# 基于混合处理模型的乱序数据流分布式聚合查询处理技术*

杨　宁[1],许　嘉[1,2**],吕　品[1,2],李陶深[1,2,3]
(1.广西大学计算机与电子信息学院,广西南宁　530004;2.广西高校并行与分布式计算技术重点实验室,广西南宁　530004;3.南宁学院,广西南宁　530200)

**摘要**:为了解决现有的乱序数据流聚合查询处理技术不能在降低查询处理延迟,同时保障聚合查询结果的最终正确性的局限性问题,本研究设计了混合嵌入分布式流处理模块和分布式批处理模块的乱序数据流分布式聚合查询处理技术。该技术一方面基于用户给定的结果质量,限制自适应地优化流处理模块所用的缓冲区大小,从而尽可能降低流处理的查询处理延迟;另一方面基于备份于分布式数据存储系统的历史流数据,并以批处理的方式实现对极其晚到流元组的查询处理,从而保障聚合查询结果的最终正确性。基于真实的乱序数据流数据集对该技术进行测试分析表明:该技术在平均查询处理时延、查询结果精度和系统可扩展性方面,比目前最好的基于缓存的乱序数据流处理技术均具有显著优势。

**关键词**:乱序数据流　混合处理模型　聚合查询　分布式查询处理

**中图分类号**:TP391　　**文献标识码**:A　　**文章编号**:1005-9164(2019)04-0398-07

## 0　引言

随着数据采集技术和网络通信技术的不断发展和成熟应用,许多行业每时每刻都在产生大量流数据,如何对汹涌而至的流数据进行及时查询处理是当下亟待解决的研究问题,受到学术界和工业界的广泛关注[1-2]。数据流上基于滑动窗口的连续聚合查询(后文简称为聚合查询)返回数据流上一定时间窗口内的流元组的聚合统计信息,支持包括事件跟踪[3]、金融分析[4]和网络监控[5]在内的许多重要应用。然而,近年来分布式并行计算的引入在有效提高数据流查询处理效率的同时,增加了查询处理系统的复杂性,导致数据流乱序现象越发突出[6-7],极大影响了数据流聚合查询结果的精度。

数据流乱序问题是数据流查询处理需要解决的首要基础性问题,国内外已有不少研究成果。按处理机理的不同,现有乱序数据流处理技术主要分为基于缓存的处理技术[8-11]、基于标点的处理技术[12]、基于推测执行的处理技术[13]、近似处理技术[14]和混合处理技术[15],并以基于缓存的处理技术为主流应用技术。基于缓存的处理技术通过构建缓冲区等待晚到的流元组,当缓冲区满载后基于流元组的时间戳对缓

*"广西八桂学者"专项经费,广西高等教育本科教学改革工程项目重点项目(2017JGZ10),广西大学科研基金项目(XGZ141182,XGZ150322)和广西研究生教育创新计划项目(YCSW2018036)资助。

【作者简介】
杨　宁(1991—),男,硕士研究生,主要从事大规模流数据查询处理问题的研究。
【**通信作者】
许　嘉(1984—),女,副教授,硕士生导师,主要从事大数据管理等领域的研究,E-mail:xujia@gxu.edu.cn。

【引用本文】
DOI:10.13656/j.cnki.gxkx.20190808.010
杨宁,许嘉,吕品,等.基于混合处理模型的乱序数据流分布式聚合查询处理技术[J].广西科学,2019,26(4):398-404.
YANG N,XU J,LV P,et al.Distributed aggregation query processing technology for out-of-order data streams based on hybrid processing model [J].Guangxi Sciences,2019,26(4):398-404.

