DOI: 10.11992/tis.201908011
网络出版地址: http://kns.cnki.net/kcms/detail/23.1538.tp.20191126.1539.002.html

易变数据流的系统资源配置方法

王春凯[1,2], 庄福振[2], 史忠植[2]
（1. 中国再保险 (集团) 股份有限公司 博士后科研工作站, 北京 100033; 2. 中国科学院 计算技术研究所, 北京 100190）

摘　要: 大规模数据流管理系统往往由上层的关系查询系统和下层的流处理系统组成。当用户提交查询请求时, 往往需要根据数据流的流速和分布情况动态配置系统参数。然而, 由于数据流的易变性, 频繁改变参数配置会降低系统性能。针对该问题, 提出了 OrientStream+框架。设定以用户自定义查询延迟阈值为间隔片段的微批量数据流传输机制; 并利用多级别管道缓存, 对相同配置的数据流进行批量处理; 然后按照数据流的时间戳计算出精准查询结果; 引入基于异常检测的增量学习模型, 用于提高 OrientStream+的预测精度。最后, 在 Storm 上实现了该资源配置框架, 并进行了大量的实验。实验结果表明, OrientStream+框架可进一步降低系统的处理延迟并提高系统的吞吐率。

关键词: 大规模数据流管理系统; 易变数据流; 增量学习; 模型预测; 参数配置; 微批处理; 系统性能; 异常检测
中图分类号: TP311　　**文献标志码**: A　　**文章编号**: 1673−4785(2019)06−1278−08

中文引用格式: 王春凯, 庄福振, 史忠植. 易变数据流的系统资源配置方法 [J]. 智能系统学报, 2019, 14(6): 1278–1285.
英文引用格式: WANG Chunkai, ZHUANG Fuzhen, SHI Zhongzhi. System resource allocation for variable data streams[J]. CAAI transactions on intelligent systems, 2019, 14(6): 1278–1285.

System resource allocation for variable data streams

WANG Chunkai[1,2], ZHUANG Fuzhen[2], SHI Zhongzhi[2]
(1. Post-doctoral Research Center, China Reinsurance (Group) Corporation, Beijing 100033, China; 2. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: A large-scale data stream management system (LSDSMS) usually contains a relational query system (RQS) and a stream processing system (SPS). When users submit queries to the RQS, it is often necessary to set system parameters according to the rate and distribution of the data streams. However, because of the variability of data streams, changing the resource allocation often reduces the performance of the LSDSMS. In view this problem, we propose a framework for automating the characterization deployment in the LSDSMS OrientStream+. First, based on a user-defined query latency threshold, we designed a data stream transmission mechanism for a mini-batch scheme. Then, we introduced a multi-level pipeline cache for processing batch data streams in the same configuration and obtained accurate query results using the timestamp of the data streams. We also propose an incremental leaning technique with outlier detection to improve the prediction accuracy of OrientStream+. Finally, we validated the proposed approach on the open-source SPS–Storm. Our experimental results show that OrientStream+ can reduce processing latency and improve the LSDSMS throughput.

Keywords: large-scale data stream management system; variable data stream; incremental learning; model prediction; parameter configuration; mini-batch processing; system performance; outlier detection

日前, 许多应用需要大规模的连续查询和分析, 如: 社会网络中的微博分析、金融领域中的高频交易监控, 以及电子商务中的实时推荐等[1-3]。这些应用往往需要快速响应用户提交的查询请求, 要求大规模数据流管理系统对数据流的查询

收稿日期: 2019−08−15.　**网络出版日期**: 2019−11−27.
基金项目: 国家自然科学基金项目 (U1836206, 61773361); 中国博士后科学基金项目 (2019M650044).
通信作者: 王春凯. E-mail: chunkai_wang@163.com.

处理具有较高的吞吐率和较低的处理延迟。这往往需要用户预先设置相关的系统参数, 如查询算子的并行度、查询进程的内存使用率等。然而, 由于数据流的易变性和查询任务的不同, 为确保实时处理查询请求的同时尽量减少资源使用情况是一个非常有挑战性的问题。接下来举例说明该问题的普遍性。

我们以交通监控系统实时分析路况为例, 使用流处理系统 Storm[4] 和轨迹数据集 GeoLife[5] 实现如下查询任务。查询包含一个映射处理逻辑, 用于接收由 GPS 设备采集的轨迹数据, 并通过函数映射找到使用该 GPS 设备的对象所在的道路信息。此外, 包括一个测速处理逻辑接收来自映射处理逻辑发送的数据, 并实时计算出不同道路上的各 GPS 设备对象的平均行驶速度。

