基于性能收益模型的流处理算子优化技术综述

檀国林[1,2]，张鹏[1,2]，赵静芬[1,2]，刘庆云[1,2]，李正民[1,3]，陈志鹏[1,2]

(1. 中国科学院 信息工程研究所，北京 100093；

2. 信息内容安全技术国家工程实验室，北京 100093；

3. 国家计算机网络应急技术处理协调中心，北京 100029)

摘 要：大数据移动互联网时代的到来，让我们身处数据洪流之中，数据之大使得我们对数据进行高效实时处理的需求也变得越来越迫切，促使国内外的研究团队开发出了许多流处理应用。为了提高流处理应用的性能，这些流处理应用底层实现都采用了各种各样复杂的流处理算子优化技术。本文在调研学习这些流处理应用的基础上，概括总结了其中最常见的八种流处理算子优化技术，并结合实际例子，分别从性能收益、安全条件、动态性等方面详细介绍了这些算子优化技术的特点，随后我们又探讨了算子优化和流处理应用领域进一步的研究方向。

关键词：算子优化；流处理；性能收益

Operator Optimizations for Stream Processing Based on Performance Gain: A Survey

TAN Guo-lin[1,2], ZHANG Peng[1,2], ZHAO Jing-fen[1,2], LIU Qing-yun[1,2], LI Zheng-min[1,3], CHEN Zhi-peng[1,2]

(1. Institute of Information Engineering CAS, Beijing 100093, China;

2. National Engineering Laboratory for Information Security Technologies , Beijing 100093, China

3. National Computer Network Emergency Response and Coordination Center, Beijing 100029, China)

Abstract: The arrival of the era of big data and mobile Internet let us be in data torrent. Big data make the need more and more urgent to process data efficiently and real-timely. And big data also prompt research teams around the world to develop a lot of stream processing applications. The implementations of these stream processing applications use a variety of operator optimizations. Based on the study of these stream processing applications, this paper summarizes the most common eight operator optimizations of stream processing applications. And combined with practical examples, this paper introduces the features of these operator optimizations from aspects of performance gain, safety condition and dynamic. Then we discuss the further research direction in the field of operator optimizations and stream processing.

Key words: operator optimization; stream processing; performance

1 引言

随着移动互联网时代的到来，越来越多的流处理应用不断出现。这些流处理应用广泛的应用于各种生产生活环境，例如，网络监控、通讯数据管理、网站点击数据流监控、传感器网络、移动设备扫描监控等等。数据在这些应用中是以一种持续的数据流的形式存在，而不是存在于传统关系数据库中。

数据流的广泛存在，促使了国内外许许多多的研究团队开发出了各种流处理应用。例如 Storm、

基金项目：国家自然科学基金资助项目（61402464）；国家自然科学基金资助项目（61402474）；国家自然科学基金资助项目（61602467）；

Foundation Items: National Natural Science Foundation of China (61402464); National Natural Science Foundation of China (61402474) ; National Natural Science Foundation of China (61602467)

S4、Spark Streaming 等等。这些流处理应用实现方法和原理虽然不尽相同，但是它们的底层实现或多或少都使用到了流处理算子优化技术。流处理应用由算子和数据流组成，如果把算子表示成有向图中的节点，把数据流表示成有向图中的边，那么它们构成的有向图被称作流图，或者数据流图、流拓扑图。

本文提到流处理算子优化就是针对流图中的算子进行优化重组，尽可能大的提升流处理应用的处理性能。根据不同算子优化技术的特点，本文按照表一中所列的八种算子优化技术分别进行介绍。

表 1　算子优化技术

算子优化技术	流拓扑图	语义	动态性
算子重排	改变	未改变	不定
算子消冗	改变	未改变	不定
算子状态共享	未改变	未改变	静态
算子融合	改变	未改变	不定
算子分裂	改变	不定	不定
算子负载均衡	未改变	未改变	不定
算子替换	未改变	不定	不定
算子负载分流	未改变	改变	动态

其中“语义”表示该优化技术的输入/输出行为，未改变表示采用该算子优化技术后，算子的输入和输出都没有变化；“动态性”表示算子优化技术是在算子编译之前静态发生还是运行过程中发生的，“不定”表示两种情况都有可能，具体看相应的流处理应用的实现。

2　算子重排

算子重排是把选择度低的算子移动到上游算子前面，使流处理应用尽可能早的过滤掉无用的数据，减少下游算子需要处理的数据量，从而提升系统性能。算子重排原理如图 2 所示，将选择度低的算子 B 移动到算子 A 前面。

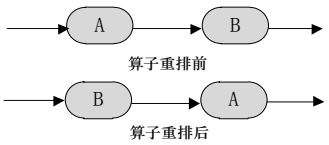

图 1　算子重排示意图

2.1　应用举例

考虑一个视频监控的流处理应用，某城市各个区都布置一定数量的视频监控设备，监控把实时的见拍摄的视频帧数据流回传到服务器。在一次犯罪嫌疑人抓捕过程中，现已经确定该犯罪嫌疑人的身份，警方希望通过该城市的视频监控设备来帮助他们实施抓捕，在视频帧回传服务器端实时监测犯罪嫌疑人，一旦发现可疑人物，立即通知警方实施抓捕。