冲区内的流元组进行重排序,继而将有序的数据流片段发送至后端查询处理单元完成查询处理。$K$-slack 技术[8]是基于缓存的处理技术的典型代表,其中参数 $K$ 是和缓冲区大小有关的松弛因子。具体而言,$K$-slack 技术维护一个大小为 $K$ 的缓冲区用于缓存已到达的流元组,缓冲区内的流元组在等待至多 $K$ 个时间单位后,会按其时间戳顺序被释放提交给查询处理单元完成查询处理。$K$-slack 技术的设计难点在于如何确定 $K$ 的大小,因为 $K$ 的大小直接决定元组在缓冲区内的等待时间,从而影响对乱序数据流的查询处理效率。具体而言,$K$ 设置得越大,元组在缓冲区内的等待时间就越长,则有可能等到更多的晚到流元组一起进入下阶段的查询处理,从而提升查询结果质量。然而,等待时间的延长同时也会提高查询处理代价、增大查询处理延迟以及降低查询处理吞吐率。因此,不少相关研究工作专门针对 $K$ 值的设定进行研究和优化,以 MP-$K$-slack 技术[9]和 AQ-$K$-slack 技术[10-11]为代表。其中 MP-$K$-slack 技术基于流元组延迟的动态变化来不断调整 $K$ 值,即用不断捕获的数据流上流元组的最大延迟值来更新 $K$。随着系统捕获的流元组的最大延迟值的不断增大,MP-$K$-slack 技术设定的缓冲区也将不断变大,流元组在缓冲区内的等待时间也将不断增长,导致查询处理代价和查询处理延迟的上升,以及查询处理吞吐率的下降。其后提出的 AQ-$K$-slack 技术则分别针对聚合查询和多流连接查询,优化了 $K$ 值的设定策略。特别地,在处理聚合查询时,AQ-$K$-slack 技术以控制乱序数据流上聚合查询的结果精度为目标,实现了在保障一定结果精度的前提下,基于对较近历史流数据的延迟的统计信息动态优化 $K$ 值,尽可能降低缓冲区大小。AQ-$K$-slack 技术能够根据数据流上流元组延迟的动态变化自适应的增减 $K$ 值,从而权衡了聚合查询的结果精度和查询处理延迟这两个重要指标,比 MP-$K$-slack 技术更具优越性。然而,由于内存大小的限制,基于缓存的处理策略只能忽略对延迟较大的流元组的查询处理,因而无论是 MP-$K$-slack 技术还是 AQ-$K$-slack 技术,均不能保障聚合查询结果的最终正确性。

可见,现有乱序数据流的查询处理技术通过牺牲查询处理结果质量换取了查询处理延迟的降低,从而保障对乱序数据流查询处理的及时性。然而,以用户点击数统计[16]为代表的许多数据流查询分析应用,既要求系统能够对快速到达的乱序数据流进行及时的查询处理,又要求系统能够最终提供精确的查询结果,便于精准计费。鉴于现有研究工作不能很好地满足上述类型应用的实际需求,基于当下流行的开源数据流分布式处理平台 Apache Storm[17],本文提出了分布式并行计算环境下基于混合处理模型(Hybrid processing model,HPM),并基于 HPM 提出了乱序数据流连续聚合查询处理技术。混合处理模型(HPM)是分布式流处理模块和分布式批处理模块的混合。其中,分布式流处理模块(后文简称为流处理模块)利用基于缓存的处理思想,通过权衡聚合查询的结果精度和查询处理延迟来保障查询处理的及时性;分布式批处理模块(后文简称为批处理模块)则基于备份至分布式文件系统的历史流数据,处理极其晚到的流元组,从而保障聚合查询结果的最终精准性。本研究工作充满诸多挑战:一方面需要基于数据流乱序状况的统计信息,确定流处理模块中缓冲区大小的优化值(即优化的 $K$ 值),从而保证流处理模块在产生一定精度的聚合查询结果的前提下,尽可能降低查询处理延迟;另一方面需要确定批处理模块处理的数据对象,以及批处理模块执行查询处理的触发条件。

## 1 材料与方法

### 1.1 混合处理模型

本研究基于流行开源的数据流分布式处理平台 Apache Storm,设计与实现了分布式数据流查询处理的混合处理模型(HPM),用于在模型框架层面支持对乱序数据流聚合查询处理,确保其及时性和查询结果最终的精确性。需要说明的是,Storm 提供 Spout 和 Bolt 这两类分布式处理逻辑单元:其中 Spout 代表数据流的源头,负责生产和喷射流数据;Bolt 代表消息处理者,负责处理流数据,既可执行过滤、聚合、查询等数据库操作,又能够通过将多个 Bolt 相连实现对数据流的逐级处理。HPM 在 Storm 下的系统架构包括流处理模块、批处理模块和分布式数据存储模块。

