软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2022,33(9):3437−3452 [doi: 10.13328/j.cnki.jos.006298]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

ApproxECIoT: 一种基于自适应分层采样的边缘计算新架构*

张德干[1,2], 颜浩然[1,2], 张 捷[3], 张 婷[1,2], 王嘉旭[1,2]

[1](天津理工大学 计算机科学与工程学院, 天津 300384)

[2](计算机视觉与系统教育部重点实验室 (天津理工大学), 天津 300384)

[3](北京交通大学 电子信息工程学院, 北京 100044)

通信作者: 颜浩然, E-mail: yhr99999@126.com

摘 要: 随着物联网技术的发展, 目前的物联网系统产生的数据量越来越多, 这些数据持续不断的传输到数据中心, 传统的物联网数据处理分析系统效率低下且无法处理数量如此庞大的数据流. 另外, 物联网智能设备存在资源受限的特性, 在分析数据时这一特性是不可忽略的. 提出一种适用于物联网实时数据流处理的新架构 ApproxECIoT (approximate edge computing Internet of Things), 实现了一种自调整分层采样算法, 用于处理物联网系统中产生的实时数据流. 该算法在维持已给出的资源预算不变的情况下, 根据每层方差的大小进行样本层内大小的调整, 这对于资源有限的情况下提高计算结果准确度是非常有益的. 最后使用模拟数据流和真实数据流进行实验分析, 结果表明 ApproxECIoT 在边缘节点资源有限的情况下, 仍能获得具有较高准确度的计算结果.

关键词: 物联网; 边缘计算; 近似计算; 数据分析; 实时数据流处理

中图法分类号: TP393

中文引用格式: 张德干, 颜浩然, 张捷, 张婷, 王嘉旭. ApproxECIoT: 一种基于自适应分层采样的边缘计算新架构. 软件学报, 2022, 33(9): 3437–3452. http://www.jos.org.cn/1000-9825/6298.htm

英文引用格式: Zhang DG, Yan HR, Zhang J, Zhang T, Wang JX. ApproxECIoT: New Edge Computing Architecture Based on Adaptive Stratified Sampling. Ruan Jian Xue Bao/Journal of Software, 2022, 33(9): 3437–3452 (in Chinese). http://www.jos.org.cn/1000-9825/6298.htm

ApproxECIoT: New Edge Computing Architecture Based on Adaptive Stratified Sampling

ZHANG De-Gan[1,2], YAN Hao-Ran[1,2], ZHANG Jie[3], ZHANG Ting[1,2], WANG Jia-Xu[1,2]

[1](School of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300384, China)

[2](Key Laboratory of Computer Vision and System, Ministry of Education (Tianjin University of Technology), Tianjin 300384, China)

[3](School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China)

Abstract: With the development of the Internet of Things (IoT) technology, the current amount of data generated by the IoT system is increasing, and the data is continuously transmitted to the data center. The traditional IoT data processing and analysis system is inefficient and cannot handle such a large number of data streams. In addition, IoT smart devices have a resource-limited feature, which cannot be ignored during data analysis. This study proposes a new architecture ApproxECIoT (approximate edge computing IoT) suitable for real-time data stream processing of the IoT. It realizes a self-adjusting stratified sampling algorithm to process real-time data streams. The method adjusts the size of the sample strata according to the variance of each stratum while maintaining the given memory budget. This is beneficial to improving the accuracy of the calculation results when resources are limited. Finally, the experimental analysis is performed using simulated datasets and real-world datasets. The results show that ApproxECIoT can still obtain high-accuracy calculation results even with limited resources of the edge nodes.

Key words: Internet of Things (IoT); edge computing; approximate computing; data analysis; real-time data stream processing

* 基金项目: 国家自然科学基金 (61571328); 天津市自然科学基金 (18JCZDJC96800); 天津市重大科技专项 (17YFZCGX00360)
收稿时间: 2020-05-21; 修改时间: 2020-09-16, 2020-12-15; 采用时间: 2020-12-31; jos 在线出版时间: 2022-07-15

随着计算机网络技术的飞速发展, 物联网 (Internet of Things, IoT) 的发展趋势也是突飞猛进. 传统的物联网数据处理分析系统是将所有采集数据都传输到数据处理中心并进行计算和存储[1−10]. 针对这种中心化数据处理模式计算效率低下、边缘设备资源无法得到充分利用的情况, 本文提出一种基于自适应分层采样的边缘计算架构 ApproxECIoT. 这种架构在保证计算效率和计算结果精度的条件下, 能够对物联网中产生的大量数据流进行高效的分析和计算. 为了实现这个目标, 引入近似计算 (approximate computing, AC) 技术来解决传统物联网架构中计算效率低下的问题. 物联网智能感知设备会产生大量的实时数据流, 通过近似计算可以在保证计算结果的精度符合用户要求的前提下快速获得计算结果, 从而实现更低的时延和更高效的资源利用. 另一方面, 为了获得更精确的计算结果而花费更多的资源有可能会导致成本和收益不成正比的情况. 因此, 利用近似计算技术来保证所花费的资源成本能够获得最大的收益, 同时又能满足用户的需求.

1 相关研究工作

目前, 随着物联网相关技术的进步和发展, 物联网环境中产生的数据量越来越多, 对于一些实时的物联网系统, 大量的数据都是实时产生的, 连续不断的到达处理中心, 这就形成了实时数据流. 传统物联网架构的数据处理方式是将各种终端传感设备的采集信息全部传输到数据处理中心进行计算, 但目前的物联网系统中加入了更多的智能终端传感设备, 从而产生了更多的数据, 使得原有的物联网系统不再适应这种大量数据的实时数据流处理.

文献 [11] 提到, 为了降低传输时延和带宽压力, 采用了 CISCO 提出的雾计算模型, 将计算任务放到边缘节点或雾节点. 但是这对于边缘节点来说, 可供使用的资源是有限的, 并不能胜任所有的计算任务. 而且没有考虑边缘节点计算任务的负载均衡. 文献 [12] 根据 IoT 平台架构标准, 设计了 SAT-IoT 架构, 通过实现 IoT 数据流动态路由模块来选择具有更优的计算条件的处理节点. 虽然实现了合理选择更优的计算节点, 但是大量的数据流都传输到计算节点, 不仅对传输带宽造成了较大的压力, 而且节点的计算效率也会大大降低. 文献 [13] 基于边缘智能计算设计了一种应用于卫星物联网的架构, 这种架构的核心层包括边缘节点层、云节点层、数据服务中心. 尽管提出了让云节点承担计算任务的策略, 但是同样没有考虑云节点计算任务的负载均衡. 文献 [14] 采用近似计算技术设计了物联网安全策略, 通过近似计算降低计算时所需的资源, 在考虑物联网安全性的同时又保证了物联网设备的低功耗. 文献 [15] 提出了一种资源高效利用的边缘计算架构, 设计了延迟感知任务图分区算法和最佳虚拟机选择方法, 从而实现高效的计算任务负载均衡. 在最大化利用边缘节点设备资源的同时满足其 QoS 要求.

从当前所提出的物联网计算架构来看, 传统的中心化数据处理模型存在数据中心传输和计算压力大, 计算效率低, 建设成本高的问题, 最近新提出的边缘计算架构对将产生大量数据流的 IoT 系统来说是一种较好的解决方案[16−20], 但是如何资源高效地实现计算任务在边缘节点的均衡负载, 提高资源利用率和计算效率, 降低用户查询延时, 仍是一个有待深入研究的课题. 因此, 将近似计算 AC 技术引入到物联网数据处理系统中, 在提高边缘智能节点资源利用率的同时, 又能够应对海量设备产生的实时数据流. 近似计算技术的关键就是使用一部分数据能够快速计算出结果, 同时还要保证计算结果能够满足一定的精度要求, 使近似计算结果得到质量保证. 引入近似计算技术, 不仅能提高计算效率, 还能使物联网系统中节点的有限资源能够得到高效利用.

2 ApprxoECIoT 边缘计算新架构设计

ApproxECIoT 架构面向物联网环境中的实时数据流处理, 那么就要考虑实时数据流的一些特性. 数据流中的每个元素都是连续不断到达的, 而且由于数据量非常庞大, 存储全部数据项并不现实. 所以考虑对实时数据流进行采样, 即接收一部分具有代表性的样本, 通过这些样本进行计算, 在节约资源的同时也能提高计算效率.

