个任务的资源，有没有更自动化的方式?

2018：半自动化配置

『资源配置』和『性能调优』一直是每年双十一工作的重点。性能调优不仅仅局限于资源配置，但是资源配置和性能息息相关。Blink团队一直在积极探索更加智能的资源配置方式，希望能降低用户的工作量，同时能节省集群资源。

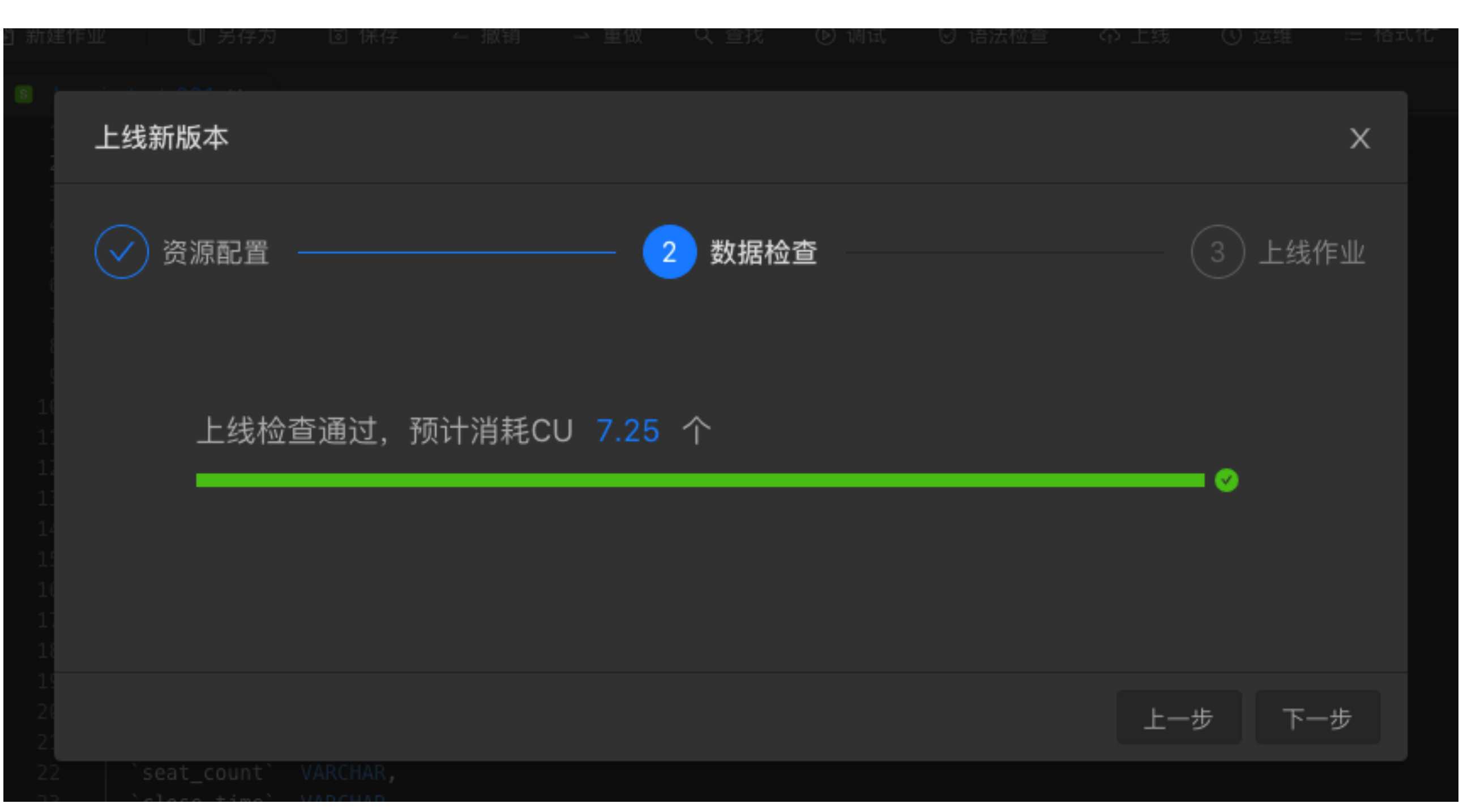
经过一段时间的开发测试，Blink团队于2017年12月推出了1.4.5版本，提供自动配置（AutoConf）功能。其基本原理是：采样收集任务运行时的指标（Metrics）信息，在下次运行前通过算法给出推荐的资源配置。

AutoConf 使用流程

- （1）用户指定任务使用多少 CU 资源（1 CU对应 1个CPU核，4G内存），AutoConf算法会根据历史运行情况计算生成一份资源配置
  - 如果任务是第一次运行，之前没有历史记录，这时候算法会根据经验值生成一份初始推荐配置。
  - 注意：初始值只是根据经验值算出来的，不一定准确，甚至可能和实际情况相差较大（例如用户的source源partition数很大）。需要用户多进行几次调优才能得到比较合理的配置。
- （2）使用生成的配置运行，在运行期间Blink会采集一些指标信息（比如：输入输出tps, latency,cpu,memory 等）
- （3）待任务运行一段时间后，重复步骤(1)

