|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| **FusionInsight Flink**  **V100R002C70SPC200** | | |  |
| **性能调优指导手册** | | |
| **文档版本** | **01** | |
| **发布日期** | **2017-11-10** | |
|  | | | | |
|  | 华为技术有限公司 | |  |  |

|  |
| --- |
| 版权所有 © 华为技术有限公司2017。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | <http://e.huawei.com> |

目 录

[1 概述 1](#_Toc496774545)

[1.1 模块架构模型 1](#_Toc496774546)

[1.2 性能衡量指标 2](#_Toc496774547)

[1.2.1 衡量指标 2](#_Toc496774548)

[1.2.2 指标观测方法 2](#_Toc496774549)

[2 集群服务部署规划 3](#_Toc496774550)

[2.1 服务规模与业务容量参数配置对照表 3](#_Toc496774551)

[3 典型业务的调优 4](#_Toc496774552)

[3.1 调优目标 4](#_Toc496774553)

[3.2 调优原则 4](#_Toc496774554)

[3.3 性能调优常用方法-DataStream调优 5](#_Toc496774555)

[3.3.1 配置内存 5](#_Toc496774556)

[3.3.2 设置并行度 5](#_Toc496774557)

[3.3.3 配置进程参数 6](#_Toc496774558)

[3.3.4 设计分区方法 7](#_Toc496774559)

[3.3.5 配置netty网络通信 8](#_Toc496774560)

[3.3.6 经验总结 9](#_Toc496774561)

[4 性能瓶颈监控及调优 10](#_Toc496774562)

[4.1 常见性能问题及解决方案 10](#_Toc496774563)

[4.1.1 监控手段 10](#_Toc496774564)

[4.1.1.1 Flink日志概述 10](#_Toc496774565)

[4.1.1.2 任务信息收集 10](#_Toc496774566)

[4.1.1.3 监控页面 13](#_Toc496774567)

# 概述

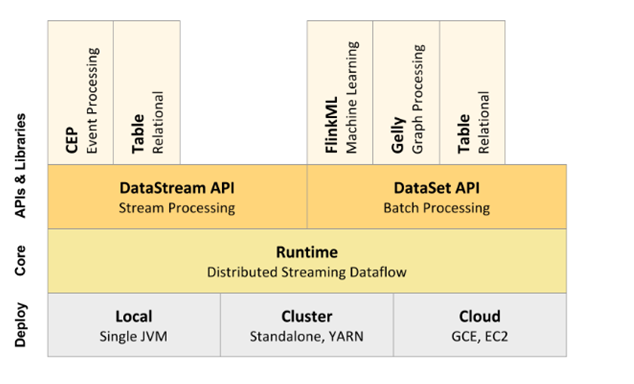
[1.1 模块架构模型](#_ZH-CN_TOPIC_0067016242)

[1.2 性能衡量指标](#_ZH-CN_TOPIC_0067016243)

## 模块架构模型

Flink是一个批处理和流处理结合的统一计算框架，其核心是一个提供了数据分发以及并行化计算的流数据处理引擎。它的最大亮点是流处理，是业界最顶级的开源流处理引擎。

Flink最适合的应用场景是低时延的数据处理（Data Processing）场景：高并发pipeline处理数据，时延毫秒级，且兼具可靠性。



## 性能衡量指标

### 衡量指标

* 吞吐量：单位时间内完成的计算任务数量
* 资源利用率：对分配到资源的使用率
* 伸缩性：
* 横向扩容带来的性能提升曲线
* 增加系统负担带来的性能下降曲线

### 指标观测方法

* 吞吐量：在相同资源环境下，执行相同计算任务，查看任务的完成速度
* 资源利用率：执行计算任务，查看在不同负载情况下，CPU、内存、网络的使用率。
* 伸缩性：
* 横向扩容带来的性能提升曲线：增加资源，执行相同计算任务，查看性能提升比率。
* 增加系统负担带来的性能下降曲线：在相同资源环境下，增加计算负载，查看性能下降比率。

# 集群服务部署规划

[2.1 服务规模与业务容量参数配置对照表](#_ZH-CN_TOPIC_0067016247)

## 服务规模与业务容量参数配置对照表

Flink作为流数据处理引擎，依赖内存和CPU。用户在规划规格时，应根据当前的业务容量和增长速度，规划合理的内存和CPU资源，特别需要关注以下几点：

* 根据自己的业务目标，规划CPU资源和内存资源。规划时，需要结合当前的数据分布情况，业务复杂度，设置JobManager的内存，TaskManager的数量，TaskManager的内存，每个TaskManager的slot数量，规划适当的CPU核数和内存大小。
* 在规划内存时，要预留一定量的内存空间作为操作系统的buffer cache，一般预留20%。
* 从HDFS中读入数据时，要考虑block解压缩后的数据膨胀。
* 规划一定的磁盘作为缓存空间，包括：
* 缓存数据：通过分析业务逻辑，估算缓存数据容量，并需要注意压缩数据的膨胀。
* 日志：通过“yarn.nodemanager.log-dirs”参数配置，可以设置多个磁盘的目录，分担磁盘IO负载，提高读写盘性能。

# 典型业务的调优

[3.1 调优目标](#_ZH-CN_TOPIC_0067016185)

[3.2 调优原则](#_ZH-CN_TOPIC_0067016186)

[3.3 性能调优常用方法-DataStream调优](#_ZH-CN_TOPIC_0067016187)