ApproxECIoT 的架构如图 1 所示, 最接近物理环境的一层是连接的各类传感器, 大多数传感器为了节约自身能量、延长生命周期, 都会采用定时的方式传输所采集的数据, 无线传感器网络在不同时刻都会有传感器加入, 那么这些底层传感器采集的数据就会形成实时数据流传输到上层的节点, 这些节点距离传感器较近, 只需要较少的传输资源. 这些节点可以是边缘物联网服务器、智能控制器、智能网关等设备. 根据边缘计算的概念, 从数据源到

云计算中心路径上的任意资源都属于边缘设备资源, 因此考虑让边缘设备的资源得到充分的利用, 从而提高计算效率. 当用户向云处理中心提交查询任务时, 云处理中心将计算任务划分为多个子计算任务分发给边缘节点并建立这些子计算任务的动态关联图, 边缘节点计算完毕后将子计算的结果发送到云处理中心, 得到最终的计算结果. 如果计算结果的精度满足用户要求, 则直接将结果返回给用户; 否则, 通知边缘节点对样本层内大小进行调整, 并重新采样计算.

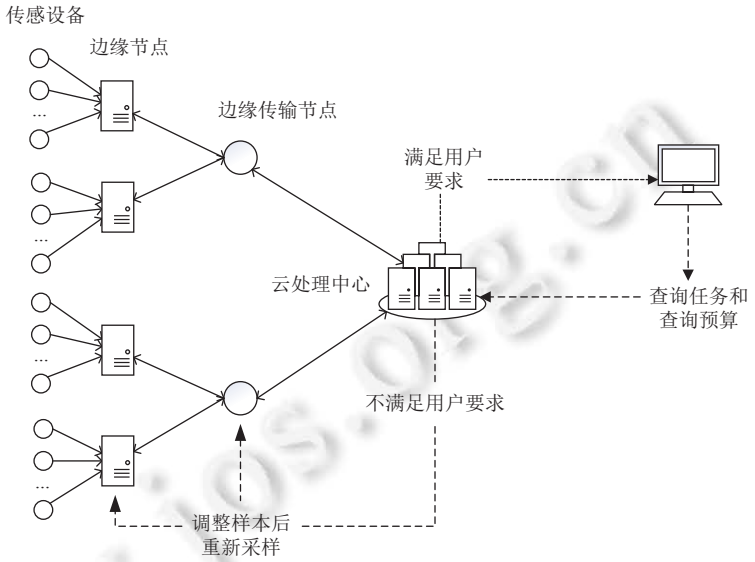


图 1 ApproxECIoT 架构

2.1 动态滑动窗口

定义 1. 动态滑动窗口. 使用 SW 来表示, 滑动窗口的大小为 SW_{len}, 分配的缓冲区的大小为 ω, 其中, ω 是一个可以动态调整的参数, ω 的大小受传感器节点产生数据流的速率以及边缘节点可用资源数量的影响, 当数据流速率越大, 可用内存资源较多时, ω 的值越大; 反之, 则 ω 的值越小. 那么滑动窗口总的大小为 $SW_{\text{len}}+\omega$.

参数 ω 的变化受数据流速率大小和可用内存资源大小的影响. 数据流速率即在时间单位内到达的数据项数目. 单位时间内到达的数据项数据越多, 数据流速率越大. 数据流速率是实时变化的, 影响物联网系统的数据流速率的因素有很多, 例如环境的变化、网络通信链路质量等. 节点承担的职责包括数据传输、数据流分层、数据计算等, 与此相关的进程都需要一定的内存资源. 除去这些资源以外, 剩余的就是可用内存资源. 参数 ω 的动态变化受到数据流速率和可用内存资源大小的影响. 将可用内存资源大小记为 M_e, 总的内存资源大小用 M_t 表示, 数据流速率记为 r_S. 当滑动窗口开始接收数据流, 设置计时器用来记录填“满”滑动窗口所用的时间, 用 t 表示. 同时, 设置到达的数据项数量计数器, 用来记录到下一次滑动窗口开始接收数据项时到达的数据项数量, 用 a 来表示, a 的值大于 SW_{len}, 因为窗口下一次开始接收到达的数据项之前, 需要分配新的内存资源, 而此时到达的数据项就没有办法接收, 即产生丢失. 参数 ω 根据下式进行调整:

$$\omega=\frac{M_e}{M_t}\left(\frac{a-SW_{\text{len}}}{a}\times SW_{\text{len}}+2tr_S\right) \tag{1}$$

公式 (1) 考虑了数据项接收的丢失率以及数据流速率, 同时也考虑了节点的可用内存资源大小. 丢失率大, 就说明当前的滑动窗口过小, 应该增大. 根据数据流速率的大小扩展滑动窗口, 数据流越大, 分配的资源就越多. 另外, 节点的可用资源数量是有限的, 所以也需要考虑节点的可用资源数量.

只有当滑动窗口已“满”, 才需要分配新的窗口缓冲区, 所以将滑动窗口开始接收数据项, 直到窗口“满”, 并且新的窗口缓冲区已经分配完毕的过程定义为一个周期 T. 当新的周期开始时, a 和 t 都需要进行重置.

2.2 ApproxECIoT 数据采样

在分层采样中, 每个层的样本大小分配是一个非常重要的问题. 目前已有的研究[21,22]都是固定分配样本大小, 样本的大小会影响输出结果的精度, 考虑对 Reservoir 大小根据误差估计机制的反馈进行动态调整, 因为某一层的样本可能只需要较少的数据项就能有较高的精度, 那么固定分配给该层的样本可能有一部分是多余的. 文献 [15] 通过计时器将原始数据流进行分层, 并且通过采样概率的倒数来定义每层的权值, 通过误差估计反馈机制来调整权值大小, 从而调整样本大小. 文献 [16] 通过分析误差边界来求出每层样本的最优大小, 并考虑通过启发式算法来降低算法复杂度. 但是在调整样本大小后, 如何保证采样过程仍是均匀随机采样, 不会破坏已有的样本的统计特征, 这需要深入研究. 本文所提出的 ApproxECIoT 对数据流进行分层采样的设计如图 2 所示.

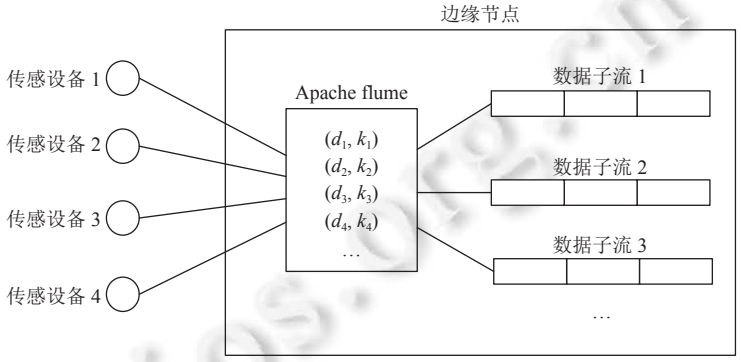


图 2 ApproxECIoT 分层采样设计

ApproxECIoT 的数据采样设计如图 3 所示, 采用第 2.1 节中所定义的动态滑动窗口接收实时到达的数据子流, 基于动态滑动窗口对数据流进行采样. 本文提出一种新的自适应分层随机采样算法, 该算法考虑了物联网系统中可用资源的有限性, 在使用有限资源的情况下, 通过动态调整每层样本的大小, 尽可能获得更精确的结果. 假设用户给定的总样本可用资源大小为 M, 即总共可以采样 M 个数据项, 应当将总的样本容量合理地分配到每一层. 当样本按比例分配时, 可以获得比较稳定的抽样效果, 拥有较好的抽样精度. 因此算法初始化时, 采用按比例分配的方式为每层分配样本大小. 通过 Flume 框架, 将数据流划分为 r 层, 每层所对应的滑动窗口中数据项的数量为 N_h, $1 \leqslant h \leqslant r$. 用 N 表示所有滑动窗口中的数据项总数, 则有:

$$N = \sum_{h=1}^{r} N_h \tag{2}$$

每层样本容量的大小为 n_h, 用 n 表示总的样本容量, 有:

$$n = \sum_{h=1}^{r} n_h \tag{3}$$

那么, 根据公式 (2) 和公式 (3), 每层样本的权值 W_h 为:

$$W_h = \frac{N_h}{N} = \frac{n_h}{n} \tag{4}$$

当采用按比例分配的方式来分配每层的样本大小时, 也就是按照每层的权值来分配样本, 则第 h 层的样本容量为:

$$n_h = M \times W_h \tag{5}$$

在自适应分层随机采样算法中, 本文参考了 Reservoir 采样. Reservoir 采样是一种用于选择简单随机样本的随机算法, 属于不放回采样. 对于一个未知大小的数据集合, 仅仅遍历一次数据集合就能采集样本. Reservoir 采样通常用于未知大小的数据集, 可能数据集合非常大, 无法将全部的数据进行存储, 而且整个数据集只能遍历一次的情况. 实时数据流均符合以上条件, 因此自适应分层随机采样引用 Reservoir 采样是可行的. Reservoir 采样的过程如

下: 使用 Reservoir 采样算法对一个数据流 S 进行采样, 样本大小为 k, 首先用 S 的前 k 项填充样本, 然后从 S 的第 $i=k+1$ 项开始迭代遍历剩余的数据项, 对于 S 中的第 i 项, 在 $(1,i)$ 生成一个随机数 j, 如果 $j\leqslant k$, 那么就使用 Reservoir 数组中的第 j 项替换成 S 的第 i 项, 否则就遍历下一项.