如图 1 所示,HPM 的分布式数据存储模块基于 Apache HBase[18]实现,负责存储查询结果、晚到流元组信息以及历史流数据。流处理模块包含 Kafka Spout 和 SQuery Bolt 这两个分布式处理单元。Kafka Spout 实现了 Kafka 分布式消息队列的功能,能够将原始数据流转换成 Storm 下的流元组形式并喷射给下游分布式处理单元,并将所有流数据备份存储至 Hbase 中。SQuery Bolt 有多个并行执行的

Task,每个 Task 均负责完成 3 项任务:一是查询处理任务,即利用基于缓存的处理思想,对从 Kafka Bolt 获取的乱序流数据执行基于滑动窗口语义的连续聚合查询处理,并将查询结果存储至 Hbase 中;二是晚到流元组信息的登记任务,即将那些无法被流处理模块处理的晚到流元组的时间戳信息写至 Hbase 中;三是缓冲区优化任务,即基于对流元组延迟信息的统计优化调整其缓冲区大小(即 $K$ 值)。批处理模块则包含 BQuery Bolt 和 Hbase Spout 这两个分布式处理单元。具体而言,BQuery Bolt 也拥有多个并行执行的 Task,每个 Task 会基于一定的规则触发对晚到流元组的查询处理,并将基于晚到流元组得到的更精确查询结果写回 Hbase 中。Hbase Spout 则负责向 BQuery Bolt 喷射其执行查询处理所需要的保存在 Hbase 中的晚到流元组信息,以及处理晚到流元组所需的历史流数据信息。实现 HPM 要解决好 3 个关键问题:1)如何在满足用户对流处理模块所提出的查询处理精度的前提下,尽可能缩减 SQuery Bolt 各个 Task 所使用的缓冲区大小(即优化其使用的 $K$ 值),从而尽可能减小流处理模块的查询处理延迟;2)如何确定哪些流元组进入流处理模块执行查询处理,而明确哪些流元组需要在批处理模块进行查询处理;3)需要优化和确定批处理模块执行查询处理的触发规则。

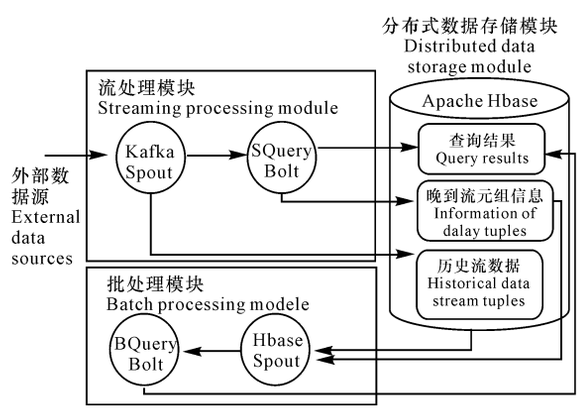


图 1　HPM 的系统架构图

Fig. 1　System architecture diagram of HPM

基于上节所述的混合处理模型(HPM),本文进而实现了针对乱序数据流上聚合查询的分布式查询处理技术。该技术需要解决的关键问题有两个:1)如何为各个 SQuery Bolt Task 设置优化的系统缓冲区大小值,从而满足用户对 HPM 流处理模块提出的结果质量要求;2)如何优化 HPM 中流处理模块和批处理模块间的数据划分策略。下文 1.2 节和 1.3 节分别针对这两个关键问题给出了解决方案。

### 1.2　缓冲区自适应优化方法

用户给定的聚合查询结果质量要求形式为 $(\varepsilon_{thr},\delta)$,其表示因数据流乱序而导致的聚合查询结果误差 $\varepsilon$ 大于等于误差阈值 $\varepsilon_{thr}$ 的概率不大于阈值 $\delta$,即 $prob(\varepsilon \geqslant \varepsilon_{thr}) \leqslant \delta$。考虑到数据流乱序导致聚合查询存在查询结果误差的原因,在于 SQuery Bolt 在执行查询处理时滑动窗口内部分流元组因晚到而缺失,这类似于滑动窗口内的抽样过程,正常到达的流元组相当于被抽样算法抽到的流元组,而晚到流元组相当于没被抽样算法选中的流元组,因而可以利用统计抽样理论确定满足用户指定的结果质量要求 $(\varepsilon_{thr},\delta)$ 时,滑动窗口内需要到达的流元组的比率(即窗口覆盖率阈值 $\lambda_{thr}$,$\lambda_{thr} \in [0,1]$)。而窗口覆盖率阈值 $\lambda_{thr}$ 与所需的缓冲区大小是正相关关系,因而可基于 $\lambda_{thr}$ 值来进一步确定 SQuery Bolt 所需的缓冲区大小。基于统计抽样理论和特定聚合查询的查询语义,可以推导出使聚合查询的结果质量达到用户给定的结果质量要求 $(\varepsilon_{thr},\delta)$ 所需的窗口覆盖率阈值。例如 Ji 等[11]给出了聚合查询 SUM 的窗口覆盖率阈值的推导过程,又如 Law 等[19]给出了聚合查询 AVERAGE、COUNT、MEDIAN 以及 QUANTILE 的窗口覆盖阈值的推导依据。