## 调优目标

Flink调优的目标是在不影响其他业务正常运行的前提下，高效的完成业务目标，通常为了达成该目标，一般需要最大限度利用集群的物理资源，如CPU、内存、磁盘IO，使其某一项达到瓶颈。

## 调优原则

1. 提高CPU使用率同时减少额外性能开销。

调优方法：

* 1. 根据业务模型，设置合适的垃圾收集器，通过分析GC日志，设置合理的分区大小，GC线程并发度，减少full gc操作。
  2. 设置合理的TaskManager数量和每个TaskManager对应的slot数量，使每个节点上的有合理的任务并行度，注意slot数量不宜太高，避免线程的额外开销。
  3. 在算子中设置partition数量，防止内存不足GC。
  4. 防止数据倾斜，可以使用rebalance等接口进行数据均匀划分。

1. 提高内存使用率。

调优方法：

* 1. 根据任务情况，设置TaskManager的内存足够。
  2. 根据数据处理大小，设置“taskmanager.network.numberOfBuffers”缓存buffer数据块多少。

1. 优化业务逻辑，减少计算量和IO操作。

调优方法：

* 1. 提前过滤不必要的数据。
  2. 尽量重用内存空间，避免重复计算。

## 性能调优常用方法-DataStream调优

### 配置内存

操作场景

Flink是依赖内存计算，计算过程中内存不够对Flink的执行效率影响很大。可以通过监控GC（Garbage Collection），评估内存使用及剩余情况来判断内存是否变成性能瓶颈，并根据情况优化。

监控节点进程的YARN的Container GC日志，如果频繁出现Full GC，需要优化GC。



GC的配置：在客户端的“conf/flink-conf.yaml”配置文件中，在“env.java.opts”配置项中添加参数：“-Xloggc:<LOG\_DIR>/gc.log -XX:+PrintGCDetails -XX:-OmitStackTraceInFastThrow -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=20 -XX:GCLogFileSize=20M”。 此处默认已经添加GC日志。

操作步骤

* 优化GC。

调整老年代和新生代的比值。在客户端的“conf/flink-conf.yaml”配置文件中，在“env.java.opts”配置项中添加参数：“-XX:NewRatio”。如“ -XX:NewRatio=2”，则表示老年代与新生代的比值为2:1，新生代占整个堆空间的1/3，老年代占2/3。

* 开发Flink应用程序时，优化DataStream的数据分区或分组操作。
* 当分区导致数据倾斜时，需要考虑优化分区。
* 避免非并行度操作，有些对DataStream的操作会导致无法并行，例如WindowAll。
* keyBy尽量不要使用String。

### 设置并行度

操作场景

并行度控制任务的数量，影响操作后数据被切分成的块数。调整并行度让任务的数量和每个任务处理的数据与机器的处理能力达到最优。

查看CPU使用情况和内存占用情况，当任务和数据不是平均分布在各节点，而是集中在个别节点时，可以增大并行度使任务和数据更均匀的分布在各个节点。增加任务的并行度，充分利用集群机器的计算能力，一般并行度设置为集群CPU核数总和的2-3倍。

操作步骤

任务的并行度可以通过以下四种层次（按优先级从高到低排列）指定，用户可以根据实际的内存、CPU、数据以及应用程序逻辑的情况调整并行度参数。

* 算子层次

一个算子、数据源和sink的并行度可以通过调用setParallelism()方法来指定，例如

final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();   
   
DataStream<String> text = [...]   
DataStream<Tuple2<String, Integer>> wordCounts = text   
 .flatMap(new LineSplitter())   
 .keyBy(0)   
 .timeWindow(Time.seconds(5))   
 .sum(1).setParallelism(5);   
   
wordCounts.print();   
   
env.execute("Word Count Example");

* 执行环境层次

Flink程序运行在执行环境中。执行环境为所有执行的算子、数据源、data sink定义了一个默认的并行度。

执行环境的默认并行度可以通过调用setParallelism()方法指定。例如：

final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();   
env.setParallelism(3);   
DataStream<String> text = [...]   
DataStream<Tuple2<String, Integer>> wordCounts = [...]   
wordCounts.print();   
env.execute("Word Count Example");

* 客户端层次

并行度可以在客户端将job提交到Flink时设定。对于CLI客户端，可以通过“-p”参数指定并行度。例如：

./bin/flink run -p 10 ../examples/\*WordCount-java\*.jar

* 系统层次

在系统级可以通过修改Flink客户端conf目录下的“flink-conf.yaml”文件中的“parallelism.default”配置选项来指定所有执行环境的默认并行度。

### 配置进程参数

操作场景

Flink on YARN模式下，有JobManager和TaskManager两种进程。在任务调度和运行的过程中，JobManager和TaskManager承担了很大的责任。

因而JobManager和TaskManager的参数配置对Flink应用的执行有着很大的影响意义。用户可通过如下操作对Flink集群性能做优化。

操作步骤

配置JobManager内存。

JobManager负责任务的调度，以及TaskManager、RM之间的消息通信。当任务数变多，任务平行度增大时，JobManager内存都需要相应增大。

您可以根据实际任务数量的多少，为JobManager设置一个合适的内存。

* 在使用yarn-session命令时，添加“-jm MEM”参数设置内存。
* 在使用yarn-cluster命令时，添加“-yjm MEM”参数设置内存。

配置TaskManager个数。

每个TaskManager每个核同时能跑一个task，所以增加了TaskManager的个数相当于增大了任务的并发度。在资源充足的情况下，可以相应增加TaskManager的个数，以提高运行效率。

* 在使用yarn-session命令时，添加“-n NUM”参数设置TaskManager个数。
* 在使用yarn-cluster命令时，添加“-yn NUM”参数设置TaskManager个数。

