基于Flink的SMaRT应用的性能优化-典型的响应式应用程序的可靠方案

Apache Flink是一种用于对无边界和有边界的数据流进行有状态计算的分布式处理引擎。 Flink具有流式计算、可靠性高、可扩展性强、高性能等优点。Flink最适合的应用场景是低延时的数据处理场景：高并发处理数据，实现毫秒级，且兼具可靠性。正是基于上述这一特性。SMaRT采用Flink框架作为自己的核心计算单元。

SMaRT是一款全新的运行在Flink平台上中央化的监管系统。新的SMaRT能够提供毫秒级的响应请求服务能力。性能问题（响应时间）是衡量SMaRT成功的一个关键因子。SMaRT团队对SMaRT系统进行全方位的性能优化，包括JVM调优、Kafka调优、Flink调优、数据库调优，甚至架构调优。本文主要基于SMaRT计算层的调优：Flink平台调优。

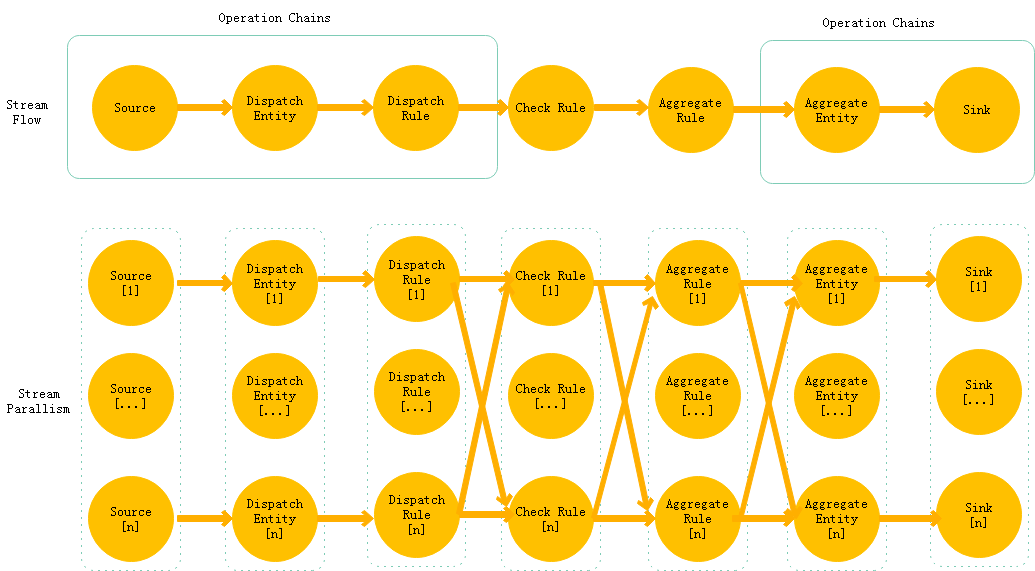
关于Flink性能优化，SMaRT主要使用了两个方面：1.并行度调整以及相关的操作链化。2.网络缓冲区调优。本文将介绍相关的原理以及SMaRT如何合理利用它们来提供性能。

1. 并行度优化及操作链化

采用分布式设计思想优化系统通常有两个思路：数据切割：多个类似的数据拆分出独立的数据；单任务拆分：即大而繁重的任务拆分成多个轻量级的子任务。这两种思路可以通过增加计算资源来增加并行度从而达到快速处理和快速响应。Flink平台的并行度因子正好对应了思路一。Flink平台的流式处理也对应了思路二。

为了更高效地分布式并行执行，Flink也会尽可能地将操作的子任务链接在一起形成一个新的任务。新的生成的任务在一个task Manager中的一个线程中执行。将Flink操作链接成成一个的任务是一个非常有效的优化手段：它能减少线程之间的切换，减少消息的序列化/反序列化，减少数据在缓冲区的交换，减少了跨节点间的延迟，从而提高整体的吞吐量和响应时间。

我们以SMaRT为例，SMaRT包含有多个交易的Entity, 每个Entity包含有多个Rule。当SMaRT请求进入到Flink平台时，会先进行Source拉取、Entity拆分、Rule拆分、Rule计算、Rule合并、Entity合并、Sink推送。SMaRT依据分配的资源，设置Source拉取、Entity拆分、Rule拆分并行度为32，Rule计算并行度为65，Rule合并、Entity合并、Sink推送并行度为36

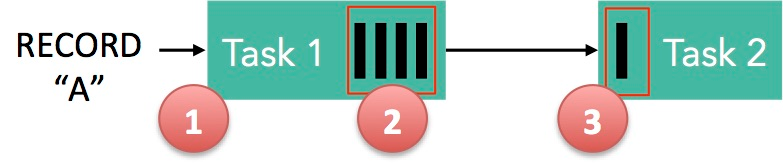


上图中Flink平台不仅将Source拉取、Entity拆分和Rule拆分三个操作进行了合并，也将Entity合并和Sink推送两个操作进行了合并。因为这些个操作合并后并不会改变整体的拓扑结构。但是并不是任意两个操作就可以链化在一起的。比如Rule合并、Entity合并这两个操作就不能组合在在一起，其条件还是很苛刻的：