然而, 配置查询任务的参数不能动态感知数据流的变化, 导致了查询延迟的增加和系统资源的浪费。为应对此问题, 文献 [6-7] 已进行了相关研究。但是, 文献 [6] 需要重启查询任务, 数据阻塞和查询延迟的问题较为突出; 文献 [7] 通过保存状态信息避免了查询任务的重启操作。然而, 针对流速频繁改变的易变数据流, 文献 [6-7] 均会导致系统延迟的缓慢增加, 以至于超过用户自定义的查询延迟阈值。为此, 本文提出了应对易变数据流的系统资源动态配置方法 OrientStream+。与文献 [6-7] 提出的 OrientStream 相比, OrientStream+可较好解决易变数据流的资源配置问题, 进一步降低流处理系统的查询延迟并提高系统的吞吐率。

针对系统资源动态配置的相关工作可总结为如下 3 个方面:

1) 动态加载调度策略。Aeolus[8] 是柏林洪堡大学和惠普实验室联合研发的 Storm 优化器, 用于动态设置算子的并行度和节点内部数据的批量大小。Aeolus 定义了处理单条元组所需时间的代价模型, 其中包括元组的传输时间、等待时间、计划处理时间和实际处理时间。依据该模型, 针对不同的查询请求和数据流特征 (如数据流速、数据分布情况等), Aeolus 可计算出算子并行度和数据批量传输大小的最佳配置样式。为避免资源浪费或无法实时获取正确的查询结果, FU 等[9] 设计了基于云环境的大规模数据流管理系统的动态资源调度器。该调度器借助开放排队网络[10] 理论来度量已使用资源和查询响应时间的关系、制定最佳资源配置方案以及使用最小开销测量系统的负载等。Aniello 等[11] 针对 Storm 平台, 利用基于拓扑的策略和基于流量的动态调度策略设计了两个调度算法, 以降低元组处理的延迟时间和减少多个拓扑节点间的传输流量。然而, Aeolus 和 DRS 需要明确每个算子的具体处理时间, 并且仅用于固定的查询应用场景。文献 [11] 仅考虑传输延迟, 而未关注资源使用的情况, 并且不能对算子的并行度做动态调整。

2) 机器学习技术。文献 [12] 提出了一种基于混合密度网络[13] 的模型来评估数据流处理任务的资源使用情况。该模型可帮助用户判断是否向流处理系统提交新的查询任务。ALOJA 项目[14] 针对 Hadoop[15] 的执行情况开发了开源平台用于预测查询任务的执行时间和异常监控。ALOJA 是基于 ALOJA-ML[16] 设计的框架, ALOJA-ML 利用机器学习技术分析了运行在 Hadoop 上的不同查询任务的基准性能数据, 并以此支持查询任务的性能调优。Jamshidi 等[17] 设计了一种自动优化流处理系统参数配置的贝叶斯优化算法 BO4CO。以 MySQL 和 Postgres 为实验平台, OtterTune[18] 利用经验数据的监督学习方法和新搜集信息的非监督学习方法, 针对不同查询请求选择出对系统性能影响最大的参数, 并通过历史查询任务对新的查询任务进行预测, 利用深度学习框架 TensorFlow[19] 向用户推荐最佳参数配置。然而, 文献 [12] 不能动态改变流处理系统的调度策略和各个算子的并行度, 且不可以预测系统资源的使用情况。ALOJA-ML 框架仅可预测 Hadoop 的处理平台, OtterTune 系统仅可预测数据库管理系统, 均不能用于数据流的查询场景。BO4CO 只能以流处理系统的历史数据作为训练集, 不能对新收集的性能数据作增量分析。

3) 针对关系查询系统的资源预测。正如我们所知, 关系查询系统往往具有类 SQL 的查询接口。因此, 有些研究也致力于检测 SQL 查询的资源消耗。针对微软的 SQL Server 数据库的不同查询请求, Li 等[20] 设计了两种特征抽取的机制用于预测 SQL 查询的资源消耗情况。两种特征包括粗粒度的全局特征和细粒度的算子特征。Akdere 等[21] 为预测不同查询计划的查询性能, 构建了 3 种层次模型: 查询计划层模型、算子层模型和针对嵌套查询的混合模型。然而, 模型[20-21] 仅考虑了静态特征的选择过程, 不能对系统进行动态监控, 并且没有考虑位于关系查询系统下面的数据处理系统的有关特征。

本文提出的 OrientStream+框架不同于以上工作。OrientStream+构建了以延迟阈值为间隔片段

的微批量样式 (mini-batch scheme) 的数据流传输机制，利用多级别管道缓存计算出精准查询结果，并提出异常检测的增量学习模型 ODRegression (outerlier detection regression)，可较好解决易变数据流的资源配置问题。

1　基本概念与问题描述

1.1　概念描述

OrientStream+需要实时监控大规模数据流管理系统的查询执行情况，并基于不同的数据流速和参数配置搜集训练数据集。接下来，我们给出形式化的定义。

由于数据流无限的特性，本文采集训练数据集的过程使用窗口模型 (见定义 1)。

定义 1　窗口模型。将无限的数据流切分成若干有限子数据流，每次的查询处理仅针对当前窗口内的子数据流。一般可根据用户设定的时间间隔或窗口内元组数量设置窗口大小，并在多查询场景下使用翻转窗口或滑动窗口的语义信息。