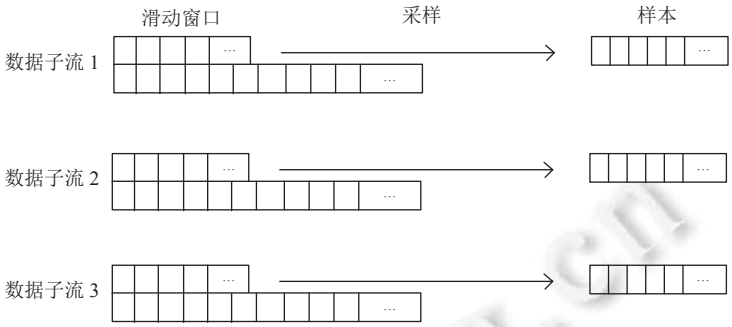


图 3 ApproxECIoT 数据采样设计

引理 1. Reservoir 采样获得的每个数据项样本都是以相同的概率被选择的.

众所周知, 对计算结果的精度要求越高, 那么消耗的资源就越多. 但是随着结果提升到一定精度, 投入更多的资源可能只会提升非常少的精度. 在物联网数据处理分析系统中, 使用较多的资源来提升精度是不合适的, 因为物联网中节点的可用资源有限. 虽然通过采样可以提升节点的资源利用率, 但是却不能忽略样本的计算结果可能存在较大偏差, 而且用户也无法接受这样的结果. 因此, 在采用近似计算技术的同时, 还要考虑计算结果的误差. 本文所提出的 ApproxECIoT 架构考虑了用户对计算结果的精度要求, 因此需要用户给出期望的结果精度. 但是, 在实际情况中, 系统无法对全部的数据项进行统计, 因此只能根据样本计算出结果的估计值. 所以提出对两个不同时刻的样本进行计算, 获得两个估计值, 并根据这两个估计值来判断计算结果是否符合用户要求. 以计算数据流的平均值为例, 用 $\bar{v}$ 表示计算结果的准确值. 接下来, 取不同时刻的样本进行计算, 假设这两个时刻分别为 t_1 和 t_2, 为了保证这两个时刻采集的样本没有交叉的数据项, 令 t_1 和 t_2 之间的时间间隔至少为 SW_{len}/r_S, 其中, r_S 表示数据流的速率. 假设时刻 t_1 采集得到的样本为 S_1, 时刻 t_2 采集得到的样本为 S_2, 计算结果的估计值分别为 $\bar{v}_{t_1}$ 和 $\bar{v}_{t_2}$. 根据 Hoeffding 不等式[17], 估计的误差范围不超过 ε:

$$P(|\hat{\bar{v}}-\bar{v}|\geqslant\varepsilon)\leqslant 2\mathrm{e}^{-\frac{2n\varepsilon^2}{(b_i-a_i)^2}} \tag{6}$$

其中, $\varepsilon>0$, a_i 表示样本中的最小值, b_i 表示样本中的最大值. 当样本大小确定时, 可以根据公式 (6) 来估计误差范围. 公式 (6) 表示平均值的近似值和准确值之间的误差超过误差边界 ε 的概率, 这个概率可以被解释为, 在准确值 $\bar{v}$ 两侧宽度为 2ε 的置信区间, 其值不会超过一个固定的值:

$$\alpha=P(\hat{\bar{v}}\notin|\bar{v}-\varepsilon,\bar{v}+\varepsilon|)\leqslant 2^{-\frac{2n\varepsilon^2}{(b_i-a_i)^2}} \tag{7}$$

其中, α 表示计算结果犯错的概率, 那么对于置信度 δ, 有 $\delta=1-\alpha$. 那么估计误差边界 ε 可以表示为:

$$\varepsilon_i=(b_i-a_i)\sqrt{\frac{1}{2n_i}\ln\frac{2}{1-\delta}} \tag{8}$$

在两个不同的时刻, 通过对数据流进行采样得到两个无交叉的样本, 根据这两个样本进行计算获得两个估计值. 根据公式 (7) 可以推出, t_1 时刻的样本的估计误差为 $|\bar{v}_{t_1}-\bar{v}|\leqslant\varepsilon_{S_1}$, t_2 时刻的样本的估计误差为 $|\bar{v}_{t_2}-\bar{v}|\leqslant\varepsilon_{S_2}$. 因为在实际情况中, 系统无法计算出准确值 $\bar{v}$, 所以令两式相减可得:

$$|\bar{v}_{t_1}-\bar{v}|-|\bar{v}_{t_2}-\bar{v}|\leqslant\left|\varepsilon_{S_1}-\varepsilon_{S_2}\right|\Rightarrow err=|\bar{v}_{t_1}-\bar{v}_{t_2}|\leqslant\left|\varepsilon_{S_1}-\varepsilon_{S_2}\right| \tag{9}$$

通过对不同时刻的数据流进行 Reservoir 采样, 根据得到的样本获得计算结果的估计值, 如果估计值之差的绝对值在所允许的误差范围内, 就直接输出计算结果. 如果不满足公式 (9), 那么就要考虑调整层的大小, 然后重新对数据流进行采样.

在分层均匀随机采样的基础上, 对分层随机样本的质量进行评估, 可以通过样本求出估计方差, 如果这个样本的估计方差较大, 就说明这个样本的采样效果较差, 这会导致获得的计算结果的估计值产生较大的误差. 针对这种情况, 本文提出一种自适应采样算法, 通过计算每层的样本估计方差的大小, 从而动态调整每层的样本大小. 该算法的核心思想是: 在用户给定的总样本内存 M 不变的条件下, 增大那些方差较大的层的样本大小, 如果要保持 M 大小不变, 就需要在其他层减去增加的大小. 由于减少了层的样本大小, 就必然会导致样本的估计方差增大, 因此考虑在减少这些层的样本大小时, 使得增加的方差尽可能最小.

对于分层随机样本来说, 当各层中每个数据项的抽样费用相同时, 各层样本大小的最优分配方法是 Neyman 分配. 在本架构中, 要求每个样本均为分层随机样本, 同时, 存储每个数据项的内存资源是相同的, 因此考虑采用 Neyman 分配的方式来调整每层样本容量的大小. Neyman 分配的公式如下:

$$n_i = n \times \frac{W_i S_i}{\sum_{i=1}^{r} W_i S_i} = n \times \frac{N_i S_i}{\sum_{i=1}^{r} N_i S_i} \tag{10}$$

其中, S_i 表示第 i 层样本的标准差. 通过上述公式, 在维持原有的内存资源预算 M 不变的情况下, 将样本容量重新进行分配, 并重新采样, 尽可能使得计算结果能够满足用户的精度要求. 从 Neyman 分配的计算公式中可以看出, 层样本方差越大, 分配到的样本容量越多; 反之, 则越少. 因为在原有的内存资源预算不变的情况下调整样本大小, 所以一些层样本容量增加, 必然导致剩余的层样本容量缩减. 理论上来说, 样本容量的减少会导致样本方差增大. 在减少样本容量的同时, 应尽可能使得增加的方差最少. 对数据流进行分层采样后, 每一层子样本的均值可以按照公式 (11) 进行计算:

$$\bar{y}_h = \frac{\sum_{i=1}^{n_h} y_{hi}}{n_h} \tag{11}$$

其中, y_{hi} 表示第 h 层中的第 i 个数据项. 那么总样本的估计均值为:

$$\bar{y} = \frac{\sum_{h=1}^{r} n_h \bar{y}_h}{n} = \sum_{h=1}^{r} W_h \bar{y}_h \tag{12}$$

对于分层随机抽样, 估计均值 $\bar{y}$ 的方差为:

$$V(\bar{y}) = \frac{1}{N^2} \sum_{h=1}^{r} N_h (N_h - n_h) \frac{S_h^2}{n_h} = \sum_{h=1}^{r} \frac{W_h^2 S_h^2}{n_h} - \sum_{h=1}^{r} \frac{W_h S_h^2}{N} \tag{13}$$

其中, S_h 表示第 h 层样本的标准差.

