

前言

Flink 是目前最流行的大数据及流式计算框架之一，用户可以使用 Java/Scala/Python 的 `DataStream` 接口或者标准 SQL 语言来快速实现一个分布式高可用的流式应用，通过内部的 Java JIT、off-heap 内存管理等技术优化性能，并且有完整的 Source、Sink、WebUI、Metrics 等功能集成，让 Flink 几乎成为了流式计算的事实标准。

但是当处理海量数据的时候，很容易出现各种异常和性能瓶颈，这时我们需要优化系统性能时，常常需要分析程序运行行为和性能瓶颈。Profiling 技术是一种在应用运行时收集程序相关信息的动态分析手段，常用的 JVM Profiler 可以从多个方面对程序进行动态分析，如 CPU、Memory、Thread、Classes、GC 等，其中 CPU Profiling 的应用最为广泛。CPU Profiling 经常被用于分析代码的执行热点，如“哪个方法占用 CPU 的执行时间最长”、“每个方法占用 CPU 的比例是多少”等等，通过 CPU Profiling 得到上述相关信息后，研发人员就可以轻松针对热点瓶颈进行分析和性能优化，进而突破性能瓶颈，大幅提升系统的吞吐量。

本文介绍我们在做性能优化常用的火焰图以及为如何集成火焰图到通用的 Flink 作业中。

火焰图介绍

火焰图是《性能之巅》作者以及 DTrace 等一系列 Linux 系统优化工具作者 Brendan Gregg 大神的作品之一，可以非常清晰地展示应用程序的函数调用栈以及函数调用时间占比，基本原理是通过各种 agent 在程序运行时采样并输出日志，使用 FlameGraph 工具把日志提取出来输出可在浏览器交互式查看的 SVG 图片。

Uber 开源了 `jvm-profiler` 项目，介绍如何为 Spark 应用和 Java 应用添加火焰图支持，但是目前 Flink 社区和 `jvm-profiler` 官网都还没有相关的使用教程。

Example to Run with Spark Application

You could upload jvm-profiler jar file to HDFS so the Spark application executors could access it. Then add configuration like following when launching Spark application:

```
--conf spark.jars=hdfs://hdfs_url/lib/jvm-profiler-1.0.0.jar
--conf spark.executor.extraJavaOptions=-javaagent:jvm-profiler-1.0.0.jar
```

Example to Run with Java Application

Following command will start the example application with the profiler agent attached, which will report metrics to the console output:

```
java -javaagent:target/jvm-profiler-1.0.0.jar=reporter=com.uber.profiling.reporters.ConsoleOutputRep
```

Example to Run with Executable Jar

Use following command to run jvm profiler with executable jar application.

```
java -javaagent:/opt/jvm-profiler/target/jvm-profiler-1.0.0.jar=reporter=com.uber.profiling.reporters
```

Example to Run with Tomcat

Set the jvm profiler in CATALINA_OPTS before starting the tomcat server. Check logs/catalina.out file for metrics.

```
export CATALINA_OPTS="$CATALINA_OPTS -javaagent:/opt/jvm-profiler/target/jvm-profiler-1.0.0.jar=rep
```

Example to Run with Spring Boot Maven Plugin

实际上基于 JVM 的程序都可以使用这个工具，本文将基于 jvm-profiler 来介绍如何生成 Flink 作业的火焰图。

下载和编译 jvm-profiler

```
1 | git clone https://github.com/uber-common/jvm-profiler.git
2 |
3 | mvn clean install -DskipTests=true -Dcheckstyle.skip -Dfast -T 8C
```

编译好了之后，将项目 target 目录下的 jvm-profiler-1.0.0.jar 复制一份到 flink 的 lib 目录下面

```
1 | cp target/jvm-profiler-1.0.0.jar /usr/local/flink-1.11.1/lib
```

下载 FlameGraph

由于 jvm-profiler 支持生成火焰图需要的日志文件，将日志转成交互式 SVG 图片还是使用 Brendan Gregg 的 FlameGraph 工具。

```
1 | git clone https://github.com/brendangregg/FlameGraph.git
```