为了使缓冲区大小的调整过程更具平稳性,这里基于 PD 控制器[20]确定 SQuery Bolt Task 上所用的缓冲区大小值。PD 控制器的输入参数有两类,分别是推导所得的窗口覆盖率阈值 $\lambda_{thr}$,以及查询处理过程中 SQuery Bolt Task 统计得到的每个历史滑动窗口的实际窗口覆盖率值序列,表示为 $\{\cdots,\lambda_i,\cdots\}$。其中,SQuery Bolt Task 基于公式 $n_{rcv}/(n_{rcv}+n_{late})$ 计算滑动窗口的实际窗口覆盖率值,这里 $n_{rcv}$ 表示窗口闭合时 Task 所收到的窗口内流元组的个数,$n_{late}$ 则表示在窗口闭合后一段周期内才到达 Task 的本应落在该窗口内的流元组的个数。此处设缓冲区大小 $K=\alpha k$,其中 $k$ 是缓冲区大小的基础值,等于当前到达系统的所有流元组的延迟值的最大值;$\alpha$ 是调整因子,其值由 PD 控制器计算得到。因此,基于 PD 控制器优化 $\alpha$ 在下一阶段查询处理中的优化取值即可继而确定下一阶段缓冲区的大小的优化值 $K^*$。给定基于用户给出的查询结果质量要求 $(\varepsilon_{thr},\delta)$,推导得到的窗口覆盖阈值 $\lambda_{thr}$ 和 SQuery Bolt Task,计算得到每个历史滑动窗口的实际窗口覆盖率值序列

$\{\cdots,\lambda_i,\cdots\}$,则基于 PD 控制器理论,下一查询处理阶段中调整因子 $\alpha$ 的优化值为 $\alpha^*=\alpha+\Delta\alpha$。其中 $\Delta\alpha$ 的求取过程如公式(1)所示。

$$\Delta\alpha=K_p err(i)+K_d(err(i)-err(i-1)), \quad (1)$$

这里 $err(i)=\lambda_{thr}-\lambda_i$ 表示第 $i$ 个滑动窗口的实际窗口覆盖率相对于预期需达到的窗口覆盖率(即 $\lambda_{thr}$)的误差,简称为第 $i$ 个滑动窗口的窗口覆盖率误差。可见,公式(1)由两部分组成,第一部分 $K_p err(i)$表示最近的窗口覆盖率误差,第二部分 $K_d(err(i)-err(i-1))$ 则表示对未来窗口覆盖率误差的估计。$K_p$ 和 $K_d$ 分别为这两部分的权重系数。

### 1.3 流处理模块和批处理模块间的数据划分策略

如图 2 所示,基于优化的缓冲区大小值 $K^*$ 和滑动窗口的大小 $|W|$,可以将流元组按时间域划分为 3 类。若用符号 $t_{\max}$ 表示到达查询处理系统的流元组的最大时间戳,则第一类流元组的时间戳落在区间 $(t_{\max}-K^*,t_{\max}]$中,第二类流元组的时间戳落于区间$[t_{\max}-K^*-|W|,t_{\max}-K^*]$,第三类流元组的时间戳则小于 $t_{\max}-K^*-|W|$。由于数据流上窗口的闭合条件为窗口的最大时间戳小于等于值 $t_{\max}-K^*$,可见若当前到达的流元组属于第一类,则其所对应的滑动窗口还未闭合,此时应将其发送给流处理模块的 SQuery Bolt,并置于缓冲区内等待后续被查询处理。若当前到达的流元组属于第二类,则意味着 SQuery Bolt 正在对其所属的滑动窗口执行聚合查询处理,故应该将其发送给 SQuery Bolt 执行查询处理。若当前到达的流元组属于第三类,则说明其所对应的滑动窗口已经闭合且过期,即 SQuery Bolt 已删除了该滑动窗口的所有流元组,因而该流元组已无法被流处理模块进行查询处理,而由批处理模块负责后续完成对其的查询处理。