定理 1. Φ 表示样本层的集合, $O \in \Phi$ 表示缩减样本的层的集合, ApproxECIoT 在维持原有的样本大小 M 不变的情况下, 对每层样本大小进行调整, 当缩减第 $i, i \in O$ 层样本时所引起的误差的增加是最小的.

证明: 在样本未缩减之前, 根据等式 (13) 可得, 样本的估计均值方差为 $V(\bar{y}) = \sum_{i=1}^{r} \frac{W_i^2 S_i^2}{n_i} - \sum_{i=1}^{r} \frac{W_i S_i^2}{N}$. 假设样本调整后, 每层样本的大小为 $n'_i, i = 1, 2, ..., r$, 那么此时样本的估计均值方差为 $V'(\bar{y}) = \sum_{i=1}^{r} \frac{W_i^2 S_i^2}{n'_i} - \sum_{i=1}^{r} \frac{W_i S_i^2}{N}$, 用 Δ 表示样本大小改变所引起的误差的变化, 则 $\Delta = V'(\bar{y}) - V(\bar{y}) = \sum_{i=1}^{r} \frac{W_i{}^2 S_i{}^2}{n'_i} - \sum_{i=1}^{r} \frac{W_i{}^2 S_i{}^2}{n_i}$, 期望对于第 $i, i \in O$ 层样本容量的减小所导致的方差增长是最小的, 可以得出下式:

$$\min \left(\sum_{i \in O} \frac{n_i^2 S_i{}^2}{n'_i} - \sum_{i \in O} \frac{n_i^2 S_i{}^2}{n_i} \right) \tag{14}$$

其中, n'_i 应当满足 $\sum_{i=1}^{r} n'_i = M$, 即每个子层的样本大小改变后, 它们的总大小仍要满足用户给定的资源预算. 因为

ApproxECIoT 调整样本大小是采用 Neyman 分配方式, 而 Neyman 是一种最优分配方式, 即采用公式 (10) 进行样本大小调整时, 可以使样本估计均值的方差有最小值, 从而满足公式 (14).

引理 2. 均匀置信度定义如下:

$$U = \frac{K}{P} \times 100\%,$$

其中, K 表示算法获得的相同大小的不同样本数, P 表示统计上可获得的具有相同大小的不同样本数.

引理 3. 在某个采样算法中, 如果所有统计可能的大小相同的样本被选择的可能性相同, 那么该算法产生的样本称为均匀随机样本, 该算法的均匀置信度为 100%.

引理 4. 如果某个采样算法的均匀置信度为 100%, 那么该采样算法获得的样本为均匀随机样本.

引理 5. 在 Reservoir 采样中, 当从一个数据流采样的时候, 如果样本大小从 r 缩减为 $r-\delta\ (\delta>0且\delta\neq r)$, 能够维持缩减后的样本为 100% 的均匀置信度; 如果样本大小从 r 扩大到 $r+\delta\ (\delta>0)$, 不可能维持扩大后的样本为 100% 的均匀置信度.

定理 2. ApproxECIoT 采样获得的样本为均匀随机样本.

证明: 算法第一次进行采样时, 样本中的每一个数据项均是以相同的概率从滑动窗口中选择的, 所以此时得到的样本为均匀随机样本. 如果此时计算结果符合用户要求, 那么该结果是通过均匀随机样本计算得到的; 否则, ApproxECIoT 对每层样本大小进行调整, 如果某一层调整后的大小小于原来的样本大小, 根据引理 5, 缩减后的样本可以维持 100% 的均匀置信度, 所以直接从已采样的样本中随机删除多余的数据项即可. 如果某层调整后的大小大于原来的样本大小, 根据引理 5, 如果直接扩大样本, 得到的样本不是均匀随机样本. ApproxECIoT 采取的策略是删除该层样本中的全部数据项, 然后扩大层样本大小后重新采样, 这样得到的样本中每个数据项是以 $\frac{r+\delta}{N}$ 概率选择的, 所以仍然是均匀随机样本. 所以, ApproxECIoT 可以保证调整每层样本大小后, 获得的样本仍为均匀随机样本.

在样本调整每层的大小后, 每个层的调整结果有 3 种情况, 即增大、缩小、保持不变. 根据引理 5, 只有当层的大小增大时, 样本无法保持 100% 的均匀置信度. 因此, 只有增大的层才会删除该层的全部数据, 然后只对增大的层进行重新采样. 通过这种方式使样本层大小调整所带来的延时尽可能最小. 例如: 假设样本共有 4 层, 每层的大小分别为 r_1、r_2、r_3、r_4, 数据流的大小为 N. 采样时, 第 1 层的每个数据项是以 $\frac{r_1}{N}$ 概率选择的, 第 2 层的每个数据项是以 $\frac{r_2}{N}$ 概率选择, 依次类推. 经过一次层大小调整后, 每层的大小变为 $r_1-\delta_1$、$r_2+\delta_2$、$r_3-\delta_3$、$r_4+\delta_4$, 即第 1、3 层缩小了, 第 2、4 层增大了, 但是总的样本大小没变. 根据引理 5, 可以通过从原来的层样本中随机删除多余的数据项来实现层的缩小, 且层中每个数据项被选择的概率仍是相同的. 那么第 1 层缩小后得到的数据项是以 $\frac{r_1-\delta_1}{N}$ 概率得到的、第 3 层缩小后得到的数据项是以 $\frac{r_3-\delta_3}{N}$ 概率得到的. 而对于第 2 层, 即使选择一个大小为 N 的新数据流, 并在其基础上直接采样 δ_2 个数据项并放入到第 2 层, 以实现第 2 层的增大. 这同样没有办法保证第 2 层为均匀随机样本, 因为第 2 层的前 r_2 个数据项是以 $\frac{r_2}{N}$ 概率选择的, 而后 δ_2 个数据项是以 $\frac{\delta_2}{N}$ 概率选择的, 明显 $\frac{r_2}{N}\neq\frac{\delta_2}{N}$. 所以, 如果采用这种方式, 那么第 2 层的样本就不是均匀随机样本. 因此, 对增大的层, ApproxECIoT 会删除该层的样本, 并只对该层重新采样. 这样就可以保证增大的层在增大后获得的样本为均匀随机样本.

根据本文所提出的算法, 在对物联网系统中所产生的实时数据流进行采样时, 通过动态调整分层随机样本的大小, 进一步提高采样精度, 在保证给出用户可靠的计算结果的同时, 又能够实现有限资源的高效利用. 实现 ApproxECIoT 的自适应分层采样算法的伪代码如算法 1.

算法 1. SampleAdjust(A, M).

输入: A (层的集合), $A=\{1,2,...,r\}$, M (总的样本大小);

输出: 对 $i\in A$, L 即调整后每层样本的大小.

$\vartheta \leftarrow \varnothing$;//表示需要缩减样本大小的层的集合

$\Lambda \leftarrow \varnothing$;//表示样本大小增加的层的集合

$R \leftarrow \{R_1, R_2, ..., R_r\}$;//$R_i$表示第$i$层的样本

R_i.var $\leftarrow 0$;//表示第i层样本的估计方差

$L \leftarrow$ 根据公式(10)计算当前的最优样本大小

delta_stratum_n $\leftarrow A.size - L$;//每层样本大小的变化量

FOR $i \in \Lambda$ *DO*

  IF delta_stratum_n > 0 *THEN*

    $\Lambda \leftarrow \Lambda \cup$ GetStratum(i);

  ELSE

    $\vartheta \leftarrow \vartheta \cup$ GetStratum(i);

  ENDIF

ENDFOR

FOR $i \in \Lambda$ *DO*

  clearAll($\Lambda[i]$);//删除第i层所有的原有数据项

ENDFOR

AdaptiveSamplingCompute(M, δ, Λ);//使用算法3重新采样并计算

FOR $j \in \vartheta$ *DO*

  $\Delta n_j \leftarrow |n'_j - n_j|$;

  randomRemove($\vartheta[j], \Delta n_j$);//随机删除Δn_j个数据项

ENDFOR

在算法 1 中, 通过 Neyman 分配计算每次样本调整时的最优样本大小, 然后找出样本大小增加的层和减少的层. 根据 Mohammed Al-Kateb[18]对均匀随机采样提出的观点, 如果采样算法无法以 100% 的均匀置信度获得样本, 那么就不能保证采样算法所得到的样本一定是均匀随机样本. 因为本文要求对数据流的采样必须是均匀随机采样, 所以对于样本容量增加的层, 如果对增加的容量直接执行 Reservoir 采样, 无法保证获得的样本是均匀随机样本, 所以样本容量增加的层需要先移除之前采集的数据项, 并重新采样. 对于样本容量缩减的层, 可以在原来采集的数据项的基础上直接移除需要缩减的数据项的数量, 且移除后仍能保证获得的样本为均匀随机样本.