在 OrientStream+框架中，我们使用基于时间间隔的窗口模型获取训练集。每个具有不同时间间隔的窗口作为一条训练数据。训练样本搜集过程中，时间窗口的下限是 30 s，上限是 120 s。首先，以初始的参数配置启动查询请求，当窗口大小达到时间间隔约束时，我们采集一条训练数据。接下来，以新的窗口大小和新的参数配置重启查询请求，用于获取下一条训练数据。最后，通过迭代操作，我们可采集到不同窗口大小和配置信息的训练数据集。

定义 2　算子并行度。构建在分布式集群上的流处理系统往往可同时执行由不同类型算子构成的若干拓扑任务。每个算子可根据不用的查询请求设置不同的并行度，一般以多线程的形式实现。以本文使用的 Storm 系统为例，可动态设置数据源部件 spout 和查询处理部件 bolt 的单元实例 task 的并行度。

定义 3　系统处理延迟。每个数据流元组被各个查询算子处理延迟的总和。数据源 i 的处理延迟表示为每个单元实例处理延迟的平均值，形式化定义为

$$\text{Latency}(\text{source}_i)=\frac{\sum_{j=1}^{m}(\text{Latency}(\text{source}_i))_j}{m} \tag{1}$$

式中：m 是数据源 (source) 节点的并行度。然而，由于查询请求往往涉及到多个数据流，因此，大规模数据流管理系统的处理延迟需要按照每个数据源的最大处理延迟来定义，形式化定义为

$$\text{Latency}=\max_{i=1}^{n}\text{Latency}(\text{source}_i) \tag{2}$$

式中 n 是查询请求中涉及到的数据源个数。

定义 4　参数配置。向流处理系统提交查询任务时，需提前定义进程的内存大小、不同算子的并行度等参数信息。该过程称为系统资源的参数配置。

1.2　问题定义

向大规模数据流管理系统提交查询请求时，往往需要凭借用户的经验和平台的硬件情况动态配置不同的参数。参数配置是否合理将直接影响系统的吞吐率和处理延迟，以及平台的资源使用情况。随着数据流的变化情况，我们需要动态调整参数配置以满足用户对处理延迟和系统吞吐率阈值的要求。但是，流处理系统往往不允许任意调整配置参数。比如，Storm 的“re-balance”机制，仅可降低处理单元的并行度，不能超越用户设定的最大处理单元实例值。

我们需要对任意的参数配置预测资源使用情况、处理延迟和系统吞吐率。根据预测结果，从中选取最优配置。即，在保证处理延迟和系统吞吐率满足用户设定阈值的前提下，尽量减少 CPU 和内存的资源使用率。该问题可形式化表示为资源使用最优化的问题，定义如下。

令 $N=(n_1,n_2,\cdots)$ 为集群的各个节点集合，每个节点 n_i 关于 CPU 使用情况 U_{cpu} 和内存使用情况 U_{memory} 可用式 (3) 表示。

$$U(n_i)=\alpha\times U_{\text{cpu}}(n_i)+\beta\times U_{\text{memory}}(n_i)\ (\alpha+\beta=1) \tag{3}$$

式中：α 和 β 分别是 CPU 和内存使用率的权重。本文中，α 和 β 均设置为 50%。

接下来，令 $C=(c_1,c_2,\cdots)$ 为用户提供的候选配置集合。对整个集群来讲，我们需要预测出最佳的配置 c_{opt} 以实现式 (4) 的优化目标。

$$\begin{aligned}&\text{Minimize}\sum_{n_i\in N}U(n_i)\\&\text{s.t. }R(\text{latency})<T(\text{latency})\\&\text{and }R(\text{throughput})>T(\text{throughput})\end{aligned} \tag{4}$$

式中：R(latency) 和 R(throughput) 是查询请求的处理延迟和吞吐率；T(latency) 和 T(throughput) 是用户设置的对应阈值。

2　系统设计

2.1　OrientStream+概述

如图 1 所示，给出了 OrientStream+系统的架构图。该系统主要分为 3 个部分：左部是层次性的特征抽取机制，从下向上主要分为 3 个部分即

硬件集群层特征集、流处理系统算子层特征集, 以及流查询系统的查询计划层特征集; 中部是对 n 个数据流以微批量的处理方式, 通过模型预测获取 m 个参数配置, 并将相同参数配置的子数据流存放至同一个 Kafka[22] 消息队列中。右部是查询监视器 (query monitor), 主要负责采集特征数据并通过增量学习模型预测系统资源的使用情况, 可从候选配置项集中预测出最佳配置和异常警告。