配置TaskManager Slot数。

每个TaskManager多个核同时能跑多个task，相当于增大了任务的并发度。但是由于所有核共用TaskManager的内存，所以要在内存和核数之间做好平衡。

* 在使用yarn-session命令时，添加“-p NUM”参数设置SLOT数。
* 在使用yarn-cluster命令时，添加“-yp NUM”参数设置SLOT数。

配置TaskManager内存。

TaskManager的内存主要用于任务执行、通信等。当一个任务很大的时候，可能需要较多资源，因而内存也可以做相应的增加。

* 将在使用yarn-sesion命令时，添加“-tm MEM”参数设置内存。
* 将在使用yarn-cluster命令时，添加“-ytm MEM”参数设置内存。

----结束

### 设计分区方法

操作场景

合理的设计分区依据，可以优化task的切分。在程序编写过程中要尽量分区均匀，这样可以实现每个task数据不倾斜，防止由于某个task的执行时间过长导致整个任务执行缓慢。

操作步骤

以下是几种分区方法。

* **随机分区：**将元素随机地进行分区。

dataStream.shuffle();

* **Rebalancing (Round-robin partitioning)：**基于round-robin对元素进行分区，使得每个分区负责均衡。对于存在数据倾斜的性能优化是很有用的。

dataStream.rebalance();

* **Rescaling：**以round-robin的形式将元素分区到下游操作的子集中。如果你想要将数据从一个源的每个并行实例中散发到一些mappers的子集中，用来分散负载，但是又不想要完全的rebalance 介入（引入`rebalance()`），这会非常有用。

dataStream.rescale();

* **广播：**广播每个元素到所有分区。

dataStream.broadcast();

* **自定义分区：**使用一个用户自定义的Partitioner对每一个元素选择目标task，由于用户对自己的数据更加熟悉，可以按照某个特征进行分区，从而优化任务执行。

简单示例如下所示：

// fromElements构造简单的Tuple2流   
DataStream<Tuple2<String, Integer>> dataStream = env.fromElements(Tuple2.of("hello",1), Tuple2.of("test",2), Tuple2.of("world",100));   
   
// 定义用于分区的key值，返回即属于哪个partition的，该值加1就是对应的子任务的id号   
Partitioner<Tuple2<String, Integer>> strPartitioner = new Partitioner<Tuple2<String, Integer>>() {   
 @Override   
 public int partition(Tuple2<String, Integer> key, int numPartitions) {   
 return (key.f0.length() + key.f1) % numPartitions;   
 }   
};   
   
// 使用Tuple2进行分区的key值   
dataStream.partitionCustom(strPartitioner, new KeySelector<Tuple2<String, Integer>, Tuple2<String, Integer>>() {   
 @Override   
 public Tuple2<String, Integer> getKey(Tuple2<String, Integer> value) throws Exception {   
 return value;   
 }   
}).print();

### 配置netty网络通信

操作场景

Flink通信主要依赖netty网络，所以在Flink应用执行过程中，netty的设置尤为重要，网络通信的好坏直接决定着数据交换的速度以及任务执行的效率。

操作步骤

以下配置均可在客户端的“conf/flink-conf.yaml”配置文件中进行修改适配，默认已经是相对较优解，请谨慎修改，防止性能下降。

* “taskmanager.network.netty.num-arenas”： 默认是“taskmanager.numberOfTaskSlots”，表示netty的域的数量。
* “taskmanager.network.netty.server.numThreads”和“taskmanager.network.netty.client.numThreads”：默认是“taskmanager.numberOfTaskSlots”，表示netty的客户端和服务端的线程数目设置。
* “taskmanager.network.netty.client.connectTimeoutSec”：默认是120s，表示taskmanager的客户端连接超时的时间。
* “taskmanager.network.netty.sendReceiveBufferSize”：默认是系统缓冲区大小(cat /proc/sys/net/ipv4/tcp\_[rw]mem) ，一般为4MB，表示netty的发送和接收的缓冲区大小。
* “taskmanager.network.netty.transport”：默认为“nio”方式，表示netty的传输方式，有“nio”和“epoll”两种方式。

### 经验总结

数据倾斜

当数据发生倾斜（某一部分数据量特别大），虽然没有GC（Gabage Collection，垃圾回收），但是task执行时间严重不一致。

* 需要重新设计key，以更小粒度的key使得task大小合理化。
* 修改并行度。
* 调用rebalance操作，使数据分区均匀。

缓冲区超时设置

* 由于task在执行过程中存在数据通过网络进行交换，数据在不同服务器之间传递的缓冲区超时时间可以通过setBufferTimeout进行设置。
* 当设置“setBufferTimeout(-1)”，会等待缓冲区满之后才会刷新，使其达到最大吞吐量；当设置“setBufferTimeout(0)”时，可以最小化延迟，数据一旦接收到就会刷新；当设置“setBufferTimeout”大于0时，缓冲区会在该时间之后超时，然后进行缓冲区的刷新。

示例可以参考如下：

env.setBufferTimeout(timeoutMillis);   
   
env.generateSequence(1,10).map(new MyMapper()).setBufferTimeout(timeoutMillis);

# 性能瓶颈监控及调优

[4.1 常见性能问题及解决方案](#_ZH-CN_TOPIC_0067016237)