因为已经确定犯罪嫌疑人的性别为男性，以及他的主要活动区域。视频帧回传服务器端流处理应用其中包含两个算子，算子 A 在视频帧中识别人物的性别，并将包含男性人物的数据帧发送给下游算子 B，算子 B 负责将接收到的数据帧按照该市不同的区进行过滤。

假设视频帧中男女性别分布是 1:1，那么算子 A 将过滤掉一半的数据；算子 B 过滤掉绝大部分无关区域的数据。如果将算子 B 移动到算子 A 前面，那么算子 A 需要处理的数据将会大大减少，系统的性能也会因此而提高。

2.2　收益模型

对于一个算子 β，如果输入的数据项个数是 N，输出的数据项个数是 n，那么我们将 n 与 N 的比值称作算子 β 的选择度。

对于上面的例子中，算子 A 的选择度是 0.5。如果将选择度低的算子移动前面，那么算子重排是有收益的。

假设算子 A 的选择度固定为 0.5，算子 B 的选择度是变化的。下图显示了在算子 B 选择度变化的情况下，算子重排前和算子重排后的系统负载量。

算子重排前，所有到来的数据算子 A 都要处理（数据量按单位 1 计算），接着算子 A 筛选一半的数据给下游算子 B 处理，所以算子重排前算子 A 和算子总共处理的数据恒为 1.5。

算子重排后，所有到来的数据现有算子 B 处理（数据量按单位 1 计算），接着算子 B 按照自己的选择度 β 筛选数据给下游算子 A 处理，所以算子 A 和算子 B 处理的数据量总共为 1 ＋ β 。

从图中可以看出，当横坐标算子 B 选择度 β 低于 0.5 的时候，算子重排前的系统负载量要高于算子重排后的，当算子 B 的选择度 β 大于 0.5 的时候，算子重排后的系统负载量要高于算子重排前的。因此我们得出结论，只要将低选择度的算子移动到上游，我们就可以降低系统负载量，从而提高系统性能。

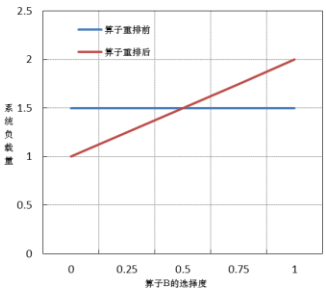

图 2　不同算子选择度下算子重排对系统负载量的影响

2.3　安全条件

并不是所有情况下算子重排都是安全的。当满足以下条件的时候算子重排是安全可行的。

确保属性可用性：由于第二个算子 B 处理的数据是经过算子 A 过滤后的数据项，因此如果要算子重排的话，算子 B 读取的数据属性集必须和算子 A 写入的数据属性集是不相交的。

确保交换性：算子重排后和算子重排前的结果必须是一样的，因为必须保证算子 A 和算子 B 是可以交换的。如果已经确保了属性的可用性，那么交换的充分条件是算子 A 和算子 B 都是无状态的。

2.4　动态性

从前面的介绍中我们知道，算子重排的性能收益是受算子选择度的影响，因此算子重排的性能收益通常依赖于输入的数据的分布。如果输入数据的分布是明确并且不变的，那么算子重排是可以在算子静态编译之前确定的；反之，需要在算子运行的时候动态的改变算子顺序，从而来提高算子性能。

Eddy 算子是通过静态拓扑图转换实现动态算子重排[8]，它假定一个数据项被一个算子丢弃的概率与被另一个算子丢弃的概率是独立的。优点是不必事先知道其选择度，但是这样会导致一些额外的开销来进行数据路由选择。

Babu 等人提出另一种利用近似算法的动态算子重排技术，该方法能够处理数据项在算子之间被丢弃的概率是相互依赖的情况并且保证结果是在最优解的一个小的常数区间之内[11]。

3　算子消冗

算子消冗又叫做子图共享，它是通过共享算子的方式将数据流图中冗余算子消除。

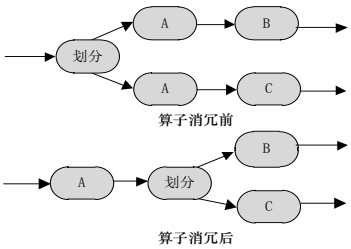

图 3　算子消冗示意图

3.1　应用举例

有两个 web 统计应用，分别接收来自不同网站发来的请求，它们都是以算子 A 开始来解析发送来的 JSON 请求数据，然后将解析后的数据分别交给下游算子 B 和算子 C。由于这两个应用都包含解析 JSON 数据的算子 A，因此可以通过算子消冗技术来消除多余的算子 A，使它们共享一个算子 A，从而节省了计算资源。