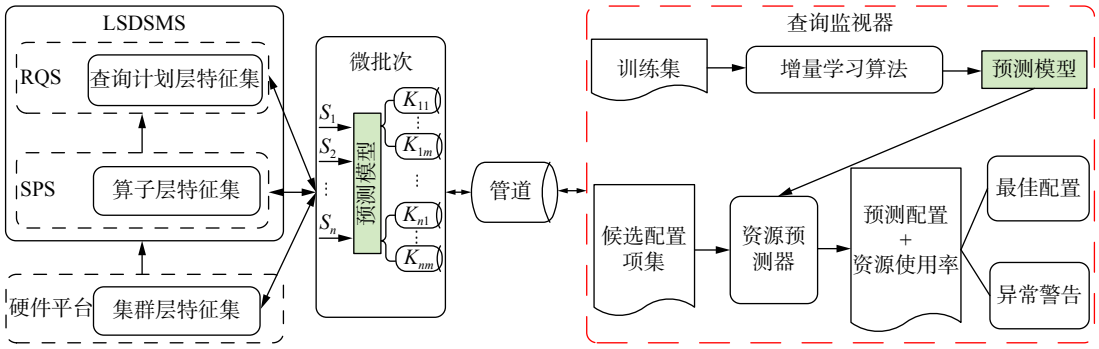

图 1　OrientStream+系统架构图

Fig. 1　OrientStream+ architecture

2.2　微批量数据样式传输

由于频繁设置系统的参数配置会导致处理延迟的不断增加, 所以引入 Sax 等在文献 [23] 中提出的批量层次策略, 以用户设置的查询延迟阈值为滑动窗口大小, 在 Storm 系统上使用该策略实现窗格内数据的微批量处理。

2.3　资源配置策略

2.3.1　多管道数据缓存

根据微批量数据传输模式, 我们以用户定义的延迟阈值为微批量传输的窗口大小。如图 1 中间部分所示, 窗口内的微批量数据首先通过增量学习的模型进行参数配置的预测, 依次记录需要调整配置的次数 c_1、c_2、…、c_n。针对不同类型的查询请求, 现场调整机制下,以文献 [7] 的实验结果所示, 拓扑任务的调整延迟在 100~300 ms 之间。本文中, 我们以上限 300 ms 当作单次调整的延迟, 设计了多管道数据缓存算法 MPDC(multiple pipeline data cache), 如算法 1 所示。

算法 1　MPDC 算法

输入　用户自定义阈值 $T_{\text{threshold}}$; 单次调整延迟阈值 $L_{\text{threshold}}$;

输出　多个子管道 m。

1) $n_{\text{prediction}}$ = 模型预测需要调整配置的次数;

2) $n_{\max} = T_{\text{threshold}} / L_{\text{threshold}}$;

3) if($n_{\text{prediction}} > n_{\max}$)

4) n_{diff} = 统计不同的参数配置个数;

5) if($n_{\text{diff}} < n_{\max}$)

6) $m = n_{\text{diff}}$;

7) else

8) $m = n_{\max}$;

9) 选取并行度最高的 $n_{\max}$ 个配置;

10) 随机向 m 个子管道分发并行度低的 n_{diff}–$n_{\max}$ 个子数据流;

11) end if

12) end if

算法 1 首先以用户定义的延迟阈值作为微批量传输的窗口大小, 在窗口内使用增量学习模型预测出需要调整参数配置的次数 $n_{\text{prediction}}$(行 1), 根据用户定义的延迟阈值和单次调整拓扑结构的最大阈值, 我们可计算出数据传输子管道的最大值 $n_{\max}$(行 2)。在调整次数 $n_{\text{prediction}}$ 大于子管道最大值 $n_{\max}$ 的情况下, 需要统计出 $n_{\text{prediction}}$ 个调整次数中不同的参数配置个数 n_{diff}(行 4)。如果参数配置个数 n_{diff} 小于子管道最大值 $n_{\max}$, 则根据不同的配置参数, 将数据流划分至 n_{diff} 个子数据流进行处理(行 5~6)。如果参数配置个数 n_{diff} 大于子管道最大值 $n_{\max}$, 则首选选取并行度最高的 $n_{\max}$ 个配置参数, 并将余下的 n_{diff}–$n_{\max}$ 个子数据流随机向 $n_{\max}$ 个子管道中发送并进行处理 (行 7~10)。此时, 按照并行度高的参数配置策略, 在消耗部分过多系统资源的情况下, 可满足用户定义的延迟阈值。

2.3.2　精准查询处理

由于我们使用子管道的数据处理方式, 数据流通过 MPDC 算法后, 各个子管道内的数据流并非按照时间顺序排序。因此, 我们需要完成原始数据流的精准查询处理。如图 2 所示, 我们在流处理系统之上构筑基于元组时间戳的映射函数, 将不同子管道数据流的处理过程通过哈希映射后, 确保输出精准的查询结果。