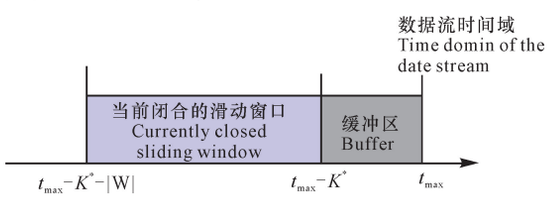


图 2 数据流时间域划分

Fig.2 Time domain division the a data stream

### 1.4 批处理模块的查询触发策略

为了完成对晚到数据流元组的查询处理,保证聚合查询结果的最终正确性,HPM 中的批处理模块需要从 Apache Hbase 中读取晚到流元组对应的滑动窗口内的所有历史流数据。然而,如果每收到一个晚到流元组都执行一次对 Hbase 的访问,会降低批处理模块的查询执行效率,因为网络传输代价和异地磁盘访问的 I/O 代价都会影响查询执行的效率。故可以将一系列到达时间相近的晚到流元组作为一个批次,统一执行一次批查询处理。具体而言,批处理模块中的 Hbase Spout 会监控流处理模块中的 Kafka Spout 存入 Hbase 的晚到流元组信息,并依次计算当前 Hbase 中存储的所有晚到流元组中最大时间戳和最小时间戳的差值,当该差值大于某一特定时间间隔阈值 $\Gamma$ 时,则将这些遍历过的晚到流元组打包为一个批次,并触发批处理模块中的 BQuery Bolt 对该批次的晚到流元组执行查询处理。

### 1.5 评估方法

实验使用的集群由 3 个计算节点构成,每个计算节点的配置是双核 CPU、2 GB 内存,运行 64 位的 Linux (Ubuntu 16.04)操作系统。参数设置方面,参照文献[16]的参数设置方法,将 HPM 涉及的 PD 控制器的输入参数 $K_p$ 和 $K_d$ 的值分别设置为 0.2 和 4;将 HPM 批处理模块触发条件判定时用到的参数 $\Gamma$ 设定为 5 s;将用户给定的查询结果质量要求设置为(0.05,0.05);并将聚合查询的滑动窗口大小和滑动步长分别设置为 0.5 s 和 0.1 s。在查询设置方面,以连续聚合查询 SUM 为测试对象。由于分布式计算环境下单机聚合查询的计算量不是主要代价,因而对聚合查询 SUM 的实验测试结论也同样适用于解释对其他聚合查询(例如 COUNT、MEDIAN、QUANTILE 和 AVERAGE)的处理效果。实验数据方面,使用“德国纽伦堡体育馆足球比赛数据集(RTLS)”[21]中两条真实的乱序数据流 Game 1 和 Game 2 进行,如表 1 所示,与 Game 1 相比,Game 2 拥有更高的晚到流元组个数、流元组最大延迟值、流元组平均延迟值和晚到流元组比率值,因而乱序程度更高。为了便于描述,实验部分将本文所提出的基于 HPM 的乱序数据流分布式聚合查询处理技术简记为△HPM。

表 1 “德国纽伦堡体育馆足球比赛数据集”数据流乱序情况统计

Table 1 Statistics on the out-of-order data stream of the "Nuremberg Stadium Football Match Dataset in Germany"

| 数据流名称 Name of data stream | 流元组个数 Number of tuples | 晚到流元组个数 Number of delay tuples | 流元组最大延迟 Max delay of tuples (s) | 流元组平均延迟 Average delay of tuples (ms) | 晚到流元组比率 Ratio of delay tuples (%) |
|---|---|---|---|---|---|
| Game 1 | 544 223 | 313 405 | 14.2 | 34 | 57.58 |
| Game 2 | 559 211 | 373 664 | 17.1 | 64 | 66.82 |