根据前面提出的结果误差范围的计算方式, 如果计算结果在给定的误差范围内, 即满足用户的精度要求, 就输出计算结果; 否则, 就要调整样本大小, 然后重新采样并计算, 具体的如算法 2 所示.

算法 2. AdaptiveSamplingCompute(M, δ, Λ).

输入: M (总样本的大小), δ (用户给定的计算结果置信度), Λ (样本增大的层的集合);

输出: $\widehat{\bar{v}}$ (满足用户要求的计算结果).

$\Lambda \leftarrow \{1, 2, ..., r\}$//层的集合

count $\leftarrow 0$;//记录算法的采样次数

$R \leftarrow \{R_{t_1 1}, R_{t_1 2}, ..., R_{t_1 r}\}$;//$t_1$时刻层样本的集合

$SW \leftarrow EfficientSlidingWindow(SW_{\text{len}}, M)$;//获取接收数据流的滑动窗口实例

IF count $= 0$ *THEN*

  //如果是第一次采样, 则按比例进行分配

FOR $i \in \Lambda$ *DO*

$n_i \leftarrow n \times W_i$;

$R_{t_1 i}.data \leftarrow \mathrm{CRS}(SW_i, n_i)$;

ENDFOR

$\widehat{\overline{v}}_{t_1} \leftarrow \mathrm{compute}(R)$;

Sleep(*sample_interval*);//采样间隔

$R \leftarrow \{R_{t_2 1}, R_{t_2 2}, ..., R_{t_2 r}\}$;//$t_2$时刻层样本的集合

FOR $j \in \Lambda$ *DO*

$R_{t_2 j}.data \leftarrow \mathrm{CRS}(SW_j, n_j)$

ENDFOR

$\widehat{\overline{v}}_{t_2} \leftarrow \mathrm{compute}(R)$;

ELSE

//如果不是第一次采样,就只需对样本增大的层进行重新采样

FOR $i \in \Lambda$ *DO*

$R_{t_1 i}.data \leftarrow CRS(SW_i, n_i)$;

ENDFOR

$\widehat{\overline{v}}_{t_1} \leftarrow \mathrm{compute}(R)$;

Sleep(*sampling_interval*);

FOR $j \in \Lambda$ *DO*

$R_{t_2 j}.data \leftarrow CRS(SW_j, n_j)$;

ENDFOR

$\widehat{\overline{v}}_{t_2} \leftarrow \mathrm{compute}(R)$;

ENDIF

$\widehat{\overline{v}} \leftarrow |\widehat{\overline{v}}_{t_1} - \widehat{\overline{v}}_{t_2}|$;

$\epsilon \leftarrow$ 使用公式(8)计算误差边界

IF $\widehat{\overline{v}} < \epsilon$ *THEN*

Memo($\widehat{\overline{v}}$);//缓存计算结果

return $\widehat{\overline{v}}$;

ELSE

SampleAdjust(Λ, M);//采用算法2自适应调整样本大小

$count \leftarrow count + 1$;

ENDIF

2.3 ApproxECIoT 计算任务处理策略

定义 2. 发生改变的输入. 在每个边缘节点, 如果算法调整了样本大小, 并且重新采样和计算, 且新的估计计算结果满足用户要求, 即在限定误差范围内, 那么就认为这个新的计算结果是发生改变的输入, 即该节点进行了重新计算.

本文所提出的 ApproxECIoT 架构的计算任务的处理策略如算法 3 所示. 对于边缘节点来说, 子计算的输入数据就是样本, 在提交的查询任务的时间间隔没有超过设定阈值的条件下, 符合用户精度要求的样本是不会发生变化的, 此时子计算的结果是可重用的; 否则, 就要重新采样并计算. 对传输节点来说, 它的子计算任务所依赖的输入是子节点的计算结果, 如果用户提交的查询任务的时间间隔超过设定的阈值, 那么它就要通知它的子节点进行重新计算或重新采样. 最终, 所有节点的子计算结果都要发送给各自的父节点, 直到根节点, 即云数据处理中心.

算法 3. SelfAdjustComputation(input_Data).

输入: *input_Data* (节点计算任务的输入数据). 对于边缘节点来说, 输入数据是否符合用户要求的样本; 对传输节点来说, 输入数据是子节点的计算结果. 如果 *input_Data* 的值为 NULL, 则表示没有发生改变的输入;
输出: *result* (节点的子计算任务的计算结果).

```
ζ ← query_interval;//提交查询任务时间间隔的阈值
t_cur ← 用户提交的当前查询任务的时间点
t_pre ← 用户上一次提交查询任务的时间点
WHILE true DO
    IF receive input_Data is NULL THEN
        result ← GetMemo();//输入数据未发生改变,直接重用先前的计算结果
    ELSE
        UPDATE result by input_Data;//重新进行计算,更新计算结果
    ENDIF
    IF t_cur − t_pre > ζ THEN
        switch(NodeType){
            case(EdgeNode) :
                AdaptiveSamplingCompute(M, δ, Λ);
                //重新采样,需要将算法3中的count变量重置为0.
            case(TransmitNode) :
                NotifyChildNodes();
                //通知所依赖的子节点,重新采样并更新计算结果
            }
        ENDIF
        SendResultToParent();//将子计算的结果发送到父节点
ENDWHILE
```

在根节点, 使用 *DDG* 表示创建的管理所有子计算任务的动态依赖关系图, 有 $DDG=(V,E)$, 其中顶点集合 V 表示分发给所有的传输节点和边缘节点的子计算任务, 边的集合 E 表示所有子计算任务之间的数据关系和控制依赖关系. 根节点即云数据中心, 当用户提交查询任务时, 云数据中心将查询任务划分成多个子计算任务并分发给子节点, 同时建立 *DDG*.

3 实验测试与分析评估

本节对 ApproxECIoT 的性能进行测试评估, 采用模拟数据流和真实世界的数据流来评估 ApproxECIoT 的表现, 通过和简单随机采样 (simple random sampling, SRS)、StreamApprox、ApproxIoT、CalculIoT 对比进行分析.

首先, 构建物联网边缘计算网络模型. 网络中部署若干个不同类型的无线传感器, 不同类型的传感器采集不同类型的数据, 所有的传感器都将采集的数据传输到最近的边缘节点. 因为边缘节点需要对这些数据进行简单的处理和计算, 并且要将处理结果传输到云服务中心, 中间可能经过若干个传输节点, 所以要求边缘节点和传输节点具有较强的计算能力和通信能力. 本文采用虚拟机构建 4 个边缘节点、2 个传输节点, 1 台真实主机作为云服务中心来模拟边缘计算网络模型, 用一台笔记本电脑作为用户终端, 具体参数配置如表 1 所示.

然后, 假设用户提交的查询任务为: 查询某种类型 (在不同的数据集中有不同的数据类型) 数据的平均值, 计算结果置信度为 95%. 注意, 在实验测试中, 默认总是为用户提供距离当前时刻最近的一段时间的计算结果.

表 1 物联网网络节点参数配置

设备类型	参数
云服务中心	处理器: core i7-7700 @3.60 GHz; 运行内存: 16 GB; 操作系统: Windows 10
传输节点	处理器内核数量: 1; 运行内存: 2 GB; 操作系统: CentOS7
边缘节点	处理器内核数量: 1; 运行内存: 2 GB; 操作系统: CentOS7
用户终端	处理器: AMD Ryzen5 4600H @3.0 GHz; 运行内存: 16 GB; 操作系统: Windows 10

3.1 采用模拟数据流

首先采用模拟数据流对 ApproxECIoT 的性能进行评估, 生成的模拟数据流包含 4 种不同的数据分布, 其中包含 beta 分布、Gaussian 分布、均匀分布以及 Zipf 分布, 每种类型的分布包含 250 000 个数据项, 数据流中总共包含 1 000 000 个数据项. 采用第 2.1 节定义的动态滑动窗口来接收数据项, 然后在滑动窗口的基础上执行采样算法以及数据分析.

对于 Beta 分布, 设置的参数为 $\alpha=5$, $\beta=5$; 正态分布设置的参数为 $\mu=1000$, $\sigma=20$; 均匀分布设置的参数为 $a=1$, $b=3$; Zipf 分布设置的参数为 $n=10$, $\lambda=2$. 评估 ApproxECIoT 在结果精度损失方面的表现, 精度损失定义为 $|\hat{v}-v|/v$, 其中 $\hat{v}$ 表示计算结果的近似值, v 表示准确值, 准确值是通过直接对数据流的全部数据项进行计算得出的.