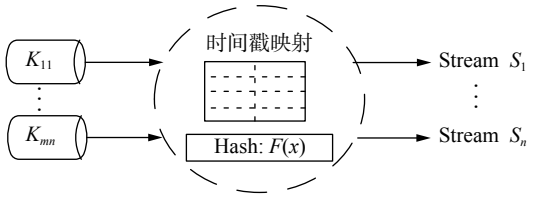

图 2　映射过程

Fig. 2　Mapping process

2.3.3　增量学习模型

利用预测精度最高的 4 个模型 (贝叶斯模型[24]、Hoeffding 树模型[25]、在线装袋模型[26] 和最近邻模型[27]), 文献 [7] 给出了集成学习方法 EDKRegression。但是, 在增量学习过程中, 由于训练数据的动态变化和分布的不均衡性, 导致个别模型的预测精度和实际值偏差较大。为此, 本文在 EDKRegression 算法的基础上, 提出了异常检测回归模型 ODRegression (如算法 2 所示)。

算法 2　ODRegression 算法

输入　4 个学习模型对样本 n 的预测值 P_1、P_2、P_3、P_4;

输出　样本 n 的预测值

1) E = 模型预测值的均值;

2) δ = 模型预测值的方差;

3) for (i=1; i=<N; i++) do

4) if($|P_i-E|>\delta$)

5) 移除第 i 个预测模型;

6) end if

7) end for

8) 调用 EDKRegression[5] 算法计算预测值;

首先, 根据 4 个预测模型对样本 n 的预测值 P_1、P_2、P_3、P_4, 算法计算出预测值的均值 E 和方差 δ(行 1~2)。然后, 如果模型预测值 P_i 与均值 E 相差的绝对值大于方差 δ 时, 利用行 4 的公式移除偏移较大的预测模型。最后, 针对过滤后的预测模型, 调用集成回归模型 EDKRegression 算法, 计算出样本 n 的最终回归预测值。通过回归模型的异常检测, 可进一步提高集成学习模型的预测精度。

3　实验与结果分析

3.1　实验准备

1) 实验环境。本文实验平台用 1 GB 网络连通 14 个物理节点, 其中 5 个是使用 Kafka 的数据发送节点, 1 个是 Storm 的 nimbus 节点, 其余 8 个是 Storm 的 supervisor 节点。数据发送与 nimbus 各节点配置如下: CPU 为 Intel E5-2620 2.00 GHz, Memory 为 4 GB。supervisor 各节点配置如下: CPU 为两颗 Intel E5-2620 2.00 GHz, Memory 为 64 GB; 操作系统为 Ubuntu-14.04.3; Storm 版本 0.9.5。

2) 查询任务。我们依据不同的查询特征, 分别选取了 3 个查询任务。

① 交通监控 (traffic monitoring, TM)。此查询任务的细节请参见第 1 节相关内容。

② 单词计数 (word count, WC)。统计不同语句中各个单词的出现频率。该查询任务包含一个将句子切分成单词的处理逻辑, 和一个使用哈希映射来统计单词出现频率的处理逻辑。我们使用 HiBench[28] 提供的单词计数数据集, 共涉及 300 万个句子和超过 3 000 万个单词。

③ TPC-H(Q3)。TPC-H[29] 是一个决策支持基准, 其包含的查询和数据具有广泛的行业相关性。为验证多个数据流的查询处理过程, 选择 Q3 作为第 3 个查询任务。Q3 共包括 3 个过滤数据源的处理逻辑, 两个做等值连接的处理逻辑, 一个对连接结果做分组的处理逻辑, 以及一个对分组结果进行排序的处理逻辑。在查询任务的执行过程中, 对每个数据源各取 1 500 万个元组。

3) 数据规模。为保证模型预测的准确性, 针对每个训练样本, 计算窗口时间内 CPU 使用率、内存使用率、处理延迟和吞吐率的平均值。对于每个查询任务, 通过随机设置数据速率和 30~120 s 内的动态窗口大小, 分别采集 3 000 个训练样本。

3.2　延迟与吞吐率

通过利用微批次的处理方式, OrientStream+ 应对易变数据流的效果显著, 处理数据的延迟和吞吐率情况均优于 Storm 和 OrientStream。

这里使用 3 个不同类型的查询任务, 对比了 OrientStream+和 OrientStream 的延迟与吞吐率。如图 3 所示, 随着数据流速的频繁变化, 由于频繁调整系统的参数配置, OrientStream 的查询延迟不断增加, 超过了用户自定义阈值。OrientStream+ 利用多管道数据缓存的策略确保了查询任务的延迟低于用户自定义阈值。同时, 如图 4 所示, OrientStream+的系统吞吐率在满足用户定义阈值的前提下, 均高于 OrientStream 的系统吞吐率。

3.3　在线资源预测

关于资源使用的回归模型预测, 我们使用 EDKRegression[7] 和 ODRegression 两个模型。针对不同的查询任务, 表 1 和表 2 分别给出了使用不同模型的测试结果, 包括平均绝对误差值 (mean absolute error, MAE) 和相对绝对误差值 (relative absolute error, RAE)。