## 2 结果与分析

图 3 和图 4 分别比较了△HPM 和 MP-$K$-slack 技术在执行乱序数据流聚合查询处理过程中,缓冲区大小变化和平均查询处理时延(即流元组从进入系统到系统最终输出查询结果之间的平均时间间隔)。由于 AQ-$K$-slack 技术和△HPM 一样,也是基于用户给定的结果质量要求来调整缓冲区设置大小,因而此处不针对 AQ-$K$-slack 技术进行横向比较。如图 3 所示,由于 MP-$K$-slack 技术不断用当前得到的流元组的最大延迟值来更新缓冲区大小 $K$,因而其缓冲区大小随时间推移不断增大,前 500 s 的处理过程中其缓冲区大小最后维持在 17 s 左右。而△HPM 由于基于用户对查询质量的要求,在数据流乱序程度不高时动态调减了缓冲区的设置大小,因而其平均缓冲区大小仅为 2.7 s,显著低于 MP-$K$-slack 技术的缓冲区大小。由于缓冲区大小值 $K$ 决定了流元组的排队等待时间,结合图 3 的结论易理解,在图 4 中 MP-$K$-slack 技术的平均查询处理时延显著高于△HPM 的平均查询处理时延。特别的,△HPM 在乱序程度最高的 Game 2 数据流上的平均查询处理时延,仅为 MP-$K$-slack 技术的 20%。

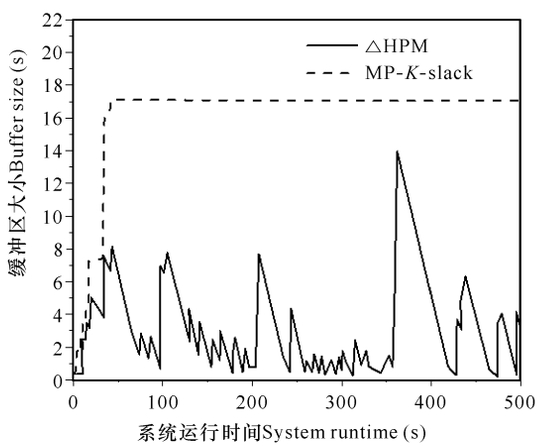


图 3 缓冲区大小对比

Fig. 3 Comparison of buffer size

图 5 展示了△HPM 和 AQ-$K$-slack 技术在处理乱序数据流上连续聚合查询时,累计查询结果精度随系统运行时间的变化情况。△HPM 和 AQ-$K$-slack 技术都可以在流处理时保障用户指定的查询结果质量(即保障查询结果精度为 95%)。由图 5 可见,随着晚到流元组的逐步到达,△HPM 能够基于 Hbase 中备份的历史流数据完成对晚到流元组的查询处理,并提供最终精确的查询结果,因而其累计查询结果精度随时间推移逐渐逼近于 100%。而 AQ-$K$-slack 技术为了保障查询处理的及时性,在满足结果质量要求后会放弃对部分晚到流元组的查询处理,故该技术不能保障聚合查询结果的最终正确性。