3.2　收益模型

如果算子消冗的两个应用是运行在同一台机器上，并且是单核执行的，那么通过算子消冗是有收益的。

算子消冗前，假设系统的负载量是单位 1，算子消冗后，由于流处理应用共享了同一个算子 A，那么系统会少做一次算子 A 的计算，所以系统的负载量会因为消除的算子 A 的负载量而减少。

从下图我们可以看出算子 A 的负载量在整个应用中的负载比值越大，那么算子消冗所带来的性能收益越大。

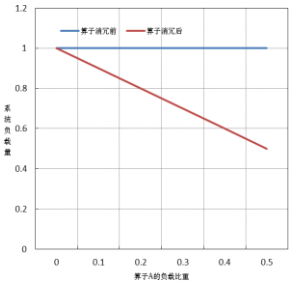

图 4　算子 A 不同负载比重消冗后对系统负载量的影响

3.3　安全条件

确保相同的算法：由于算子消冗会改变流拓扑图，在这种情况下要保证算子优化技术的语义不变，那么要确保消除的冗余算子是等价的。算子等价是经典的不可判定问题，在实际应用过程中可以采用判断算子是否具有相同代码的方法。

确保组合的状态：当消除的冗余算子是无状态

的，那么消除冗余算子是很容易的事情；但是，如果消除的冗余算子是有状态的，那么需要考虑的问题就更多了。

3.4　动态性

基于数据流图的算子静态编译器已经可以检测和消除一个流应用中的冗余算子、无意义算子、幂等算子和死子图等情况。

动态的算子消冗更多的是应用于分布式集群流处理应用的情况，流拓扑图中的算子被分配到集群中的机器上面运行。当提交新的拓扑到集群当中时，系统可以先搜索判断当前是否已经存在公共子图，如果存在的话，就可以通过算子消冗技术来减少系统资源占用。Pietzuch 等人提出比较极端的方法 [36]，把流应用的添加和移除看成一个类似常规数据项添加和移除的优先操作。

4　算子融合

算子融合是把两个连续的算子融合成一个更大的算子。

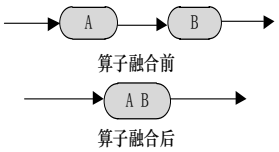

图 5　算子融合示意图

4.1　应用举例

考虑这样一个流量监控应用，算子 A 接收前端回传的流量，并将流量还原成<源 IP，源端口，目的 IP，目的端口，协议类型>形式的五元组信息，然后将五元组发送个后端的算子 B。算子 B 根据五元组信息过滤掉一些常规的流量，对异常的流量进行提取分析。

由于算子B只是根据规则简单的匹配并提取信息，所花费的计算成本非常小，因此可以将算子 A 和算子 B 融合到一个算子里面，这样就可以减少数据传输的成本，同时因为减少了数据处理步骤，也减低了系统出错的几率。在这种情况下，算子融合技术是可以考虑的。

4.2　收益模型

算子融合技术与前面介绍的几种算子优化技术不一样，前面介绍的算子优化技术是以减少整个流处理应用的负载量来提升性能收益的，而算子融合技术并没有减少整个流处理应用的负载量，它只是把位于不同机器上的算子融合到一个算子里面。这样做的好处的是可以减少数据在不同算子之间移动所带来的成本，但是同时它也减少了管道并行。这就需要在管道并行和数据算子间移动的成本进行比较，权衡后选择是否进行算子融合。当算子 B 计算成本很小，同时数据算子间移动的成本很大时，可以考虑进行算子融合。

另一方面，由于算子融合技术减少了数据在算子间移动的步骤，因此也减少了数据传输错误和机器故障的可能性。

4.3　安全条件

目前大多数流处理应用都是基于分布式集群的，流处理任务提交到集群后，会将算子分发到集群中的机器上去执行。因此不同机器上的算子间不会存在本地资源的竞争，例如本地文件，CPU，内存等等。

避免无限递归：如果在流拓扑图中存在闭环，例如一个反馈回路，那么数据可以在回路中无限循环。如果算子是通过函数调用的方式实现融合的，那么将会导致堆栈调用溢出。

4.4　动态性

融合通常是静态完成的，也有一些系统是动态的使用算子融合技术。flextream 系统通过暂停应用，重新编译融合后的代码，然后恢复应用来实现动态融合[27]。这种动态融合技术可以应对流处理应用可用资源的改变，但是由于重新编译和运行也会导致一定的延迟。Tang 和 Gedik 应用算子融合技术，把那些算子共享一个线程的决策留给了运行时，这使得他们能够动态的控制流水线，线程切换和通信之间的成本开销[26]。