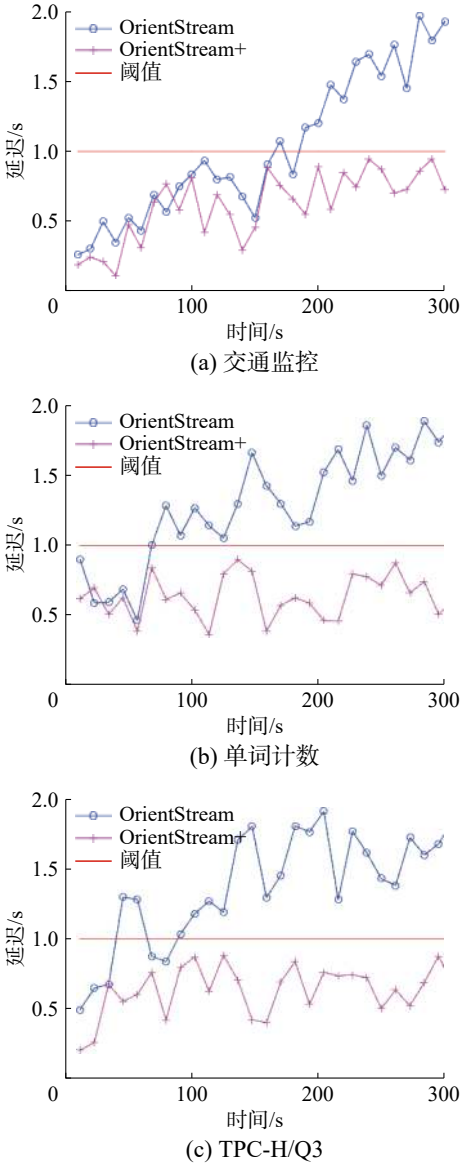

图 3　不同查询任务的延迟

Fig. 3　The latency of different workloads

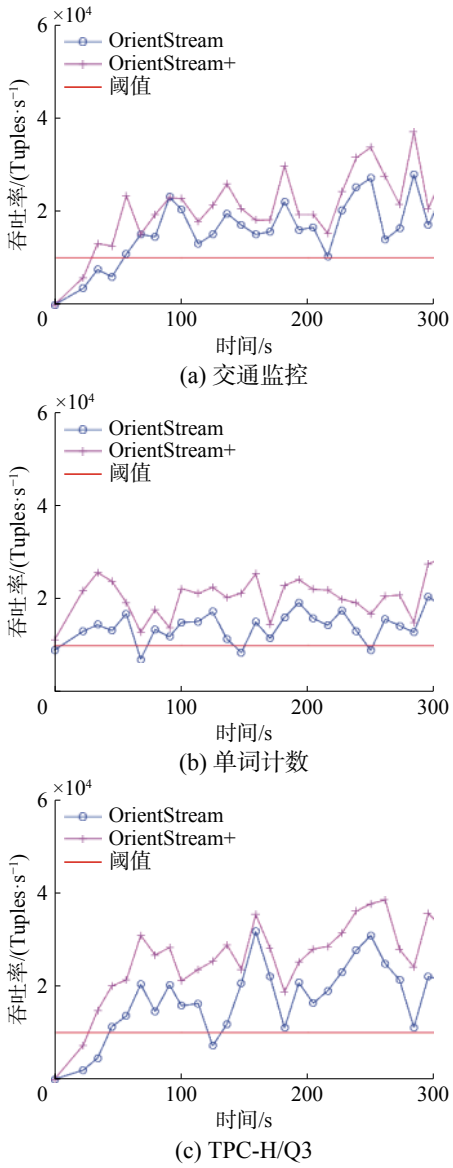

图 4　不同查询任务的吞吐率

Fig. 4　The throughput of different workloads

表 1　预测 CPU 使用情况的 MAE 和 RAE 值

Table 1　Mean and relative absolute error per method for CPU usage prediction

模型	交通监控(TM)		单词计数(WC)		TPC-H(Q3)	
	MAE值	RAE值	MAE值	RAE值	MAE值	RAE值
EDKRegression	4.232 6	0.231 5	3.893 1	0.172 9	4.525 3	0.252 1
ODRegression	3.863 5	0.173 2	3.591 7	0.126 6	4.183 7	0.208 1

表 2　预测内存使用情况的 MAE 和 RAE 值

Table 2　Mean and relative absolute error per method for memory usage prediction

模型	交通监控(TM)		单词计数(WC)		TPC-H(Q3)	
	MAE值	RAE值	MAE值	RAE值	MAE值	RAE值
EDKRegression	2.271 2	0.108 1	2.203 7	0.110 5	2.524 1	0.153 2
ODRegression	1.937 6	0.082 7	1.869 2	0.078 1	2.232 7	0.097 5

根据该实验结果, 可以得出如下结论:

1) 预测内存使用情况的相对绝对误差 (RAE) 的平均值比预测 CPU 使用率的略低, 这是因为内存使用率的波动幅度没有 CPU 使用率的波动幅度大。

2) 在不同查询任务下预测 CPU 的使用情况，ODRegression 模型优于 EDKRegression，其中，平均绝对误差值 (MAE) 可降低 0.3~0.37，相对绝对误差值 (RAE) 可降低 4.4%~5.8%。在预测内存使用情况方面，ODRegression 模型也优于 EDKRegression，其中，平均绝对误差值 (MAE) 可降低 0.29~0.33，相对绝对误差值 (RAE) 可降低 2.5%~5.6%。