下载项目源码即可，后面会使用 flamegraph.pl 工具来生成图片文件。

配置 Flink

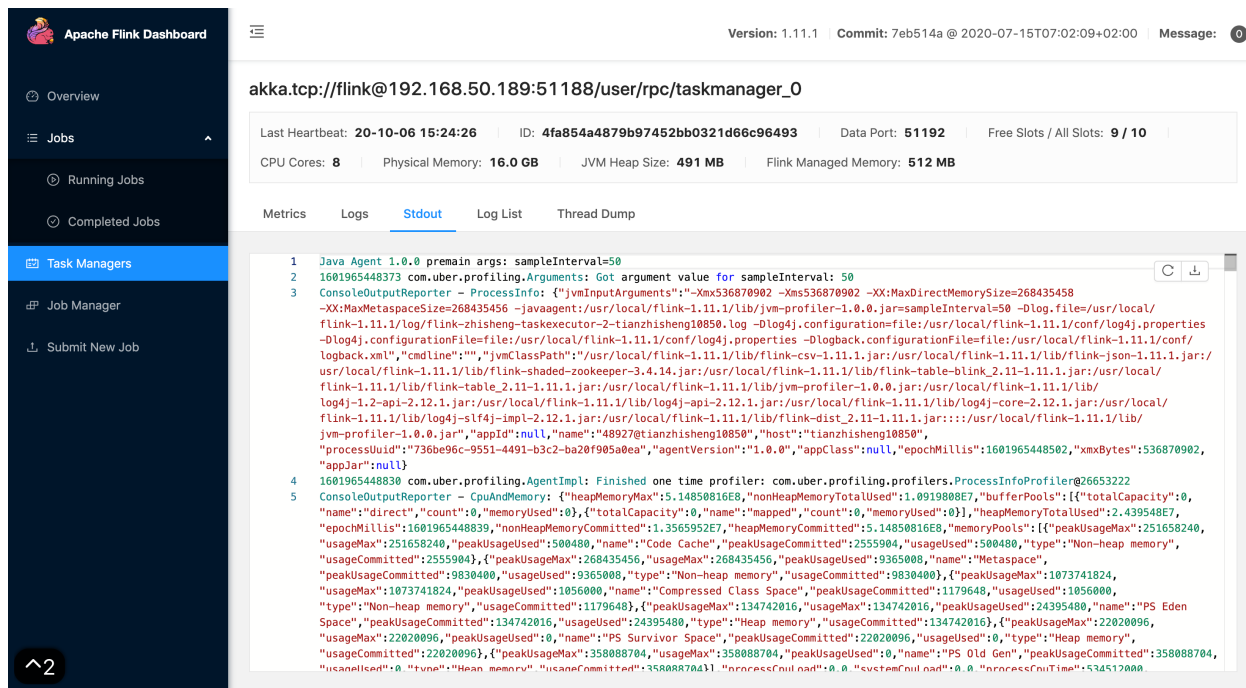
对于 Flink 应用，我们只需要在 TaskManager 中注入打点的 Java agent 即可，这里测试，我就使用本地 standalone 模式，修改 Flink conf 目录下的 flink-conf.yaml 文件，添加一下如下配置：

```
1 env.java.opts.taskmanager: "-javaagent:/usr/local/flink-1.11.1/lib/jvm-  
  profiler-1.0.0.jar=sampleInterval=50"
```

目前最小的采样间隔就是 50 毫秒，然后启动集群和运行一个 Flink 作业：

```
1 ./bin/start-cluster.sh  
2  
3  
4 //运行一个作业  
5 ./bin/flink run ./examples/streaming/StateMachineExample.jar
```