首先, 考虑不同的采样比对精度损失所产生的影响, 和另外 4 种架构进行对比. 如图 4 所示, 当采样比为 10% 时, ApproxECIoT 比 CalculIoT 的精度损失降低了 89.6%, 比 SRS 采样的精度损失降低了 99.8%. 即 ApproxECIoT 的计算结果拥有更高的准确性, 这是因为 ApproxECIoT 不仅对数据流进行了分层处理, 而且加入了误差控制, 保证计算结果有更小的误差.

再考虑滑动窗口的不同大小对精度损失的影响. 如图 5 所示, 将采样比固定为 10%, 窗口大小从 1 000 增加到 4 000, 可以看到 SRS、ApproxIoT、CalculIoT、ApproxECIoT 的精度损失都是随着窗口增大而降低的, 这是因为当滑动窗口增大时, 样本中缓存的数据项也会增多, 从而降低了计算结果的精度损失. 这也是 ApproxECIoT 在算法初始化时采用按比例分配方式的原因. 窗口大小对 StreamApprox 并没有产生影响, 这是因为 StreamApprox 在窗口的滑动间隔执行采样. 另外, 从图 5 中可以看出当滑动窗口大小相同时, ApproxECIoT 比其他 4 种架构的精度损失小, 这是因为 ApproxECIoT 不仅对数据流采取了分层操作, 而且加入了误差控制策略.

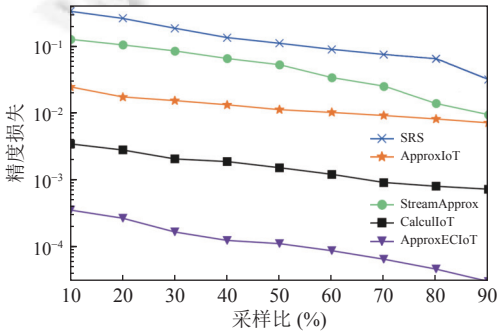


图 4 不同架构之间的精度损失对比

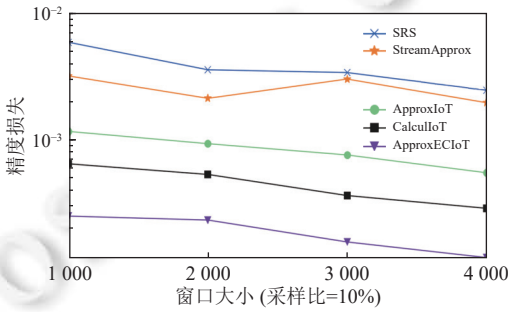


图 5 不同窗口大小情况下精度损失对比

ApproxECIoT 引入了边缘计算和自调整计算技术, 将边缘节点的计算结果进行缓存, 当没有进行重新采样时, 节点的缓存结果是可以重用的, 这极大地提高了任务的计算效率. 如图 6 所示, 当采样比为 90% 时, ApproxECIoT 的计算时间和 SRS 已经非常接近了. 另外, 还可以看出, ApproxECIoT 对应的曲线斜率比其他 4 种架构大, 也就是说, 当采样比增加时, ApproxECIoT 因样本增大而增加的计算时间明显比另外 4 种架构增加的多. 这是因为 ApproxECIoT 考虑了用户对计算结果的精度要求, 加入了误差控制策略, 当计算结果不满足用户要求时, 会调整层之间样本大小并重新采样计算.

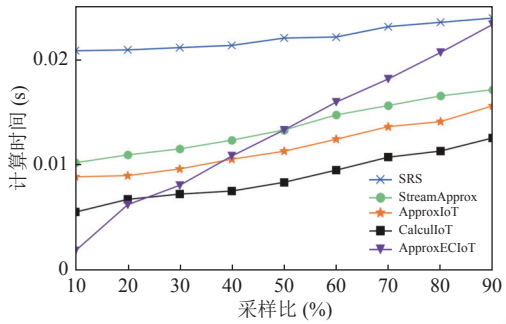


图 6　不同架构之间计算效率对比

在 4 种分布生成的模拟数据流的基础上, 将滑动窗口的大小固定为 10 000, 数据流总共分为 4 层, 分析 ApproxECIoT 算法的样本调整过程. 算法调整过程中的详细参数变化如表 2 所示, 表格按照横向来看, 每个采样比中的一行是一次样本调整.

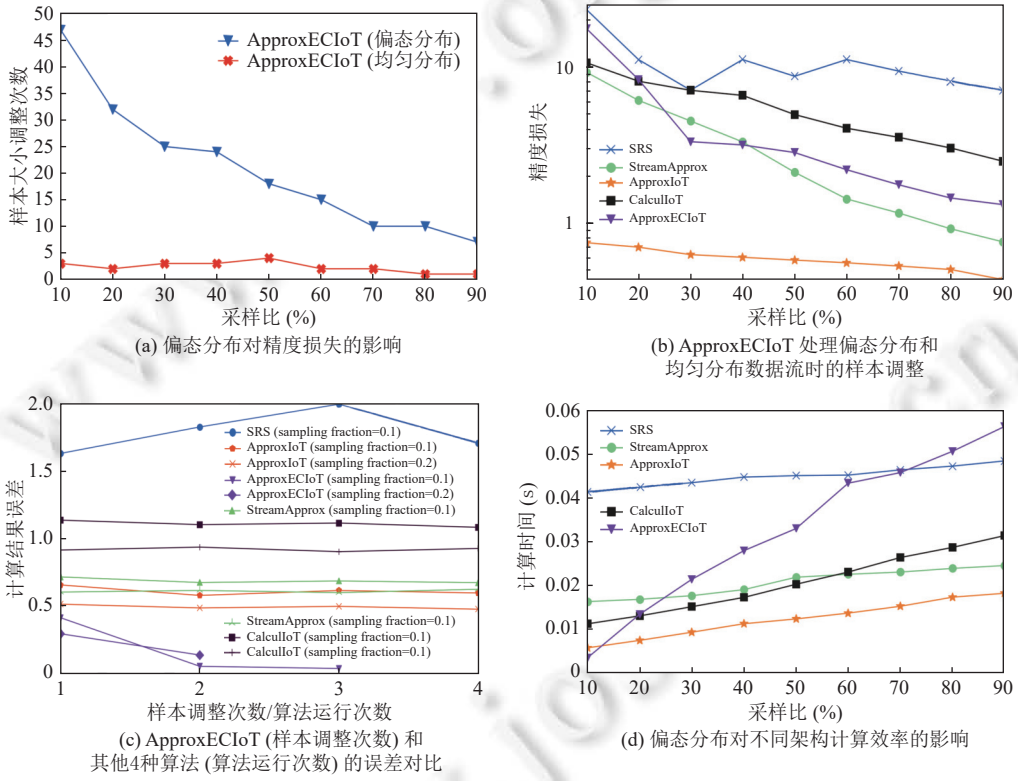
表 2　ApproxECIoT 样本调整过程中的参数变化

采样比 (%)	层1	层2	层3	层4	t_1时刻的计算结果	t_2时刻的计算结果	估计误差	ε之差	准确值	估计值	精度损失	样本调整次数
10	250	250	250	250	251.0668	251.1711	0.10428	0.9749	251.1134	251.1711	2.2967×10^{-4}	0
30	750 20 18	750 2679 2644	750 76 72	750 225 266	251.122 251.1952 251.1372	251.5225 251.8813 251.1484	0.40045 0.31392 0.011258	0.017003 0.26487 0.058545	251.1134	251.1484	1.3937×10^{-4}	2
50	1250	1250	1250	1250	251.29	251.1341	0.1559	0.15625	251.1134	251.1341	8.2459×10^{-5}	0
70	1750 47	1750 6238	1750 181	1750 534	251.1483 251.114	251.0115 251.1148	0.13677 0.0007841	0.031435 0.059494	251.1134	251.1148	5.4239×10^{-5}	1
90	2250 59	2250 8035	2250 230	2250 675	251.35 251.1391	251.1111 251.105	0.23891 0.034119	0.19969 0.052515	251.1134	251.105	3.3461×10^{-5}	1

当算法初始化时, 采用按比例分配的方式分配样本, 如果得到的估计值在限定误差范围内, 则不需要调整样本大小; 否则对样本大小进行调整. 从表 2 可以看出, 在调整样本大小时, 分配给第 2 层的样本是最多的, 这是因为第 2 层的样本质量最差, 对应类型的数据具有较大的不稳定性. 相反, 第 1 层的样本质量最好, 所以只需要分配给它较少的样本就能满足精度要求. ApproxECIoT 框架通过对样本大小的调整, 这对于资源有限的边缘物联网来说, 可以提高物联网实时数据流的处理能力.