3.4　动态资源配置

根据增量学习模型的预测结果和在线参数配置策略，我们监控了 3 个查询任务的整体执行过程。如图 5 和图 6 所示，相对于固定参数配置的查询过程而言，ORDegression 算法分别可节省 10%~16% 的 CPU 使用率和 32%~45% 的内存使用率。相对于使用 EDKRegression 算法的参数配置策略而言，ORDegression 算法分别可节省 1.6%~4.3% 的 CPU 使用率和 4.5%~8% 的内存使用率。

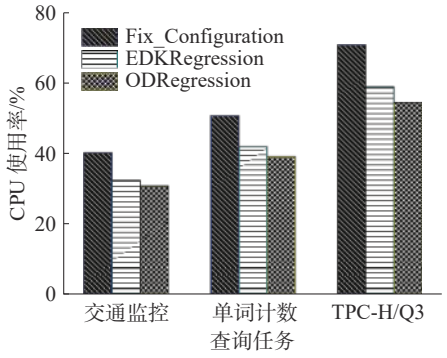

图 5　CPU 使用率

Fig. 5　The usage of CPU

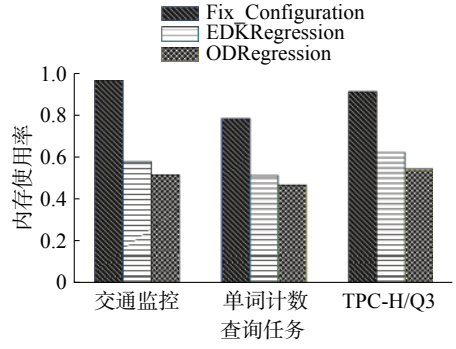

图 6　内存使用率

Fig. 6　The usage of memory

4　结束语

为应对易变数据流的查询请求，频繁改变资源配置会导致系统处理的延迟增加，降低系统性能。针对此问题，本文提出了 OrientStream+框架。根据用户自定义数据处理的延迟阈值，设定以阈值为间隔片段的微批量样式的数据流传输机制；并利用多级别管道缓存，对相同配置的数据流进行批量处理，再按照数据流的时间戳，获取精准查询结果；根据训练数据的持续增长和动态变化的特性，引入具有异常检测功能的增量学习模型，用于进一步提高 OrientStream+的预测精度。最后，我们在 Storm 上实现了上述资源配置框架，并进行了大量的实验。实验结果表明，本文所提出的 OrientStream+框架可在显著降低系统资源使用的情况下，进一步降低系统的处理延迟并提高系统的吞吐率。

针对窗口内的易变数据流，文本利用多级缓存和增量学习的方法以获取较优解。接下来，根据速率无重复波动的频繁变化问题，我们需要设计更加高效的数据缓存策略,使系统更加稳定和健壮。

参考文献：

[1] 孙大为, 张广艳, 郑纬民. 大数据流式计算: 关键技术及系统实例 [J]. 软件学报, 2014, 25(4): 839–862.
SUN Dawei, ZHANG Guangyan, ZHENG Weimin. Big data stream computing: technologies and instances[J]. Journal of software, 2014, 25(4): 839–862.

[2] 崔星灿, 禹晓辉, 刘洋, 等. 分布式流处理技术综 [J]. 计算机研究与发展, 2015, 52(2): 318–332.
CUI Xingcan, YU Xiaohui, LIU Yang, et al. Distributed stream processing: a survey[J]. Journal of computer research and development, 2015, 52(2): 318–332.

[3] 王春凯, 孟小峰. 分布式数据流关系查询技术研究 [J]. 计算机学报, 2016, 39(1): 80–96.
WANG Chunkai, MENG Xiaofeng. Relational query techniques for distributed data stream: a survey[J]. Chinese journal of computers, 2016, 39(1): 80–96.

[4] TOSHNIWAL A, TANEJA S, SHUKLA A, et al. Storm@twitter[C]//Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. Snowbird, Utah, USA, 2014: 147-156.

[5] ZHENG Yu, ZHANG Lizhu, XIE Xing, et al. Mining interesting locations and travel sequences from GPS trajectories[C]//Proceedings of the 18th International Conference on World Wide Web. Madrid, Spain, 2009: 791-800.

[6] WANG Chunkai, MENG Xiaofeng, GUO Qi, et al. OrientStream: a framework for dynamic resource allocation in distributed data stream management systems[C]//Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. Indianapolis, Indiana, USA, 2016: 2281-2286.

[7] WANG Chunkai, MENG Xiaofeng, GUO Qi, et al. Automating characterization deployment in distributed data stream management systems[J]. IEEE transactions on knowledge and data engineering, 2017, 29(12): 2669–2681.