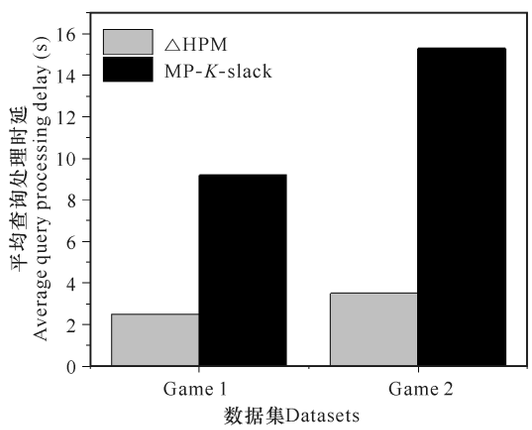


图 4 查询处理时延比较

Fig. 4 Comparison of average query processing delay

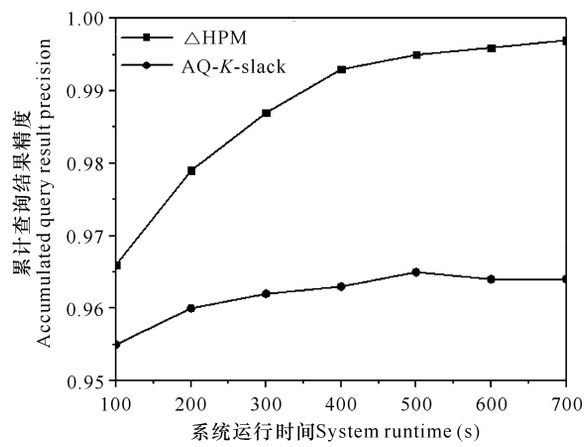


图 5 累计查询结果精度对比

Fig. 5 Accuracy comparison of accumulated query result

由图 6 可见,不论在 Game 1 还是 Game 2 数据集上,随着△HPM 中 SQuery Bolt 的并行执行 Task

数目的增大,系统查询处理的吞吐率均呈线性递增的趋势,表明△HPM 具有良好的系统可扩展性。

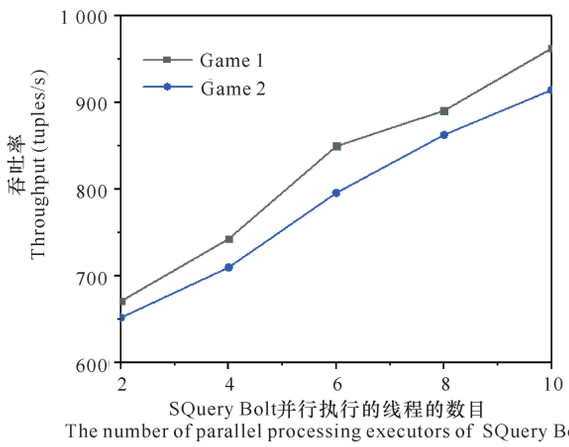


图 6 不同数据集上△HPM 的系统可扩展性测试

Fig. 6 System scalability testing of △HPM in different datasets

## 3 结论

数据流上的连续聚合查询处理是分析和挖掘数据流的重要操作。分布式并行计算是提高数据流查询处理效率的有效手段,但同时带来了突出的数据流乱序问题,导致查询处理的延迟增大、查询结果的质量降低。现有的乱序数据流分布式聚合查询处理技术,不能在降低查询处理延迟的同时,保障聚合查询结果的最终精确性,因此存在局限性。本研究设计了基于混合处理模型(HPM)的乱序数据流分布式聚合查询处理技术,一方面该技术基于用户给定的结果质量要求,自适应地调整缓冲区大小,从而尽可能降低流处理端的查询处理延迟,另一方面该技术利用分布式数据存储系统备份历史流数据,并基于批处理模块实现对极其晚到流元组的查询处理,从而保障了聚合查询结果的最终正确性。基于真实的乱序数据流数据集对本文提出的技术进行测试分析证实:本文提出的技术比目前最好的基于缓存的乱序数据流处理技术,在平均查询处理时延、查询结果精度和技术的系统可扩展性方面均具有显著优势。未来将研究乱序数据流上更多查询操作(例如偏好查询)的高效处理技术。

**参考文献**

[1] ZACHEILAS N,KALOGERAKI V,NIKOLAKOPOULOS Y,et al. Maximizing determinism in stream processing under latency constraints [C]// The 11th ACM International Conference on Distributed and Event-Based Systems. Barcelona,Spain:ACM,2017:112-123.

[2] MENCAGLI G,TORQUATI M,DANELUTTO M,et al. Parallel continuous preference queries over out-of-order and bursty data streams [J]. IEEE Transactions on Parallel and Distributed Systems,2017,28(9):2608-2624.

[3] ZHAO Q,YANG Z,TAO H. Differential earth mover's distance with its applications to visual tracking [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence,2010,32(2):274-287.

[4] GU X,ANGELOV P P,ALI A M,et al. Online evolving fuzzy rule-based prediction model for high frequency trading financial data stream [C]. 2016 IEEE Conference on Evolving and Adaptive Intelligent Systems. Natal,Brazil,2016:169-175.

[5] ACETO G,BOTTA A,PESCAPE A,et al. Efficient storage and processing of high-volume network monitoring data [J]. IEEE Transactions on Network and Service Management,2013,10(2):162-175.

[6] JI Y,SUN J,NICA A,et al. Quality-driven disorder handling for m-way sliding window stream joins [C]// 2016 IEEE 32nd International Conference on Data Engineering. Helsinki,Finland:IEEE,2016:493-504.

[7] 熊安萍,朱恒伟,罗宇豪. Storm 流式计算框架反压机制研究[J]. 计算机工程与应用,2018,54(1):102-106.