## 常见性能问题及解决方案

### 监控手段

#### Flink日志概述

Flink服务的日志主要存放在“$BigdataLogHome/flink”路径下，默认在“/var/log/Bigdata/flink”路径下。

组件服务日志主要是安装时FusionInsight对Flink的适配任务，例如HDFS上文件目录的创建、文件权限设置等。

| 日志名称 | 日志说明 |
| --- | --- |
| postinstall.log | flinkResource预安装的日志文件 |
| prestart.log | flinkResource预启动的日志文件 |

#### 任务信息收集

通常分析一个Flink任务的运行信息，主要需要如下几个信息：

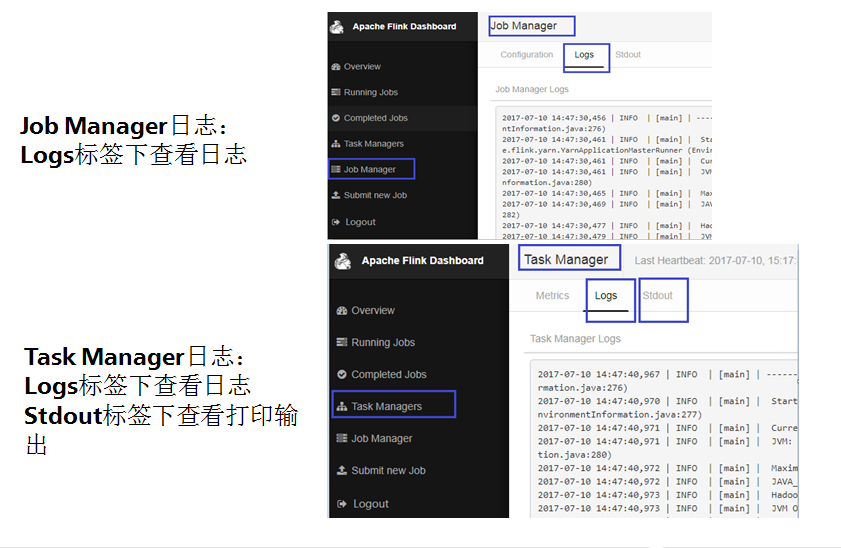
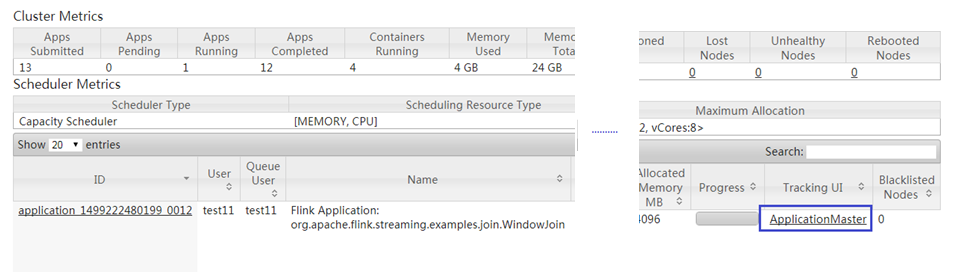
* JobManager日志：主要记录资源申请，task的分配调度等。
* TaskManager日志：主要记录task的具体执行情况。
* NodeManager（NM）日志：主要记录与TaskManager交互，监控TaskManager信息。
* ResourceManager（RM）日志：主要记录与TaskManager，JobManager的交互。

任务运行过程中

在任务运行过程中，任务的信息查看比较方便：

* WebUI日志查看：提交Flink任务后，如果想查询Flink任务的运行情况。单击“服务管理 > Yarn > ResourceManager(主)”，打开YARN的WebUI界面。点击该任务的ApplicationMaster超链接，打开Flink运行日志界面。

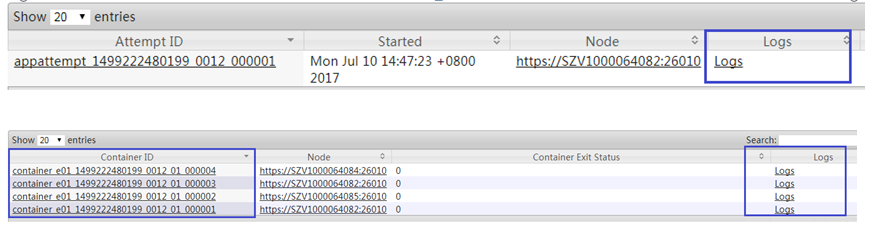
运行任务过程中WebUI日志查看



* Yarn页面 日志查看：Flink任务处于运行时：提交Flink任务后，通过Yarn页面的AM日志查看JobManager日志。单击“服务管理 > Yarn > ResourceManager(主)”，打开YARN的WebUI界面。

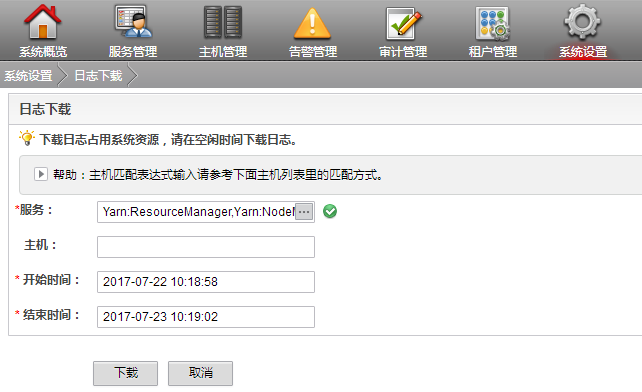
点击该任务的Application ID超链接，点击“Logs”打开Yarn的AM日志界面，或者点击“Attempt ID > container对应的Logs”打开Yarn的Container日志界面。