5　算子状态共享

算子状态共享是指多个算子共享同一块内存空间，从而减少内存占用，降低对系统的资源需求。

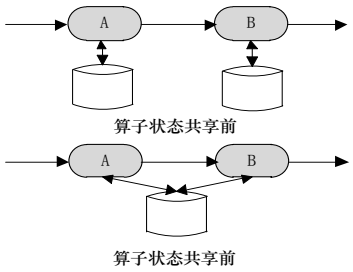

图 6　算子状态共享示意图

5.1　应用举例

假设有个两个算子 A 和 B，分别实时计算某社交网站一分钟内和一小时内最热门的 top N 动态，这样算子 A 和算子 B 需要维护一个很大的内存窗口，来记录当前所有用户发布的动态。如果算子 A 和 B 只是计算的时间粒度不一样，那么可以让算子共用算子 B 的内存空间，这样就可以节省内存空间的使用。

5.2　收益模型

算子状态共享能够带来两方面的收益。首先，由于多个算子共享内存，减少了内存空间的使用，能够减少流处理应用对资源使用的需求。另一个方面，减少内存空间的使用可以提高计算机缓存的命中率，甚至能够减少磁盘 I/O 时间。

5.3　安全条件

确保状态是可访问的：由于多个算子访问同一个内存空间，因此要求该内存空间对多个算子是可访问的。Storm 底层实现了多种并行化技术，Storm 的一个 worker 进程包含有多个 Executor 线程，而每个 Executor 线程又可以启动多个 task，这里的 task 相当于一个算子。Storm 通过这种方式能够确保同一个 Executor 线程不同 task 算子之间状态是可访问的[1]。

避免资源竞争：当多个算子读写同一块内存空间的时候就需要考虑资源竞争的问题。要么确保操作是只读的，要么通过同步访问来控制读写正确性。Brito 等人使用了软件事务内存，该机制允许尝试性地同时对同一状态进行更新，如果需要的话，可以使用回滚技术[15]。

5.4　动态性

状态共享通常是静态的执行的。StreamIt 采用全静态方法，其中一个静态调度规定了什么样的数据在什么时间被什么算子所访问。Brito 等人的工作更多地是动态方式，共享状态的访问通过软件事务内存来协商完成。

6　算子分裂

算子分裂又叫做算子划分和数据并行化，它是把数据按照一定的规则划分成好几份，然后把每份数据路由给下游相同的算子进行处理，是一种典型的数据并行化。

算子分裂又分为两种，一种是基于 key 的算子分裂，另一种是基于批处理的算子分裂。基于 key 的算子分裂是将数据中的某一个或者某一些属性按照属性值的不同，划分到不同机器上的下游算子。最常见的划分算法是利用哈希函数进行划分。基于 key 的算子分裂受限于 key 的个数，并行度有限。基于批处理的算子分裂是把数据按照一定的窗口大小进行划分，最常见的两种形式的窗口是时间滑动窗口和个数滑动窗口。

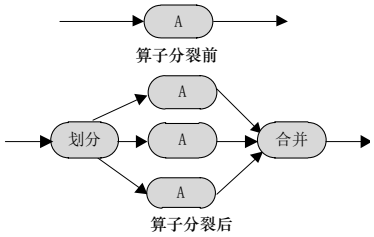

图 7　算子分裂示意图

6.1　应用举例

假设有一个大型的购物网站，时时刻刻都在接收来自全球各地的用户请求，如果只是由单个算子 A 来响应用户请求的话，在请求高峰时期必定会导致请求拥堵等待，影响用户体验。如果将用户发送来的请求按照不同地区，分别划分到下游多个相同的算子 A，那么这样就可以通过算子分裂来满足所有用户的请求。这是一种典型的基于 key 的算子分裂。

6.2　收益模型

算子分裂会引入两个额外的算子，划分算子和合并算子，划分算子将数据划分成多份，然后决定将划分的数据路由到下游哪个算子当中。合并算子负责将所有算子 A 处理后的结果合并，如果合并的数据流有先后顺序，那么合并的开销会很大。

在一般情况下，算子分裂中数据划分和合并的计算成本相对于算子 A 来说是非常小的，因此通过算子分裂来提升系统性能是非常有效果的，并且系统的吞吐量几乎是跟算子 A 的并行度成正比。

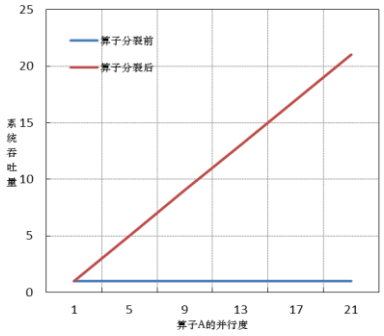

图 8　算子分裂对吞吐量的影响

6.3　安全条件

如果算子是有状态的，那么要保证划分的 key 是不想交的。无状态算子的算子分裂显然是安全的，算子分裂后也能够得到正确的结果；如果算子是有状态的，那么需要保证算子的状态是基于划分后的不同 key，也就是说算子的状态不依赖与整体的数据，而是依赖与划分到当前算子的数据。例如，计算网站每个用户消费的总金额，如果算子分裂是按照不同的用户 ID 来划分的，那么每个算子计算的总金额只与划分到当前算子的数据有关。