1. 上下游的并行度一致
2. 下游节点的入度为1也就是说下游节点没有其他节点的输入
3. 上下游节点都在同一个 slot group 中
4. 下游节点的chain 策略为 ALWAYS（可以与上下游链接，map、flatmap、filter等默认是ALWAYS）
5. 上游节点的chain策略为 ALWAYS 或 HEAD（只能与下游链接，不能与上游链接，Source默认是HEAD）
6. 两个节点间的数据分区方式是 forward
7. 用户没有禁用chain
8. 缓冲区优化

Flink网络组件是Flink Runtime模块的核心组件之一，并且在Flink上承担的核心功能。它连接来自所有TaskManager的各个工作单元。网络组件控制数据流入、流出的方向，因此衡量应用程序的吞吐量和延迟，对于Flink作业的性能至关重要。本文将深入了解流式操作的物理实现和Flink所做的各种优化。本文将简要介绍这些优化的结果以及Flink在吞吐量和延迟之间的权衡。

下图展示了Flink TaskManager之间数据传输的视图，其中包括网络组件及其周围组件的更多详细信息，从发送操作的记录到接受操作的整个过程。



Task 1 在输出端有一个相关联的本地缓冲池（称缓冲池1），Task 2 在输入端也有一个相关联的本地缓冲池（称缓冲池2）。如果缓冲池1中有空闲可用的内存来序列化记录 “A”，我们就序列化并发送该缓冲区里的数据。

这里我们需要注意两个场景：

本地传输：如果 Task 1 和 Task 2运行在同一个 Task Manager节点，该缓冲区中的数据直接交给下一个Task。此场景就是SMaRT操作链式的优化方案。

远程传输：如果 Task 1 和 Task 2运行在不同的Task Manager节点上，那么缓冲区会在发送到网络（TCP Channel）后被回收。在接收端，会从本地缓冲区中申请内存，然后拷贝网络中的数据到缓冲区中。如果没有可用的缓冲区，会停止从 TCP 连接中读取数据。

这种固定大小缓冲池就像阻塞队列一样，保证了 Flink 有一套健壮的反压机制，使得 Task 生产数据的速度不会快于消费的速度。我们上面描述的这个方案可以从两个 Task 之间的数据传输自然地扩展到更复杂的 pipeline 中，保证反压机制可以扩散到整个 pipeline。目前SMaRT没有此背压问题。

Flink网络组件总是首先将处理后的结果先放入到的缓冲区中。但是这些数据何时真正发送到网络中是不确定。显然，每当缓冲区中有数据时就立即发送到网络中不是一个很好地策略，因为这不仅会由于跨节点通信而增加大量成本。在Flink中，有三种情况缓冲区中的数据立刻发送到网络上：缓冲区已满；缓冲区超时；发送特殊事件，例如检查点障碍。缓冲区满后刷新。

Flink提供了任务级别的env.setBufferTimeout(timeoutMillis)来控制数据发送的时机。Flink 采用Buffer timeout超时策略:当数据量较少，Buffer一直没有满时，后台的 Output flusher 线程会强制地将 Buffer中的数据刷新到下游。Flink中默认 timeout 时间是 100ms，即：Buffer 中的数据要么变满时 Flush，要么最多等 100ms 也会 Flush 来保证数据不会出现很大的延迟。

timeoutMillis > 0 表示最长等待 timeoutMillis 时间，就会刷新。

timeoutMillis = 0 表示每条数据都会触发刷新，直接将数据发送到下游，相当于没有使用缓冲区。

timeoutMillis = -1 表示只有等到缓冲区满了或CheckPoint的时候，才会刷新。相当于取消了 timeout 策略

SMaRT是一种典型的用户响应式请求系统，对系统的实时性要求比较高。仅仅采用默认的Flink网络传输机制，响应时间是用户不能接受的。关于如果设置buffer Timeout 值，SMaRT做了很多的对比实验，将Buffer Timeout参数从100ms->50ms->30ms->20ms->10ms->0ms，SMaRT性能变得越来越来。考虑到吞吐量和负面影响，SMaRT最终选择了10ms作为Buffer Timeout的值。

结论

本文主要介绍了Flink平台调优的相关知识以及相关的原理实现。主要提供了两种最有效的优化手段：1. 并行度优化以及相应的操作链化，尤其是Flink的链化操作可以将多个操作组合在一起作为一个新的任务来执行，从而更加有效的使用资源， 因此本文建议尽可能多的使用此特性来优化资源；2. 网络传输的缓冲区优化，减少网络节点的数据网络延迟。对于响应时间优先的系统，降低Flink缓冲区的延时发送是一个必要操作。SMaRT正是基于上述两点优化，提供了一个毫秒级别响应请求的监管平台。