Yarn页面查看日志



* ResourceManager/NodeManager日志查看与收集：在FusionInsight Manager页面，单击“系统设置 > 日志下载”，在出现的页面中，在“服务”栏选择“ResourceManager”、“NodeManager”等，选择获取日志的开始和结束时间，点击“下载”，即可下载获取相关日志，如图4-3所示。

下载日志



任务运行结束后

* Contaienr日志查看：默认情况下，Container日志在任务运行完后，会自动归集到HDFS上。可从HDFS上获取Container 日志。命令：**hdfs dfs -get /tmp/logs/username/logs/appId /path** ，其中*username* 为提交任务的用户名。如admin 或flink；或者通过**yarn logs -applicationId appid >> yourlogfile**。如果日志归集功能没有打开，则日志会存放在本地。可到相应节点的“${yarn.nodemanager.log-dirs}”路径下查看相应日志。 默认为“/srv/BigData/hadoop/data1/nm/containerlogs”。
* RM/NM日志的获取同运行过程中获取方法一致。

Flink任务与其他组件也有较多交互，如Kafka、ZooKeeper等组件，在涉及到这些组件时，也需要查看相关组件日志协助分析相关任务。

通常组件的日志信息获取，可通过与去RM/NM日志同样的方式。 在分析Flink任务的过程中，由于业务与Flink框架是耦合的，在任务运行出现问题时，通常也需要根据用户的业务逻辑、执行流程、数据情况等来具体分析。 在一些任务运行如出现某个Container运行较慢，其他Container运行正常情况下，可以到相应节点上分析该Container 的进程状态，可通过如图4-2中的信息找到Container对应的host，然后在该节点使用**ps -ef | grep appid | grep -v bash | grep java**等命令查找出对应container ID。使用如下命令如**jmap**查看进程中每个对象占用的内存空间，**jstack** 查看进程中某个现场的运行状态。Flink任务运行的Container日志由Yarn管理。

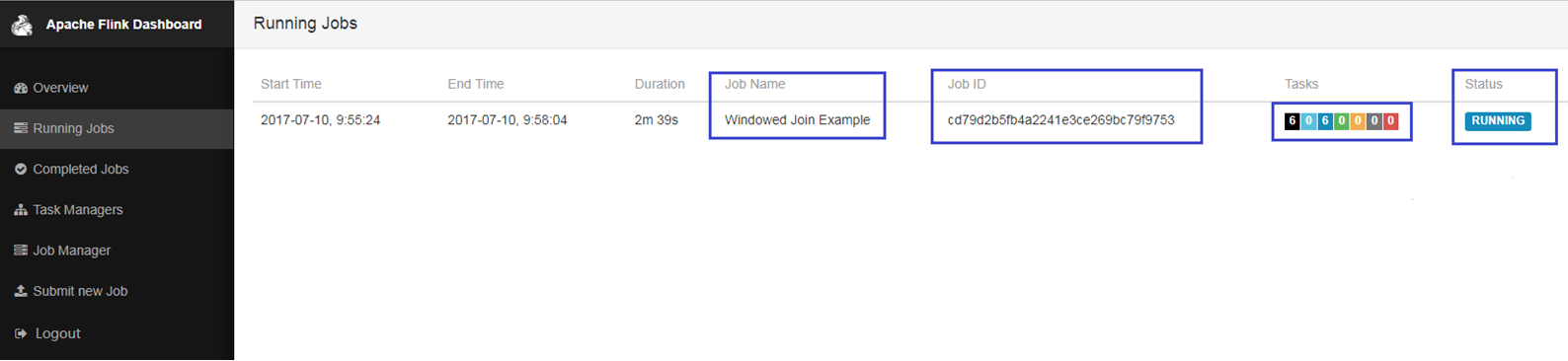
* Container日志清理：

Container日志会定期清理。分别由如下参数控制： “yarn.log-aggregation.retain-check-interval-seconds”、“yarn.log-aggregation.retain-seconds”。