[8] BABU S,SRIVASTAVA U,WIDOM J. Exploiting *k*-constraints to reduce memory overhead in continuous queries over data streams [J]. ACM Transactions on Database Systems,2004,29(3):545-580.

[9] MUTSCHLER C,PHILIPPSEN M. Distributed low-latency out-of-order event processing for high data rate sensor streams [C]//2013 IEEE 27th International Symposium on Parallel and Distributed Processing. Boston,MA,USA:IEEE,2013:144.

[10] JI Y,ZHOU H,JERZAK Z,et al. Quality-driven continuous query execution over out-of-order data streams [C]// Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Melbourne,Victoria,Australia:ACM,2015:889-894.

[11] JI Y,ZHOU H,JERZAK Z,et al. Quality-driven processing of sliding window aggregates over out-of-order data streams [C]// DEBS'15 Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems. Oslo,Norway:ACM,2015:68-79.

[12] LIN Q,OOI B C,WANG Z,et al. Scalable distributed stream join processing [C]// SIGMOD'15 Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. Melbourne,Victoria,Australia:ACM,2015:811-825.

[13] LIU M,LI M,GOLOVNYA D,et al. Sequence pattern query processing over out-of-order event streams [C]// ICDE'09 Proceedings of the 2009 IEEE International Conference on Data Engineering. Shanghai,Chi-

na:IEEE,2009:784-795.
[14] TIRTHAPURA S,WOODRUFF D P. A general method for estimating correlated aggregates over a data stream [C]//2012 IEEE 28th International Conference on Data Engineering. Washington, DC, USA: IEEE, 2012:162-173.
[15] KRISHNAMURTHY S,FRANKLIN M J,DAVIS J, et al. Continuous analytics over discontinuous streams [C]//SIGMOD10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. Indianapolis,Indiana,USA:ACM,2010:1081-1092.
[16] ANANTHANARAYANAN R,BASKER V,DAS S,et al. Photon:Fault-tolerant and scalable joining of continuous data streams [C]// SIGMOD '13 Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York, USA: ACM, 2013: 577-588.
[17] Apache Storm Project [EB/OL]. [2019-03-12]. http://storm.apache.org.
[18] Apache Hbase Project [EB/OL]. [2019-03-12]. http://hbase.apache.org.
[19] LAW Y N,ZANIOLO C. Improving the accuracy of continuous aggregates and mining queries on data streams under load shedding [J]. International Journal of Business Intelligence and Data Mining,2008,3(1): 99-117.
[20] LEVINE W S. Control System Applications [M]// The control handbook: Second Edition. Boca Raton, USA:CRC Press,2011.
[21] VON DER GRÜN T,FRANKE N,WOLF D,et al. A real-time tracking system for football match and training analysis [M]// HEUBERGER A,ELST G,HANKE R (eds). Microelectronic Systems. Berlin Heidelberg:Springer,2011:199-212.

# Distributed Aggregation Query Processing Technology for Out-of-order Data Streams Based on Hybrid Processing Model

YANG Ning[1],XU Jia[1,2],LV Pin[1,2],LI Taoshen[1,2,3]
(1. School of Computer, Electronics and Information, Guangxi University,Nanning,Guangxi,530004,China; 2. Guangxi Colleges and Universities Key Laboratory of Parallel and Distributed Computing Technology,Nanning,Guangxi,530004,China; 3. Nanning University,Nanning,Guangxi,530200,China)

**Abstract**: The existing out-of-order data stream aggregation query processing techniques cannot guarantee the final correctness of the aggregated query result while reducing the query processing delay. In order to solve this limitation, this paper designs a distributed aggregation query processing technique for out-of-order data streams based on both of the distributed streaming processing model and the distributed batch processing model. The proposed technique on one hand optimizes the buffer sizes used by the distributed streaming processing model based on a user-given constraint on query result quality, thereby minimizing the query processing delay of the stream processing as much as possible. And on the other hand, based on the historical stream data backed up in the distributed data storage system and in batch processing mode, the query processing of the extremely late tuples is realized, so as to ensure the final precision of the aggregated query results. The test analysis based on the real out-of-order data stream dataset shows that compared with the current best cache-based out-of-order data stream processing technique, the proposed technique has significant advantages in average query processing delay, query result precision and system scalability.
**Key words**:out-of-order data streams,hybrid processing model,aggregated queries,distributed query processing

责任编辑:陆　雁