如果数据是有顺序的，那么合并算子需要有序合并。对于有些应用，它要求数据在算子分裂后和算子分裂前顺序是一样的。例如，一个流量还原应用，要求处理后的流量和处理前一样是按照一定的顺序排列的。对于这种情况，需要下游合并算子能够使用一定策略保证合并后的数据依然是有序的。目前已经有许多的方法来处理这种情况，CQL（Continue Query Language）使用逻辑时间戳来标记不同的数据[5]；StreamIt 用 round-robin 或者复制的方式[23]，MapReduce 没有按原来的顺序重排，而是使用了一个分布式的“sort”阶段[20]；Storm 为用户提供了 Global Grouping 接口和 Fields Grouping 接口，Global Grouping 接口会把所有数据发送个同一个 BOLT 算子，虽然能够保证整体有序，但是也失去了算子分裂的优势，Fields Grouping 能够保证每个 Fields 内的数据有序。同时 Storm 支持用户通过编程的方式，自己实现合并算子的有序性[1]。

6.4　动态性

在有些情况下，我们一开始并不知道算子分裂的并行度，在流处理应用运行的过程中，可能随着 key 的个数的增加需要动态的复制需要分裂的算子。SEDA 通过线程控制器做到了这一点，它把线程数保持到最大值之下[37]。Storm 允许用户通过手动调整流处理应用的线程数或者进程数，从而动态的调整流处理应用的并行度[1]。

7　算子负载均衡

算子负载均衡是根据算子运行情况，动态的把数据从高负载算子路由给空闲的算子。与前面几种算子优化技术不一样，该算子优化技术不改变数据流图，只是改变数据路由的去向。

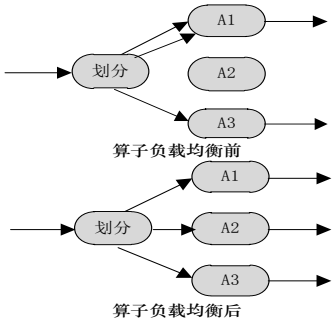

图 9　算子负载均衡示意图

7.1　应用举例

算子负载均衡是在算子分裂基础之上采用的流处理算子优化技术。假设有一个网络流量监控应用，该应用把流量的协议类型作为 key 来进行算子分裂。在运行的过程中，如果某种协议类型的流量突然增加，而其他的协议类型的流量突然减少。这样会导致流量突增的算子过载，而流量减少的算子存在空闲。这种负载倾斜会降低应用整体的吞吐量，甚至过载的算子会由于处理不及时而将数据丢弃。

在流处理应用中，像上面介绍的那样负载倾斜是很常见的，因为大多数数据的统计分布我们并不能事先知道。针对这种情况，如果采用算子负载均衡的话可以很好的解决这种问题。

7.2　收益模型

负载倾斜会导致过载的算子成为整个流处理应用的瓶颈，整个流处理的并行度不再取决于算子分裂复制的并行算子个数，而是由过载算子决定。如果算子负载均衡能够有效的解决数据划分不均匀而引起的负载倾斜问题，那么算子负载均衡是有收益的。

假设过载算子需要处理的数据是所有数据的 50%，尽管算子分裂带来的并行度是 N，那么整个流处理应用的并行度其实是 2。即流处理应用的并行度 $\mathrm{P}=\dfrac{1}{\max\{p_i\}}$，其中 p_i 算子分裂中复制的算子所处理的负载占总的百分比。从公式中我们可以看出流处理应用的整体并行度是取决于过载算子所承担的负载。

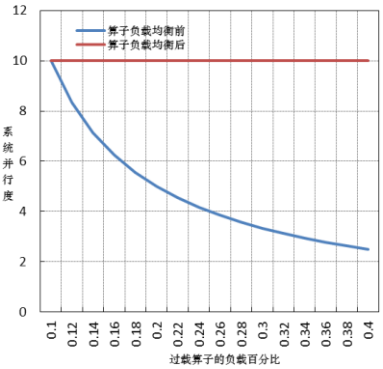

图 10　算子负载均衡对系统并行度的影响

7.3　安全条件

算子负载均衡是基于算子分裂的，因此算子负载均衡首先要满足算子分裂的安全条件。

完全性：确保重新划分的每一个数据都能被下游空闲算子处理。

兼容性：如果算子分裂是基于 key 划分的，那么将多个 key 路由到下游同一个算子的时候，要确保该算子都能够处理。即下游算子能够处理不同 key 类型的数据。

7.4　动态性

根据算子负载均衡的定义，算子负载均衡是算子执行的时候动态进行的。因为负载倾斜是由数据分布不均匀导致，这就需要划分算子在划分数据的时候统计历史数据，及时预测负载倾斜，并将数据路由到空闲算子上。