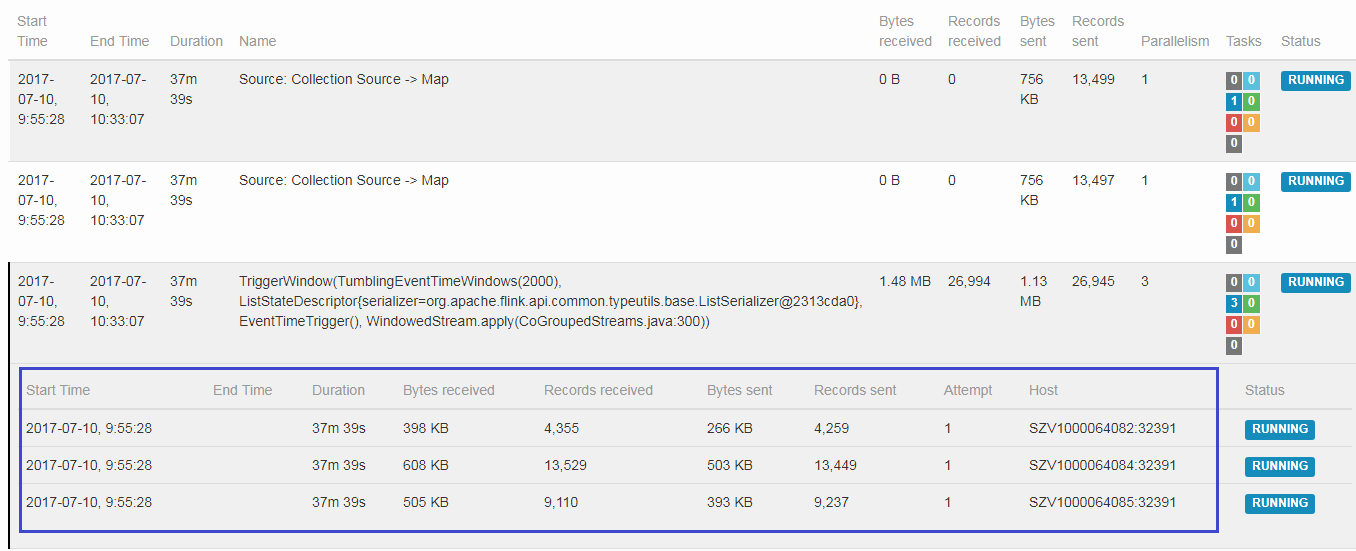
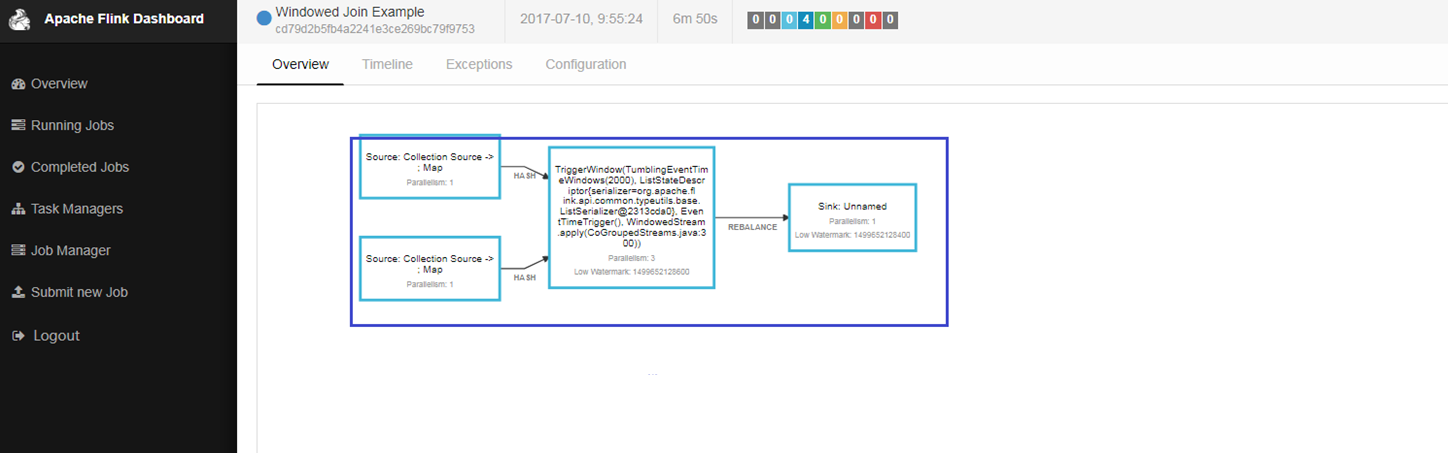
#### 监控页面