一般通过几轮迭代能得到一个较优的资源配置。

资源配置方式：● 智能推荐配置（10.00 CU 可用）：指定使用 8 CU
○ 使用上次资源配置(手动资源配置)



上线检查通过，预计消耗CU 7.25 个

AutoConf 工作原理

AutoConf 的工作原理主要基于以下几点：
- 在流计算任务中基于稳定状态的 statistics 可以得到足够的信息用来优化配置。
- Blink 任务一次实验（修改配置+运行+计算 statistics）的代价（时间+资源+用户操作+任务影响）很高，所以尽量用少数实验来推导出可用的配置。

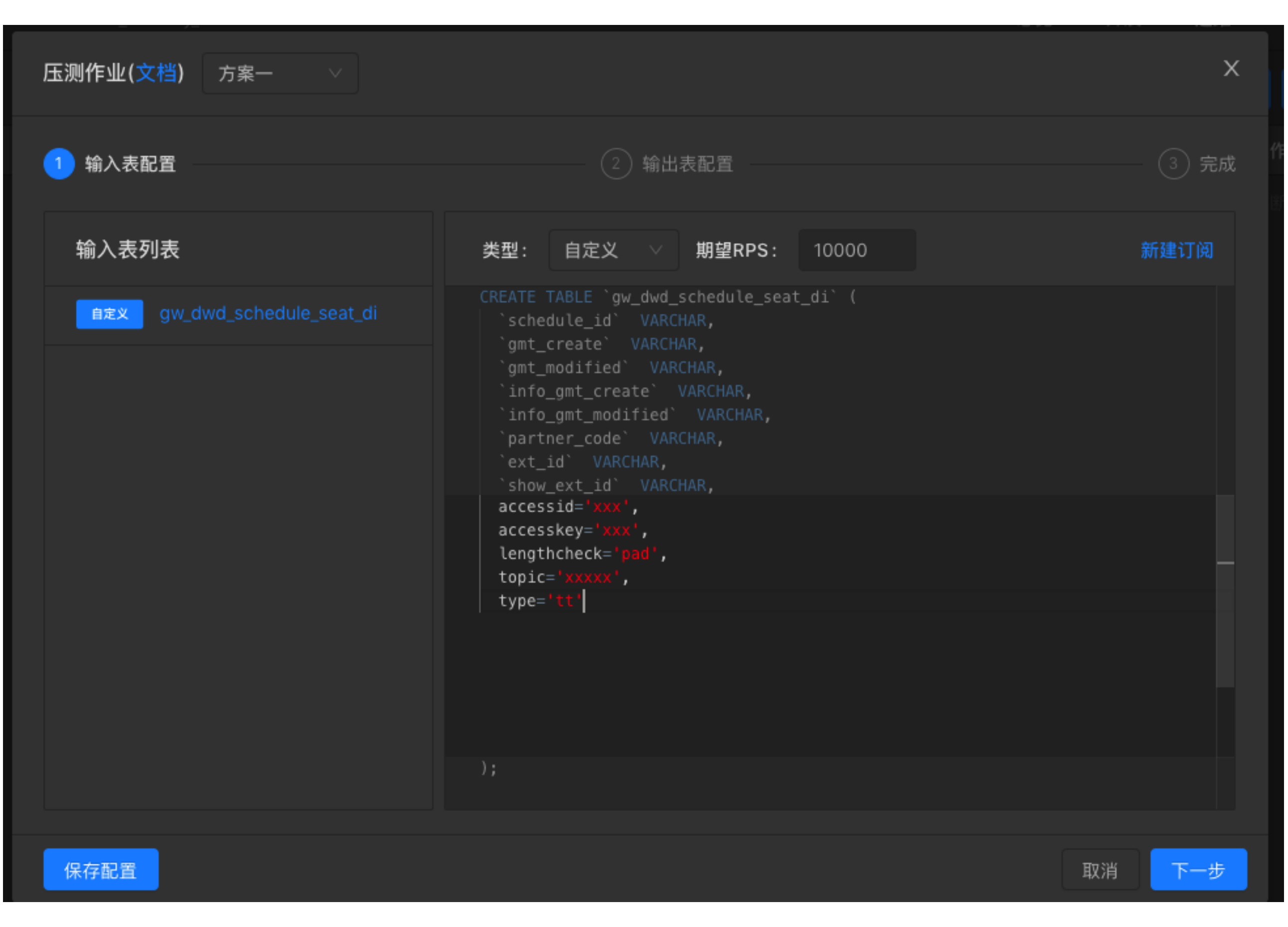
同时基于以下假设：
- 在资源配置足够的情况下，一个节点单个并发度（task）的一个 event 平均净处理时间和固定 overhead 基本不变。
- 每个 task 的平均输出 QPS 和输入 QPS 之间的比例基本不变。