Vitorovic 等人认为基于 Hash 函数的划分会产生负载倾斜问题，因此提出基于数据流数学统计分布特性来动态扩展相应算子，能够很好的解决负载倾斜问题[2]。

8　算子替换

算子替换是选择具有更高效算法的算子来替换之前的算子。

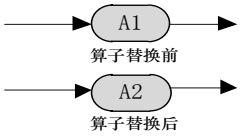

图 11　算子负载均衡示意图

8.1　应用举例

考虑这样一个在数据流上的类 SQL 查询，该查询包含一个连接算子。连接算子有多种实现方法，最常见的三种是：嵌套循环连接、哈希连接和排序连接。通常情况下用嵌套循环方法实现的连接算子磁盘 I/O 延迟很大，因此可以将嵌套循环方法实现的连接算子替换成哈希连接或者是排序连接算子。

8.2　收益模型

采用嵌套循环方法实现的连接算子的复杂度是 $o(n^2)$ 的，哈希连接算子的复杂度是线性的，而排序连接算子的复杂度是介于嵌套循环连接算子和哈希连接算子之间的。如果将一个高成本的算子替换成低成本的算子，那么收益显然是存在的。

下图显示了不同连接算子的输入数据规模 n 和算子执行成本 t 的关系。其中算子执行成本是算子执行时间的近似值。

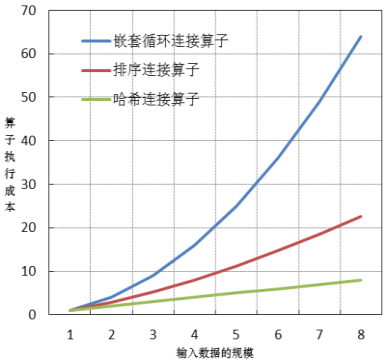

图 12　不同连接算子的执行成本

8.3　安全条件

确保算法等价性：对于要替换的两个算子，要保证它们是等价的，即执行这两个算法得到的结果是一样的。当其中一个算子只能满足特定的情况才能执行的时候，算子替换的时候还要考虑特殊的情况。

8.4　动态性

当算子替换用于对运行时状况做出反馈时，那么它必须是动态的。SEDA 的每个算子可以判定其过载的规则，其中的一个可选项是提供降级服务（即算子替换）；在 Borealis 中，算子能够控制输入，例如：为算子选择一个不同的算法变体[38]。为了实现动态的算子替换，编译器在静态时提供算法的所有变体，运行时动态地选择所需要的其中一个算法。换句话说，这种算子替换的方式和 Eddy 为算子重排所做的方式是一样，即静态地插入一个动态路由组件[8]。

9　算子负载分流

算子负载分流是通过丢弃一部分数据来提高系统吞吐量的算子优化技术。

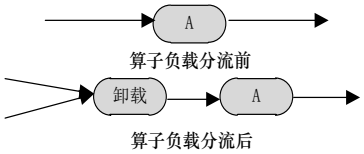

图 13　算子负载分流示意图

9.1　应用举例

负载分流一开始出现在电力供应系统，是指当高峰期电力供应不足的时候，通过灯火管制将部分地区的电力切断，从而使其他大部分地区能够正常电力供应。同样在流处理应用中，算子负载分流是指丢弃部分待处理数据，从而使应用整体能够正常运行。例如一个视频请求应用，当高峰期来临的时候，请求数量急剧增加，系统不能够及时处理的话就会导致请求堆积，影响整个系统的使用，当使用算子负载分流的时候，丢弃一部分用户的请求，以达到系统的正常稳定运行。

9.2　收益模型

根据算子负载分流的定义，算子负载分流会改变算子的语义。但是当算子执行结果不需要很精确或者不需要满足每一个用户请求的话，是可以通过算子负载分流来应对高峰时期的情况。

算子负载分流的精确度是由卸载算子决定的，当卸载算子的选择度越大，算子负载分流的精确度就越大，反之算子负载分流的精确度就越小。下图显示了算子负载分流的精确度和卸载算子的选择度之间的关系。

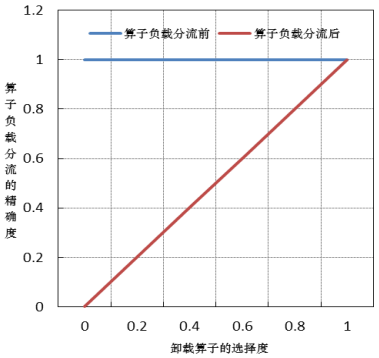

图 14　卸载算子选择度对精确度的影响

9.3　安全条件

从上面的图中我们知道，卸载算子会导致算子负载分流的精确度下降，这也就是说算子负载分流是不安全的。不同于本文描述的其他算子优化技术，算子负载分流技术是在性能和精确度之间进行折中的一种优化技术，尽管这样，但是有的流处理应用是允许这种不精确的近似结果。例如，传感器网络很多都是采集物理世界的模拟信号[56]。大数据时代的到来也使我们从寻求因果关系中解脱出来，进而探索更多的相关关系。