Flink运行时可通过WebUI监控应用运行的各个信息，如Job运行时间， task运行情况，内存情况等。

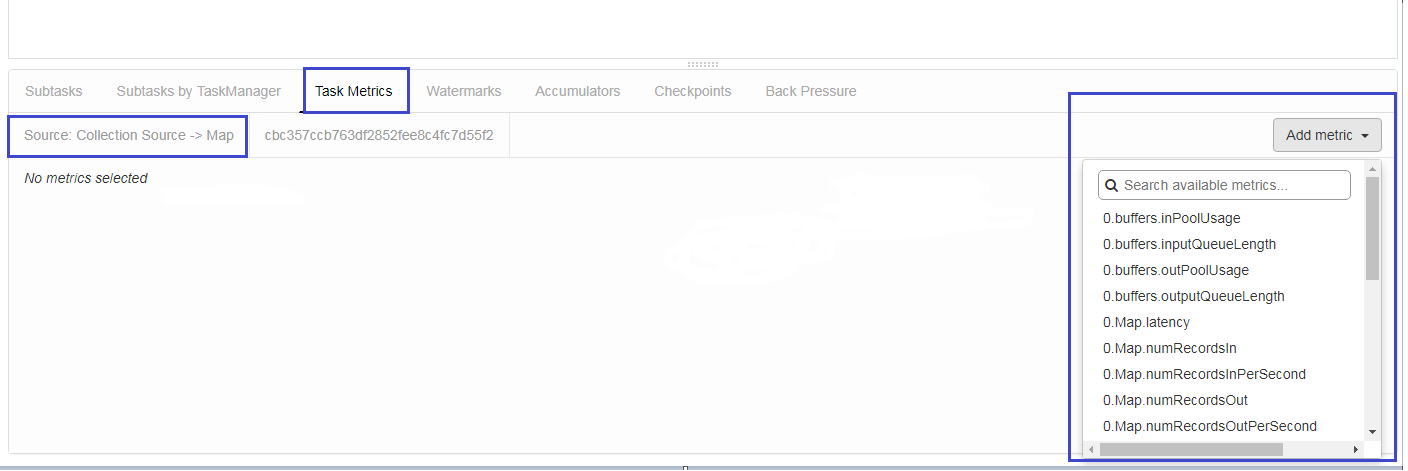
* job运行



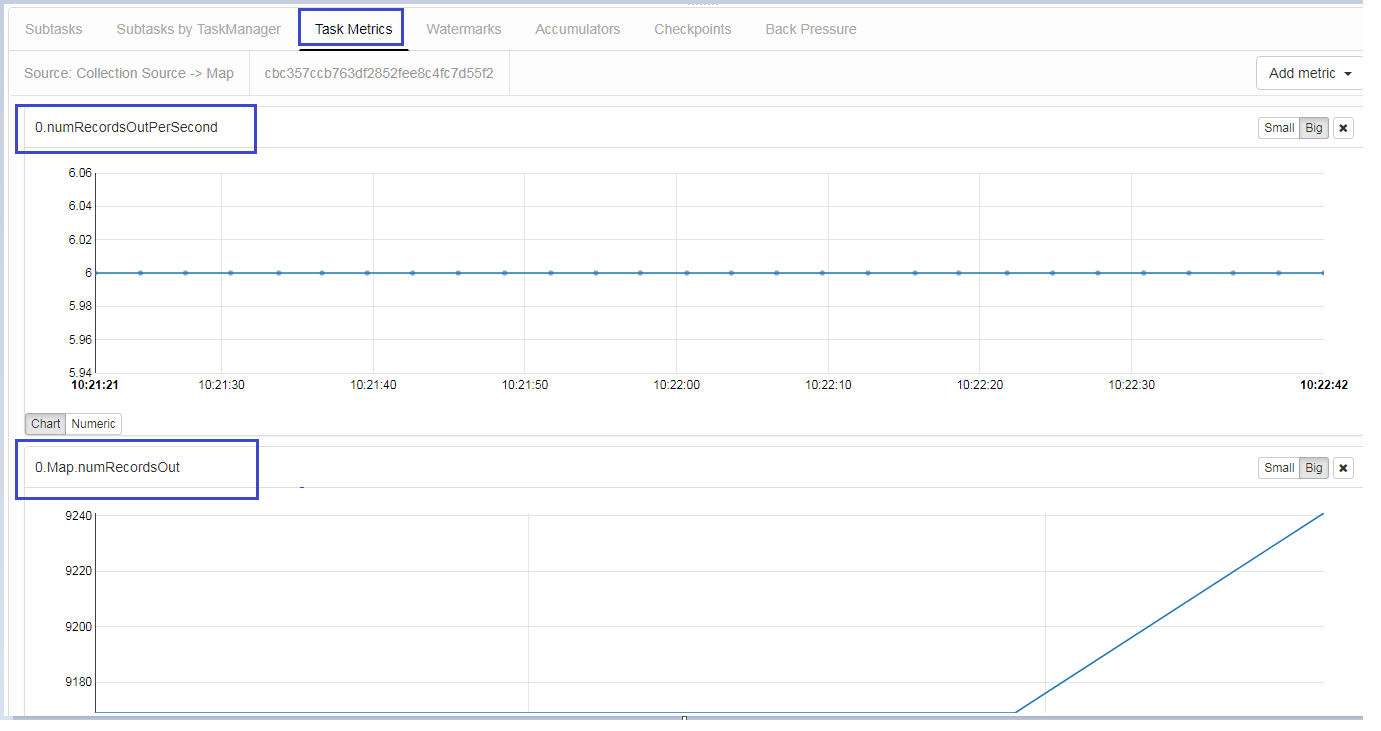
* 流图及算子概览



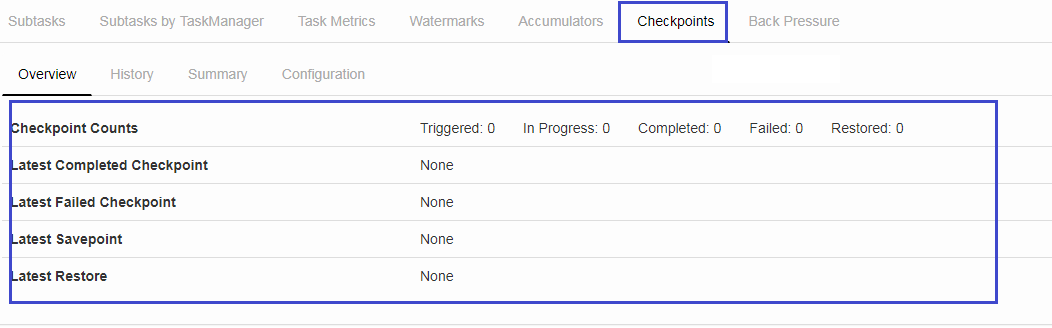
* Task metric信息