运行之后可以看到 TaskManager 的 stdout 里面打印如下：



The screenshot shows the Apache Flink Dashboard interface. The sidebar on the left contains navigation links: Overview, Jobs, Task Managers (selected), and Job Manager. The main panel displays the Task Manager's status and logs. The status bar at the top shows the version (1.11.1), commit (7eb514a), and message. The Task Manager's status is shown as akka.tcp://flink@192.168.50.189:51188/user/rpc/taskmanager_0. The status bar includes metrics: Last Heartbeat (20-10-06 15:24:26), ID (4fa854a4879b97452bb0321d66c96493), Data Port (51192), Free Slots / All Slots (9 / 10), CPU Cores (8), Physical Memory (16.0 GB), JVM Heap Size (491 MB), and Flink Managed Memory (512 MB). The Stdout tab is selected, showing the following logs:

```
1 Java Agent 1.0.0 premain args: sampleInterval=50  
2 1601965448373 com.uber.profiling.Arguments: Got argument value for sampleInterval: 50  
3 ConsoleOutputReporter - ProcessInfo: {"jvmInputArguments":{"Xmx536870902 -Xms536870902 -XX:MaxDirectMemorySize=268435458  
  -XX:MaxMetaspaceSize=268435456 -javaagent:/usr/local/flink-1.11.1/lib/jvm-profiler-1.0.0.jar=sampleInterval=50 -Dlog.file=/usr/local/  
  flink-1.11.1/log/flink-zhisheng-taskexecutor-2-tianzhisheng10850.log -Dlog4j.configuration=file:/usr/local/flink-1.11.1/conf/log4j.properties  
  -Dlog4j.configurationFile=file:/usr/local/flink-1.11.1/conf/log4j.properties -Dlogback.configurationFile=file:/usr/local/flink-1.11.1/conf/  
  logback.xml","cmdline":"","jvmClassPath":"/usr/local/flink-1.11.1/lib/flink-csv-1.11.1.jar:/usr/local/flink-1.11.1/lib/flink-json-1.11.1.jar:/  
  usr/local/flink-1.11.1/lib/flink-shaded-zookeeper-3.4.14.jar:/usr/local/flink-1.11.1/lib/flink-table-blink-2.11-1.11.1.jar:/usr/local/  
  flink-1.11.1/lib/flink-table-2.11-1.11.1.jar:/usr/local/flink-1.11.1/lib/jvm-profiler-1.0.0.jar:/usr/local/flink-1.11.1/lib/  
  log4j-1.2-api-2.12.1.jar:/usr/local/flink-1.11.1/lib/log4j-api-2.12.1.jar:/usr/local/flink-1.11.1/lib/log4j-core-2.12.1.jar:/usr/local/  
  flink-1.11.1/lib/log4j-slf4j-impl-2.12.1.jar:/usr/local/flink-1.11.1/lib/flink-dist-2.11-1.11.1.jar:/usr/local/flink-1.11.1/lib/  
  jvm-profiler-1.0.0.jar"},"appId":null,"name":"48927@tianzhisheng10850","host":"tianzhisheng10850",  
  "processUid":"736be96c-9551-4491-b3c2-ba20f985a0ea","agentVersion":"1.0.0","appClass":null,"epochMillis":1601965448502,"mxmBytes":536870902,  
  "appJar":null}  
4 1601965448830 com.uber.profiling.AgentImpl: Finished one time profiler: com.uber.profiling.profilers.ProcessInfoProfiler@26653222  
5 ConsoleOutputReporter - CpuAndMemory: {"heapMemoryMax":5.14850816E8,"nonHeapMemoryTotalUsed":1.0919808E7,"bufferPools":[{"totalCapacity":0,  
  "name":"direct","count":0,"memoryUsed":0}, {"totalCapacity":0,"name":"mapped","count":0,"memoryUsed":0}], "heapMemoryTotalUsed":2.439548E7,  
  "epochMillis":1601965448830,"nonHeapMemoryCommitted":1.3565952E7,"heapMemoryCommitted":5.14850816E8,"memoryPools":[{"peakUsageMax":251658240,  
  "usageMax":251658240,"peakUsageUsed":500480,"name":"Code Cache","peakUsageCommitted":2555904,"usageUsed":500480,"type":"Non-heap memory",  
  "usageCommitted":2555904,"peakUsageMax":268435456,"usageMax":268435456,"peakUsageUsed":9365008,"name":"Metaspace",  
  "peakUsageCommitted":9830400,"usageUsed":9365008,"type":"Non-heap memory","usageCommitted":9830400,"peakUsageMax":1073741824,  
  "usageMax":1073741824,"peakUsageUsed":1056000,"name":"Compressed Class Space","peakUsageCommitted":1179648,"usageUsed":1056000,  
  "type":"Non-heap memory","usageCommitted":1179648,"peakUsageMax":134742016,"usageMax":134742016,"peakUsageUsed":24395480,"name":"PS Eden  
  Space","peakUsageCommitted":134742016,"usageUsed":24395480,"type":"Heap memory","usageCommitted":134742016,"peakUsageMax":22020096,  
  "usageMax":22020096,"peakUsageUsed":0,"name":"PS Survivor Space","peakUsageCommitted":22020096,"usageUsed":0,"type":"Heap memory",  
  "usageCommitted":22020096,"peakUsageMax":358088704,"usageMax":358088704,"peakUsageUsed":0,"name":"PS Old Gen","peakUsageCommitted":358088704,  
  "usageUsed":0,"type":"Heap memory","usageCommitted":358088704,"processCpuLoad":0.0,"systemCpuLoad":0.0,"processCpuTime":534517000}
```