接下来, 考虑当输入数据流的分布呈现偏态分布时, 对计算结果精度损失的影响. 通过泊松分布来产生呈现偏态分布的输入数据流, 其中包含 4 种类型的泊松分布, 分别为 $A(\lambda=10)$, $B(\lambda=100)$, $C(\lambda=1000)$, $D(\lambda=1000000)$. 这 4 种不同的泊松分布生成 4 个输入子流, 其中子流 A 占总数据项数目的 80%, 子流 B 占 19.89%, 子流 C 占 0.1%, 子流 D 占 0.01%. 如图 7(a) 所示, 可以看到, StreamApprox 和 ApproxIoT 可以更好的应对呈现偏态分布的数据流, 原因就是 StreamApprox 和 ApproxIoT 都采取了分层策略, 从而不会忽略那些出现频率比较低 (例如子流 D) 的数据项. 而 SRS 和 CalculIoT 并没有采取分层策略, 所以很容易就会忽略那些出现频率低的数据项, 导致计算结果出现较大的误差. 尽管 ApproxECIoT 也采取了分层策略, 但是 ApproxECIoT 的精度损失却比 StramApprox 和 ApproxIoT 的要大. 出现这种情况是因为 ApproxECIoT 在算法在调整样本大小时是根据 Neyman 分配来调整, 有可能某一层的权重比较大, 但是它的方差比较小, 那么在调整样本大小时就可能会缩减这一层的样本容量, 导致 ApproxECIoT 在面对偏态分布的数据流时就产生了比 ApproxIoT 更大的精度损失. 不过由于 ApproxECIoT 拥有计算结果误差检测策略, 当计算结果出现较大的偏差, 可以通过调整样本大小并重新采样来减小误差, 所以随着采样比的增加, ApproxECIoT 精度损失是逐渐降低的.

测试 ApproxECIoT 在计算偏态分布数据流和均匀分布数据流时的表现, 偏态分布数据流仍使用之前的 4 种不同的泊松分布, 均匀分布数据流使用之前由 Beta 分布、高斯分布、均匀分布和 Zipf 分布产生的数据流, 其中每种类型的分布所产生的数据项数量各占 25%. 图 7(b) 中可以看出, ApproxECIoT 在处理偏态分布的数据流时, 样本大小的调整次数明显多于处理均匀分布数据流时的调整次数. 而且, 当处理均匀分布数据流时, 算法的平均样本调整次数为 2.3 次; 当计算偏态分布数据流时, 算法的平均样本调整次数为 20.9 次. 这就说明, ApproxECIoT 在处理均匀分布数据流时, 拥有更高的计算效率, 而处理偏态分布数据流的计算效率则会低一些. 从图 7(c) 中可以看出, ApproxECIoT 在采样比为 10% 的时候, 样本大小调整了 3 次; 当采样比为 20% 的时候, 样本大小调整了 2 次. ApproxECIoT 每次调整样本大小后, 误差都有所减小. 而 SRS 并没有对计算结果出现误差的处理策略, 所以执行多次 SRS 算法, 每次得到的计算结果的误差是随机的. ApproxIoT 和 StreamApprox 虽然给出了计算结果的误差估计, 但是也没有采取误差控制策略, 所以算法运行多次后误差仍没有减小. CalculIoT 由于没有采用数据流分层策略, 所以在处理偏态分布数据时, 计算结果存在较大的误差. 从图 7(d) 中可以看出, 在处理偏态分布数据流时, 当采样比未超过 12% 时, ApproxECIoT 的计算效率是最高的, 但是采样比超过 12% 后, ApproxECIoT 的计算效率开始下降. 从这一点也可以看出, 在处理偏态分布数据流时, ApproxECIoT 的计算效率并不是随着采样比的增加而增加. 因此, 在物联网系统中应用 ApproxECIoT 架构时, 最好设定合适的采样比, 使系统获得最理想的处理效率.



(a) 偏态分布对精度损失的影响

(b) ApproxECIoT 处理偏态分布和均匀分布数据流时的样本调整

(c) ApproxECIoT (样本调整次数) 和其他4种算法 (算法运行次数) 的误差对比

(d) 偏态分布对不同架构计算效率的影响

图 7 实验结果

接下来, 分析 ApproxECIoT 所使用内存资源的大小. 使用含有 Beta 分布、高斯分布、均匀分布和 Zipf 分布的数据流, 对比 ApproxECIoT 和 SRS 对内存资源的使用情况. 如图 8 所示, 当采样比从 10% 变化到 90%, ApproxECIoT 使用的内存资源比 SRS 使用的内存资源要多一点, 但是对于目前大多数智能感知设备的可用内存资源来说, 这些差距是可以忽略不计的. 总体上, 两者所使用的内存资源数量比较接近. SRS 是最简单的采样算法, 它所使用的资源也是最少的. 也就是说, ApproxECIoT 在使用和 SRS 相近内存资源的条件下, 可以得到准确性更高的结果. 这是因为 ApproxECIoT 在计算结果出现较大偏差时, 并不是申请更多的内存资源从而增大样本容量来

减小误差, 而是在已分配内存资源不变的情况下对每层样本进行调整, 所以 ApproxECIoT 并不会占用较多的内存资源. ApproxECIoT 对于资源有限的物联网系统来说, 在提高计算结果的准确性方面拥有较好的性能.

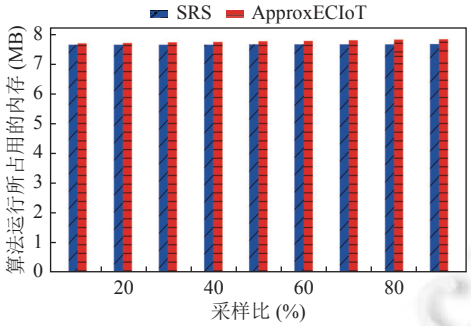


图 8 ApproxECIoT 和 SRS 对内存资源的使用情况

3.2 采用真实数据流

本节采用真实世界的数据流对 ApproxECIoT 的性能进行实验测试. 我们选择了两个真实数据集进行实验, 这两个数据集分别为 Intel Berkeleys 实验室部署的无线传感器网络所采集的气象数据集和 DEBS 2015 挑战赛公开的纽约城市出租车的运行轨迹数据集[20]. 这两个数据集均为互联网公开数据集. 考虑到这两个数据集由于网络通信质量或者数据录入失误等因素, 存在部分字段不完整数据, 所以本文对这两个数据集的噪声数据进行了处理, 具体操作是将这些噪声数据排除. 而对于一些数值异常的噪声数据不需要进行预处理, 因为本文所提出的算法可以对样本进行调整, 在调整的过程中会逐渐减小噪声数据的影响. 利用这两个真实数据集, 测试 ApproxECIoT 在实际应用中的计算结果的准确度和计算效率. 实验结果表明, 和其他处理框架相比, ApproxECIoT 在处理实际应用中的数据流时的性能具有一定的提升.

3.3 ApprxoECIoT 适用场景分析

这一节分析 ApproxECIoT 架构所适用的场景. 首先, ApproxECIoT 对样本层大小的调整是根据每层方差的大小, 采用 Neyman 最优分配计算每层的样本大小. 如果遇到非常糟糕的情况, 即某层样本的方差非常大, 在已分配有限资源的情况下, 样本层大小经过若干次调整, 将几乎全部的资源都分配给样本质量最差的一层, 而样本质量最好的层大小被调整为 1 (即层大小的最小值), 还是无法得到满足用户要求的计算结果, 那么就认为物联网数据流的 ApproxECIoT 分层采样算法并不能给出用户满意的计算结果. 在这种情况下, 所提出的 ApproxECIoT 数据流处理策略并不适用. 另外, ApproxECIoT 尽管采用 Neyman 最优分配对样本大小进行多次调整, 并重新采样. 对于数据样本极差情况, 资源并不足以用于改善样本质量, 在这种情况下不会进行多次调整, 同样也无法得出用户满意的计算结果, 此时延迟是比较小的. 对于数据样本质量可能不是特别好, 计算结果没有办法达到用户要求的情况, 可以通过多次调整样本大小并重新采样来改善样本质量. 但是, 即将到达的数据流的统计特征是无法预测的, 因此算法可能会调整多次, 从而产生较高的延迟. 所以, ApproxECIoT 也不适用于对物联网数据流计算结果的实时性有较高要求的应用场景.