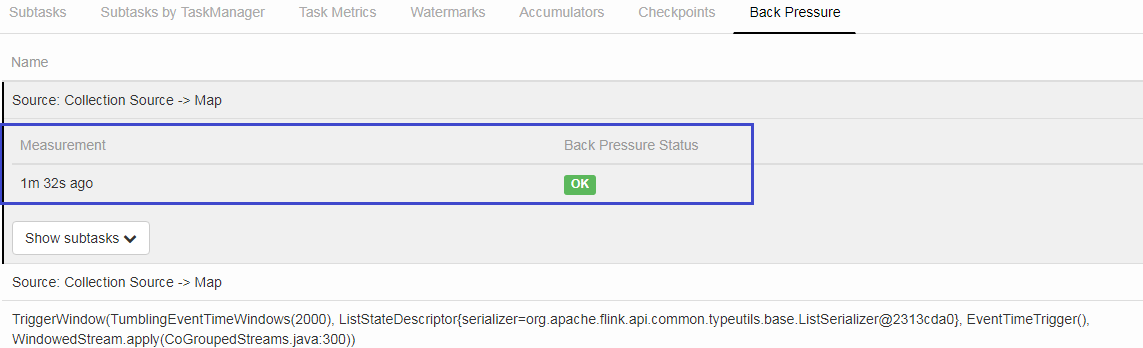
* 勾选之后显示图或者数据



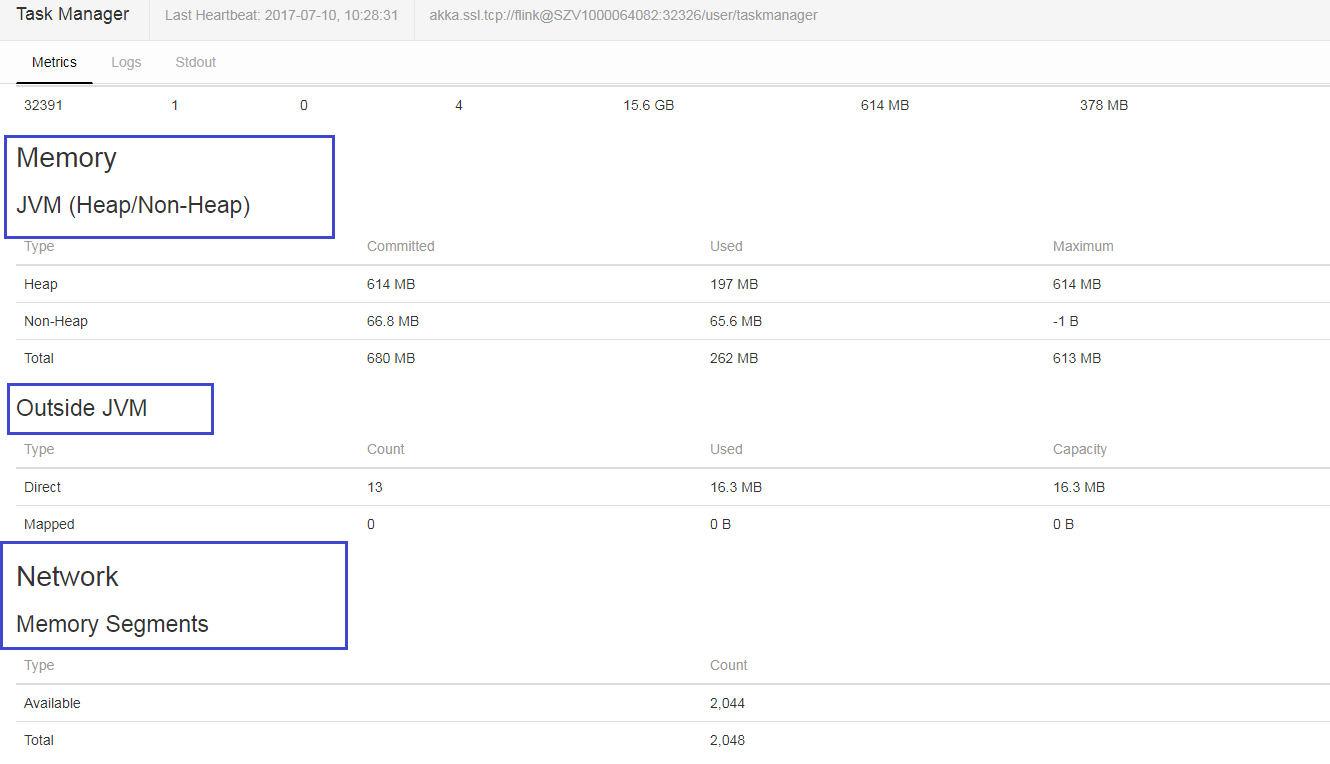
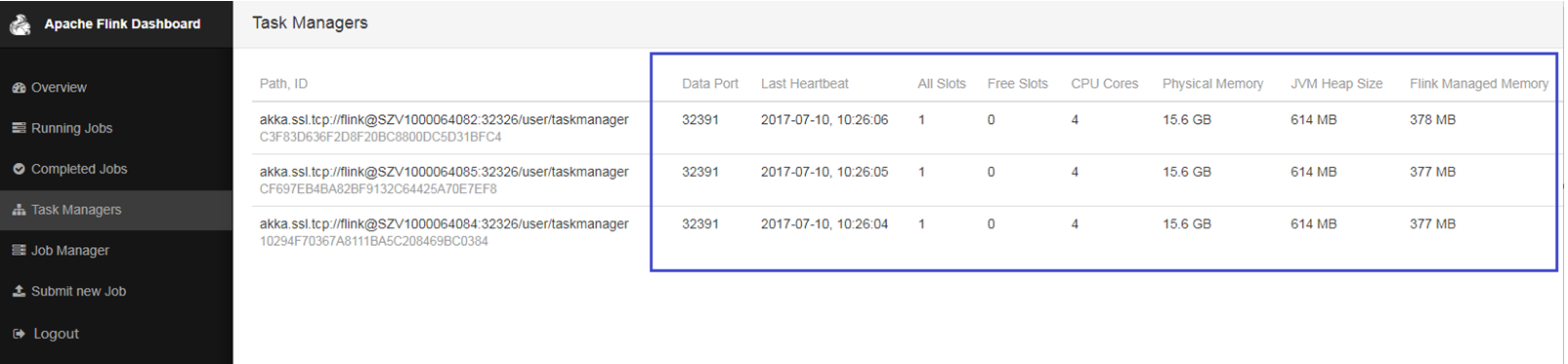
* Checkpoint信息



* 压力取样测试



* TaskManager信息



* Jobmanager的配置信息