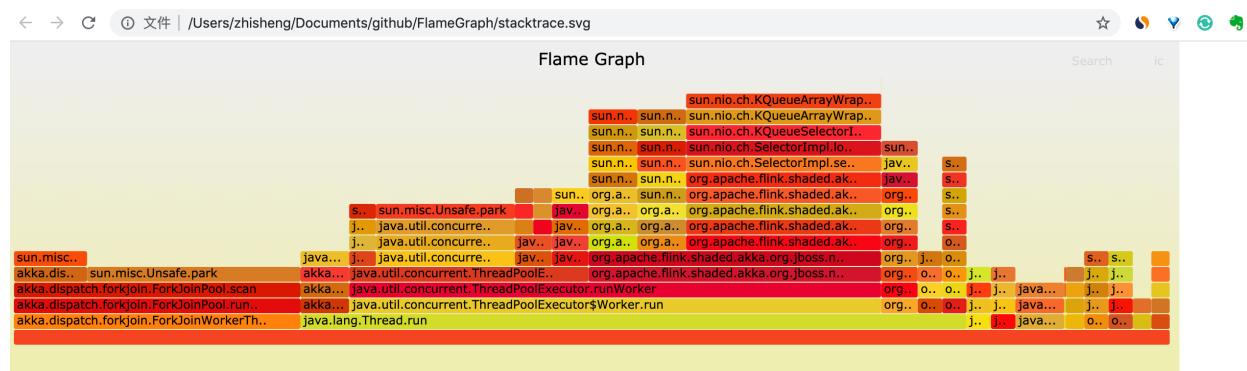
因为已经注入 Java agent，因此在标准输出中会定期添加火焰图所需要的打点数据，然后使用下面的命令提取相关日志，并且使用 jvm-profiler 和 FlameGraph 提供的工具来生成 SVG 图片文件。

```

1 //1、提取 stdout 文件中的相关日志
2
3 cat log/flink-zhisheng-taskexecutor-0-zhisheng.out | grep
  "ConsoleOutputReporter - Stacktrace:" | awk '{print substr($0,37)}' >
  stacktrace.json
4
5
6 //2、在 jvm-profiler 目录下执行下面命令
7
8 python ./stackcollapse.py -i /usr/local/flink-1.11.1/stacktrace.json >
  stacktrace.folded
9
10
11 //3、在 FlameGraph 目录下执行下面命令生成 SVG 图片
12
13 ./flamegraph.pl /Users/zhisheng/Documents/github/jvm-
  profiler/stacktrace.folded > stacktrace.svg

```

然后用浏览器打开刚才生成的 SVG 图片就可以看到火焰图信息。



总结

本文主要目的在于教大家如何利用 jvm-profiler 去生成 Flink 作业的运行火焰图，这样可以在遇到性能瓶颈问题的时候会很方便大家去定位问题，关于如何去读懂生成的火焰图，后面可以再分享系列文章。

参考资料

- [JVM CPU Profiler技术原理及源码深度解析](#)
- [jvm-profile](#)