ApproxECIoT 在不同的应用场景应当选择最优的精度阈值. 用户是可以设置 ApproxECIoT 的计算结果的精度要求的, 给出的精度越高, 那么样本大小的调整次数就会越多, 在最糟糕的情况下, 样本的层大小可能会调整至非常极端的情况而仍然不能满足用户给出的精度要求, 而且边缘节点的可用内存资源也是有限的, 那么此时就认为 ApproxECIoT 在给定的精度条件下无法得出计算结果. 反之, 用户给定的精度越小, 样本大小的调整次数就会越少, 那么用户任务的延时就更有可能得到满足. 在不同的物联网应用场景下, 所设定的最优精度阈值 (计算结果的置信度) 也不相同. 然而, 在大多数情况下, 用户无法得知最优的精度阈值. 因此 ApproxECIoT 架构在具体的物联网应用场景中, 可以采用机器学习方法根据历史日志数据选择最优的精度阈值, 使得计算结果能够满足用户要求的同时, 系统所产生的延时最小.

4 结 论

本文提出了物联网系统实时数据流处理框架 ApproxECIoT, 在物联网系统资源受限的情况下获取更精确的计算结果, 使用边缘节点缓存输入未发生改变的计算结果, 解决传统的物联网数据处理模式的计算效率低下、数据处理中心负载过大的问题. ApproxECIoT 考虑用户对计算结果的精度要求, 加入了误差控制策略, 当计算结果超过误差边界的时候, 根据 Neyman 分配重新调整样本大小, 然后重新采样并计算, 从而保证计算结果具有较小的精度损失. 在计算任务处理策略方面, ApproxECIoT 结合了边缘计算以及自调整计算技术, 将计算任务划分成多个子计算任务, 并将这些子计算任务分发到边缘计算节点, 使得物联网系统中边缘节点的资源也能够得到充分利用, 同时又能提高计算任务的效率. 最后, 使用模拟数据流和真实数据集对 ApproxECIoT 进行测试, 并且与 ApproxIoT 以及 SRS 进行对比, 结果证明 ApproxECIoT 在处理实时数据流方面能够在资源受限的情况下仍获得较高的准确性.

References:

[1] Zhang DG, Zhang T, Liu XH. Novel self-adaptive routing service algorithm for application in VANET. Applied Intelligence, 2019, 49(5): 1866–1879. [doi: 10.1007/s10489-018-1368-y]

[2] Zhang DG, Li G, Zheng K, Ming XC, Pan ZH. An energy-balanced routing method based on forward-aware factor for wireless sensor networks. IEEE Trans. on Industrial Informatics, 2014, 10(1): 766–773. [doi: 10.1109/TII.2013.2250910]

[3] Zhang DG, Wang X, Song XD. New medical image fusion approach with coding based on SCD in wireless sensor network. Journal of Electrical Engineering and Technology, 2015, 10(6): 2384–2392. [doi: 10.5370/JEET.2015.10.6.2384]

[4] Zhang DG, Wang X, Song XD, Zhao DX. A novel approach to mapped correlation of ID for RFID Anti-collision. IEEE Trans. on Services Computing, 2014, 7(4): 741–748. [doi: 10.1109/TSC.2014.2370642]

[5] Zhang DG, Zheng K, Zhang T, Wang X. A novel multicast routing method with minimum transmission for WSN of cloud computing service. Soft Computing, 2015, 19(7): 1817–1827. [doi: 10.1007/s00500-014-1366-x]

[6] Zhang DG, Zhang XD. Design and implementation of embedded un-interruptible power supply system (EUPSS) for Web-based mobile application. Enterprise Information Systems, 2012, 6(4): 473–489. [doi: 10.1080/17517575.2011.626872]

[7] Zhang DG. A new approach and system for attentive mobile learning based on seamless migration. Applied Intelligence, 2012, 36(1): 75–89. [doi: 10.1007/s10489-010-0245-0]

[8] Zhang DG, Zheng K, Zhao DX, Song XD, Wang X. Novel Quick Start (QS) method for optimization of TCP. Wireless Networks, 2016, 22(1): 211–222. [doi: 10.1007/s11276-015-0968-2]

[9] Zhang DG, Zhu YN, Zhao CP, Dai WB. A new constructing approach for a weighted topology of wireless sensor networks based on local-world theory for the Internet of Things (IOT). Computers & Mathematics with Applications, 2012, 64(5): 1044–1055. [doi: 10.1016/j.camwa.2012.03.023]

[10] Zhang DG, Zhang T, Zhang J, Dong Y, Zhang XD. A kind of effective data aggregating method based on compressive sensing for wireless sensor network. EURASIP Journal on Wireless Communications and Networking, 2018, 2018(1): 159. [doi: 10.1186/s13638-018-1176-4]

[11] Yen IL, Bastani F, Solanki N, Huang YT, Hwang SY. Trustworthy computing in the dynamic IoT cloud. In: Proc. of the 2018 IEEE Int'l Conf. on Information Reuse and Integration (IRI). Salt Lake City: IEEE, 2018. 411–418. [doi: 10.1109/IRI.2018.00067]

[12] López Peña MA, Muñoz Fernández I. SAT-IoT: An architectural model for a high-performance Fog/Edge/Cloud IoT platform. In: Proc. of the 5th IEEE World Forum on Internet of Things (WF-IoT). Limerick: IEEE, 2019. 633–638. [doi: 10.1109/WF-IoT.2019.8767282]

[13] Wei JY, Cao SZ. Application of edge intelligent computing in satellite Internet of Things. In: Proc. of the 2019 IEEE Int'l Conf. on Smart Internet of Things. Tianjin: IEEE, 2019. 85–91. [doi: 10.1109/SmartIoT.2019.00022]

[14] Gao MZ, Qu G. A novel approximate computing based security primitive for the Internet of Things. In: Proc. of the 2017 IEEE Int'l Symp. on Circuits and Systems (ISCAS). Baltimore: IEEE, 2017. 1–4. [doi: 10.1109/ISCAS.2017.8050360]

[15] Chen X, Shi Q, Yang L, Xu J. ThriftyEdge: Resource-efficient edge computing for intelligent IoT applications. IEEE Network, 2018, 32(1): 61–65. [doi: 10.1109/MNET.2018.1700145]

[16] Yan Y, Chen LJ, Zhang Z. Error-bounded sampling for analytics on big sparse data. Proc. of the VLDB Endowment, 2014, 7(13): 1508–1519. [doi: 10.14778/2733004.2733022]

[17] Hoeffding W. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 1963, 58(301): 13–30. [doi: 10.2307/2282952]

[18] Al-Kateb M, Lee BS. Adaptive stratified reservoir sampling over heterogeneous data streams. Information Systems, 2014, 39: 199–216. [doi: 10.1016/j.is.2012.03.005]

[19] Harper R. Self-adjusting computation. In: Proc. of the 19th Annual IEEE Symp. on Logic in Computer Science. Turku: IEEE, 2004. 254–255. [doi: 10.1109/LICS.2004.1319619]

[20] Jerzak Z, Ziekow H. The DEBS 2015 grand challenge. In: Proc. of the 9th ACM Int'l Conf. on Distributed Event-based Systems (DEBS). Oslo: ACM, 2015. 266–268. [doi: 10.1145/2675743.2772598]

[21] Ni SL, Wang YL, Zhao ZW, Dong Z. Approximate query algorithm of streaming data based on stratified sampling. Computer Engineering and Design, 2017, 38(10): 2697–2702, 2717 (in Chinese with English abstract). [doi: 10.16208/j.issn1000-7024.2017.10.019]

[22] Dong LY, Wang YQ, Li YL, Zhu Q. Adaptive random sampling algorithm based on the balance maximization. Journal of Northeastern University (Natural Science), 2018, 39(6): 792–796 (in Chinese with English abstract). [doi: 10.12068/j.issn.1005-3026.2018.06.007]

附中文参考文献:

[21] 倪赛龙, 王永利, 赵忠文, 董振. 基于分层抽样的数据流近似查询算法. 计算机工程与设计, 2017, 38(10): 2697–2702, 2717. [doi: 10.16208/j.issn1000-7024.2017.10.019]

[22] 董立岩, 王越群, 李永丽, 朱琪. 基于最大平衡度的自适应随机抽样算法. 东北大学学报(自然科学版), 2018, 39(6): 792–796. [doi: 10.12068/j.issn.1005-3026.2018.06.007]

张德干(1969－), 男, 博士, 教授, 博士生导师, 主要研究领域为边缘计算, 物联网, 无线传感器网络, 移动计算, 信息安全, 云计算.

张婷(1972－), 女, 博士, 教授, 主要研究领域为物联网, 车联网, 边缘计算.

颜浩然(1995－), 男, 硕士, 主要研究领域为物联网, 无线传感器网络, 移动计算, 边缘计算.

王嘉旭(1997－), 男, 硕士, 主要研究领域为物联网, 车联网, 边缘计算.

张捷(2000－), 男, 学士, 主要研究领域为车联网, 移动计算, 云计算.