9.4　动态性

根据定义，我们知道算子负载分流是动态进行的，它是在算子运行过程中，需要卸载算子动态监测当前的负载，当负载超过算子的处理能力的时候，就采取一定的策略来卸载部分数据。

10　结论

10.1　算子优化技术总结

本文调研学习了当前比较新的流处理框架，结合实际流处理应用，概括总结了其中最常见的八种流处理技术，并且分析了这八种流处理技术的性能收益模型，从而对算子优化技术的性能收益有一个更加直观的认识。同时为了能够将这些算子优化技术正确地应用于实际，我们还从安全条件和动态性方面介绍了这八种算子优化技术的特点。在实际应用中，这些算子优化技术的情况是复杂的，并且是可以结合起来同时使用的，具体采用哪种算子优化技术要根据具体的条件和优化目标而定。

10.2　未来工作方向

通过这篇综述可以对算子优化技术有一个比较全面的认识，但是细化到具体的某一项算子优化技术，还是有很多需要开拓和改进的地方。例如，在算子替换当中，已经有部分工作在研究如何在流处理应用中使用多路非等值连接(multi-way theta-join)[7、9]来处理更复杂的流处理情况；在算子分裂优化技术中，如何保证在数据划分后合并的整体有序性也是值得研究的问题。除了研究具体算子优化技术的突破和改进，将算子优化技术应用于实际，开发出更高效可用的流处理系统也是未来工作的重要方向。

参考文献：

[1]　Toshniwal A, Taneja S, Shukla A, et al. Storm@ twitter[C]. Proceedings of the 2014 ACM SIGMOD international conference on

Management of data. ACM, 2014: 147-156.

[2] Vitorovic A, El Seidy M, Guliyev K M O, et al. Squall: Scalable Real-time Analytics. Proceedings of the VLDB Endowment,Vol. 9, No. 10.2016.

[3] Gedik B, Schneider S, Hirzel M, et al. Elastic scaling for data stream processing[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(6): 1447-1463.

[4] Lisa Amini, Navendu Jain, Anshul Sehgal, Jeremy Silber, and Olivier Verscheure. Adaptive control of extreme-scale stream processing systems. In International Conference on Distributed Computing Systems (ICDCS).2006.

[5] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. Journal 15, 2 (June 2006):121 – 142.

[6] Kotto-Kombi R, Lumineau N, Lamarre P, et al. Parallel and Distributed Stream Processing: Systems Classification and Specific Issues[J]. 2015.

[7] Okcan A, Riedewald M. Processing theta-joins using MapReduce[C]//Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. ACM, 2011: 949-960.

[8] Ron Avnur and Joseph M. Hellerstein. Eddies: continuously adaptive query processing. In International Conference on Management of Data (SIGMOD). 2000:261 – 272.

[9] Zhang X, Chen L, Wang M. Efficient multi-way theta-join processing using MapReduce[J]. Proceedings of the VLDB Endowment, 2012, 5(11): 1184-1195.

[10] Brian Babcock, Mayur Datar, and Rajeev Motwani. Load Shedding for aggregation queries over data streams. In International Conference on Data Engineering (ICDE). 2004:350 – 361.

[11] Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In International Conference on Management of Data (SIGMOD). 2004:407–418.

[12] Roger S. Barga, Jonathan Goldstein, Mohamed Ali, and Mingsheng Hong. Consistent streaming through time: a vision for event stream processing. In Conference on Innovative Data Systems Research(CIDR). 2007:363 – 373.

[13] Mayer R, Koldehofe B, Rothermel K. Meeting predictable buffer limits in the parallel execution of event processing operators[C]. Big Data (Big Data), 2014 IEEE International Conference on. IEEE, 2014: 402-411.

[14] Irina Botan, Roozbeh Derakhshan, Nihal Dindar, Laura Haas, Ren´ee J. Miller, and Nesime Tatbul. SECRET: A model for analysis of the execution semantics of stream processing systems. In Conference on Very Large Data Bases (VLDB). 2010:232 – 243.

[15] Andrey Brito, Christof Fetzer, Heiko Sturzrehm, and Pascal Felber. Speculative out-of-order event processing with software transaction memory. In Conference on Distributed Event-Based Systems (DEBS). 2008:265 – 275.

[16] Kimberley Burchett, GregoryH. Cooper, and Shriram Krishnamurthi. Lowering: A static optimization technique for transparent functional reactivity. In Partial Evaluation and Semantics-Based Program Manipulation (PEPM). 2007:1–80.

[17] Don Carney, U˘gur Cetintemel, Alex Rasin, Stan Zdonik, Mitch Cherniack, and Mike Stonebraker. Operator scheduling in a data stream manager. In Conference on Very Large Data Bases (VLDB).2003:309 – 320.

[18] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: a scalable continuous query system for internet databases. In International Conference on Management of Data (SIGMOD). 2000:379 – 390.