其基本工作模式：
- 并发度和资源的设置可以分开考虑
- 一开始配置略宽松的资源让任务能够正常运行（不需要达到目标流量），以便拿到 metrics 来计算 statistics。从稳定状态的 statistics 可以得到比较准确的资源配置。稳定状态越接近目标状态（数据流量，并发度，资源）得到的 statistics 越准确，几次后会收敛。
- 节点并发度的比例关系可以通过workload的比例算出；节点的资源可以通过metrics得到，比如cpu，heap memory。
- 有了每个节点资源配置和节点之间并发度的比例后，可以把所有节点的并发度按比例放大到目标 CU，得到最后配置。

AutoConf 在发布以后，根据用户反馈做了多轮的优化，比如：
- 针对不同Source类型采用不同的资源配置策略
- 根据Full GC情况调整heap memory
- 在1.6.x版本使用RocksDB StateBackend时根据metrics自动调整native memory
- 更智能的选取Metrics时间段
- 自动配置与手工配置相结合
- 自动配置下可以锁定部分节点的资源配置
- 更友好的提示和报错信息

自动任务压测

AutoConf 的出现在一定程度上减轻了用户的工作量，对于很多任务来说，通过 AutoConf 生成的配置已经可以正常运行。当发现性能问题的时候，用户可以选择再次迭代或者手工配置。但是这个迭代的过程还是需要人工干预的，整个调优迭代的过程比较繁琐。因此双十一之前Blink团队开发了『自动任务压测』功能，将 AutoConf 的调优迭代的过程集成在压测系统中，免去了用户手工干预的痛苦。



gw_dwd_schedule_seat_di

类型：自定义　期望RPS：10000

```
CREATE TABLE `gw_dwd_schedule_seat_di` (
  `schedule_id`  VARCHAR,
  `gmt_create`  VARCHAR,
  `gmt_modified`  VARCHAR,
  `info_gmt_create`  VARCHAR,
  `info_gmt_modified`  VARCHAR,
  `partner_code`  VARCHAR,
  `ext_id`  VARCHAR,
  `show_ext_id`  VARCHAR,
  accessid='xxx',
  accesskey='xxx',
  lengthcheck='pad',
  topic='xxxxx',
  type='tt'

);
```

最大运行时间(分钟)：60　迭代间隔(分钟)：10

回追时间：2018-10-10 21:08:57　作业最大CU数：200 (可用 0.33 CU)

启用 AutoConf：●

压测说明

1. 压测系统会回追数据，根据任务的历史运行情况进行自动调优和迭代，生成一个更好的资源配置。尽量使用真实的数据压测，效果更好。
2. 由于SQL本身性能问题，可能无法达到目标RPS。例如UDF/UDTF/UDAF等存在性能问题，数据倾斜和热点问题，sink表性能不够等。请先对SQL进行优化，再进行压测。
3. Source表分区数太小也会影响最终效果

方案一 2018-10-19 14:05
方案一 2018-10-18 14:45
方案一 2018-10-18 14:25

任务名称：bpt_virable_job_1

迭代次数：1　任务状态：成功

开始时间：2018-10-19 14:05　结束时间：2018-10-19 14:11

平均RPS：11061.0649　峰值RPS：12160.7224

metrics：Link　Failover：否

源表名	回追比率
datahub_input	2924.980553836324

『自动任务压测』的目的是为了解放用户，让整个调优和压测的过程更加自动化，用户可以选择启用AutoConf每次生成新的配置，也可以选择使用自己已经调好的配置。

展望

AutoConf 并不是万能的，算法本身做了很多假设，比如假设每个Task处理的数据是均匀的，每条数据处理的时间是一样的等。此外，对于SQL本身写法导致的性能问题，单纯通过AutoConf来调大资源使用，收益并不明显，很容易达到性能瓶颈。例如：
- SQL里面使用了大量的正则表达式
- 用户的UDF实现有耗时较大的外部IO导致性能较差，调大并发度和CPU也很难达到预期性能。
- SQL写法导致数据倾斜，如果不解决好倾斜问题，只是调大并发度效果也不明显。
- 源表分区数有限制导致无法扩大并发；维表或者结果表性能不足

如果用户从SQL和业务代码层面去优化，性能的提升会更明显，因此Blink除了在引擎层面优化SQL执行计划，也提供了专门的SQL优化文档来指导用户实践。

Blink团队还在对资源配置方式进行不断的优化中。可视化资源配置将更方便地支持具有复杂拓扑结构的任务。AutoConf算法未来或许可以实现完全无需人工干预的智能配置，将 AutoConf 与 SQL 执行计划生成相结合，流量高低峰期自适应配置等都在考虑中。未来可期！