[8] SAX M J, CASTELLANOS M, CHEN Qiming, et al. Ae-

olus: an optimizer for distributed intra-node-parallel streaming systems[C]//Proceedings of 2013 IEEE 29th International Conference on Data Engineering. Brisbane, Australia, 2013: 1280-1283.

[9] FU T Z J, DING Jianbing, MA R T B, et al. DRS: dynamic resource scheduling for real-time analytics over fast streams[C]//Proceedings of 2015 IEEE 35th International Conference on Distributed Computing Systems. Columbus, OH, USA, 2015: 411-420.

[10] BITRAN G R, MORABITO R. State-of-the-art survey: open queueing networks: optimization and performance evaluation models for discrete manufa cturing systems[J]. Production and operations management, 1996, 5(2): 163–193.

[11] ANIELLO L, BALDONI R, QUERZONI L. Adaptive online scheduling in storm[C]//Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems. Arlington, Texas, USA, 2013: 207-218.

[12] KHOSHKBARFOROUSHHA A, RANJAN R, GAIRE R, et al. Resource usage estimation of data stream processing workloads in datacenter clouds[J]. arXiv: 1501.07020, 2015.

[13] BISHOP C M. Mixture density networks[R]. Birmingham, UK: Aston University, 1994.

[14] POGGI N, CARRERA D, CALL A, et al. ALOJA: a systematic study of Hadoop deployment variables to enable automated characterization of cost-effectiveness[C]//Proceedings of 2014 IEEE International Conference on Big Data. Washington, DC, USA, 2014: 905-913.

[15] Apache Hadoop[EB/OL].[2019-04-20]. http://hadoop.apache.org/.

[16] BERRAL J L, POGGI N, CARRERA D, et al. ALOJA-ML: a framework for automating characterization and knowledge discovery in hadoop deployments[C]//Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Sydney, NSW, Australia, 2015: 1701-1710.

[17] JAMSHIDI P, CASALE G. An uncertainty-aware approach to optimal configuration of stream processing systems[C]//Proceedings of 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems. London, UK, 2016: 39-48.

[18] VAN AKEN D, PAVLO A, GORDON G J, et al. Automatic database management system tuning through large-scale machine learning[C]//Proceedings of the 2017 ACM International Conference on Management of Data. Chicago, Illinois, USA, 2017: 1009-1024.

[19] ABADI M, AGARWAL A, BARHAM P, et al. TensorFlow: large-scale machine learning on heterogeneous distributed systems[J]. arXiv: 1603.04467, 2016.

[20] LI Jiexing, KÖNIG A C, NARASAYYA V, et al. Robust estimation of resource consumption for SQL queries using statistical techniques[J]. Proceedings of the VLDB endowment, 2012, 5(11): 1555–1566.

[21] AKDERE M, ÇETINTEMEL U, RIONDATO M, et al. Learning-based query performance modeling and prediction[C]//Proceedings of 2012 IEEE 28th International Conference on Data Engineering. Washington, DC, USA, 2012: 390-401.

[22] Kafka[EB/OL].[2019-04-20]. http://kafka.apache.org/.

[23] SAX M J, CASTELLANOS M. Building a transparent batching layer for storm. HPL-2013-69[R]. Palo Alto, California, USA: HP Labs, 2014.

[24] JOHN G H, LANGLEY P. Estimating continuous distributions in Bayesian classifiers[C]//Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. Montréal, Qué, Canada, 1995: 338-345.

[25] HOEFFDING W. Probability inequalities for sums of bounded random variables[J]. Journal of the American statistical association, 1963, 58(301): 13–30.

[26] OZA N C, RUSSELL S. Experimental comparisons of online and batch versions of bagging and boosting[C]//Proceedings of the Seventh ACM SIGKDD International conference on Knowledge Discovery and Data Mining. San Francisco, California, USA, 2001: 359-364.

[27] AHA D W, KIBLER D, ALBERT M K. Instance-based learning algorithms[J]. Machine learning, 1991, 6(1): 37–66.

[28] HiBench[EB/OL].[2019-08-10]. https://github.com/intel-hadoop/HiBench/.

[29] TPC-H. TPC-H is a decision support benchmark[EB/OL]. [2019-08-10]. http://www.tpc.org/tpch.

作者简介：

王春凯，男，1981 年生，博士后，主要研究方向为数据流管理、知识融合。曾主持和参与中国博士后科学基金项目、国家重点研发计划项目、国家自然科学基金项目以及其他横向课题的研究。发表学术论文 10 余篇。

庄福振，男，1983 年生，副研究员。主要研究方向为迁移学习、数据挖掘、机器学习。曾主持和参与国家重点研发计划项目、国家”863”计划项目、”973”子课题、国家自然科学基金项目以及其他横向课题的研究。发表学术论文 40 余篇。

史忠植，男，1941 年生，研究员。主要研究方向为智能科学、人工智能、机器学习、知识工程等。1979 年、1998 年、2001 年均获中国科学院科技进步二等奖，1994 年获中国科学院科技进步特等奖，2002 年获国家科技进步二等奖。发表学术论文 400 余篇，出版专著 5 部。