[19] Corinna Cortes, Kathleen Fisher, Daryl Pregibon, Anne Rogers, and Frederick Smith. Hancock: A language for analyzing transactional data streams. ACM Transactions on Programming Languages and Systems 26, 2 (March 2004), 301–338

[20] Duncan Coutts, Roman Leshchinskiy, and Don Stewart. Stream fusion: from lists to streams to nothing at all. In International Conference on Functional Programming (ICFP). 2007:315 – 326.

[21] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In Operating Systems Design and Implementation (OSDI). 2004:137 – 150.

[22] Bu˘gra Gedik, Kun-Lung Wu, and Philip S. Yu. Efficient construction of compact shedding filters for data stream processing. In International Conference on Data Engineering (ICDE). 2008b.396 – 405.

[23] Michael I. Gordon, William Thies, and Saman Amarasinghe. Exploiting Coarse-Grained task, data, and pipeline parallelism in stream programs. In Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2006:151 – 162.

[24] Michael I. Gordon, William Thies, Michal Karczmarek, Jasper Lin, Ali S. Meli, Andrew A. Lamb, Chris Leger, JeremyWong, Henry Hoffmann, DavidMaze, and Saman Amarasinghe. A stream compiler for communication-exposed architectures. In Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2002:291 – 303.

[25] Goetz Graefe. Encapsulation of parallelism in the volcano query processing system. In International Conference on Management of Data (SIGMOD). 1990:102 – 111.

[26] Martin Hirzel, Henrique Andrade, Bu˘gra Gedik, Gabriela

Jacques-Silva, Rohit Khandekar, Vibhore Kumar, Mark Mendell, Howard Nasgaard, Scott Schneider, Robert Soul´e, and Kun-Lung Wu. IBM streams processing language: analyzing big data in motion. IBM Journal of Research and Development(IBMRD) 57, 3/4 (May/July 2013), 2013:7:1–7:11.

[27] Amir Hormati, Yoonseo Choi, Manjunath Kudlur, Rodric M. Rabbah, Trevor N. Mudge, and Scott A. Mahlke. Flextream: adaptive compilation of streaming applications for heterogeneous architectures. In Parallel Architectures and Compilation Techniques (PACT). 2009:214–223.

[28] Fabian Hueske, Mathias Peters, Matthias J. Sax, Astrid Rheinl ¨ ander, Rico Bergmann, Aljoscha Krettek, and Kostas Tzoumas. Opening the black boxes in data flow optimization. In Conference on Very Large Data Bases (VLDB). 2012:1256 – 1267.

[29] RohitKhandekar, IrstenHildrum, Sujay Parekh, Deepak Rajan, JoelWolf,Kun-LungWu,Henrique Andrade, and Bu˘gra Gedik. COLA: optimizing stream processing applications via graph partitioning. In International Conference on Middleware. 2009:308 – 327.

[30] Monica Lam. Software Pipelining: An Effective Scheduling Technique for VLIW Machines. In programming language design and implementation (PLDI). 1988:318 – 328.

[31] Peng Li, Kunal Agrawal, Jeremy Buhler, Roger D. Chamberlain, and Joseph M. Lancaster. Deadlock-Avoidance for streaming applications with split-join structure: two case studies. In Application specific Systems Architectures and Processors (ASAP). 2010:333 – 336.

[32] Bin Liu, Yali Zhu, and Elke A. Rundensteiner. Run-time operator state spilling for memory intensive long-running queries. In International Conference on Management of Data (SIGMOD). 2006: 347 – 358.

[33] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In International Conference on Management of Data (SIGMOD). 2003:563 – 574.

[34] Guilherme Ottoni, Ram Rangan, Adam Stoler, and David I. August. Automatic thread extraction with decoupled software pipelining. In International Symposium on Microarchitecture (MICRO). 2005:105 – 118.

[35] Hirzel M, Soulé R, Schneider S, et al. A catalog of stream processing optimizations[J]. ACM Computing Surveys (CSUR), 2014, 46(4): 46.

[36] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The PageRank citation panking: bringing order to the web. Stanford Digital Libraries Working Paper (1998).

[37] Peter Pietzuch, Jonathan Ledlie, Jeffrey Schneidman, Mema Roussopoulos, Matt Welsh, and Margo Seltzer. Network-Aware operator placement for stream-processing systems. In International Conference on Data Engineering (ICDE). 2006:49 – 61.

[38] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur C etintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. 2005. The Design of the Borealis Stream Processing Engine. In Conference on Innovative Data Systems Research (CIDR). 277–289..

作者简介：

檀国林，博士研究生，中科院信息工程研究所，北京市海淀区闵庄路甲 89 号，100093，13021200633，tanguolin@iie.ac.cn

张鹏，副研究员，中科院信息工程研究所，北京市海淀区闵庄路甲 89 号，100190，15801683810，pengzhang@iie.ac.cn