分布式网络中连续时间周期的全局top-*K*频繁流测量

毛晨宇, 黄河, 孙玉娥, 杜扬

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000119

分布式网络中连续时间周期的全局 top-*K* 频繁流测量

毛晨宇[1]　黄　河[1]　孙玉娥[2]　杜　扬[1]
1 苏州大学计算机科学与技术学院　江苏 苏州 215006
2 苏州大学轨道交通学院　江苏 苏州 215131
(20225227129@stu.suda.edu.cn)

摘　要　在分布式网络中,测量 top-*K* 频繁流对资源分配、安全监控等应用至关重要。现有的 top-*K* 频繁流测量工作存在不适用于测量分布式网络流量或只考虑单时间周期等局限。为此,提出了分布式网络中连续时间周期的全局 top-*K* 频繁流测量方案,在分布节点中布置了紧凑的概率数据结构来记录网络流信息,每个时间周期结束后分布节点向中心节点发送必要信息,中心节点汇聚得到从测量开始至当前时间周期的全局 top-*K* 频繁流。考虑到每条流可能出现在一个或多个测量节点,使用了不同的方法来减少传输开销。对于每条流只会出现在单一节点的情况,采用传输分段最小值的方法来获得阈值,实验结果表明这种方法减少了全量传输超过 50%的传输开销。对于每条流会出现在多个节点的情况,提出了多阶段无误差处理方法和单阶段快速处理方法,分别应对不能容忍误差的场景和实际高速网络流量,相比每个时间周期都使用已有单周期方法,传输开销的实验表现降低了两个数量级。最后还提出了一种利用历史平均增值信息降低通信延迟的方法,实验结果表明该方法有效降低了限制信息的平均相对误差。

关键词:流量测量;top-*K* 频繁流;分布式网络;连续时间周期;sketch
中图分类号　TP391

Global Top-*K* Frequent Flow Measurement for Continuous Periods in Distributed Networks

MAO Chenyu[1],HUANG He[1],SUN Yu'e[2] and DU Yang[1]
1 School of Computer Science and Technology,Soochow University,Suzhou,Jiangsu 215006,China
2 School of Rail Transportation,Soochow University,Suzhou,Jiangsu 215131,China

Abstract　In distributed networks,the measurment of the top-*K* frequent flows is crucial for applications like resource allocation and security monitoring. Existing works on top-*K* frequent flow measurement have limitations such as being unsuitable for distributed network traffic measurement or only considering single time periods. To address these problems,this paper proposes a scheme for measuring global top-*K* frequent flows over continuous time periods in distributed networks. This involves deploying compact probabilistic data structures at distributed nodes to record network flow information. At the end of each time period,distributed nodes send necessary information to a central node,which aggregates this to identify the global top-*K* frequent flows from the start of measurement to the current time period. Considering that each flow may appear at one or multiple measurement nodes,different methods are used to reduce transmission overhead. For flows appearing at a single node,a method of transmitting segmented minimum values is used to obtain a threshold. Experiments show that this method reduces the transmission overhead by over 50% compared to full transmission. For flows appearing at multiple nodes,a multi-stage error-free processing method and a single-stage fast processing method are proposed,catering to scenarios that cannot tolerate errors and actual high-speed network traffic,respectively. Compared to using existing single-period methods in each time period,experimental performance of transmission overhead reduced by two orders of magnitude. Finally,a method using historical average increment information to reduce communication delay is also proposed,and experimental results show that it effectively reduces the average relative error of constraint information.

Keywords　Traffic measurement,top-*K* frequent flows,Distributed network,Continuous time period,sketch

到稿日期:2023-10-18　返修日期:2023-11-27
基金项目:国家自然科学基金(62332013,62072322,U20A20182,62202322)
This work was supported by the National Natural Science Foundation of China(62332013,62072322,U20A20182,62202322).
通信作者:孙玉娥(sunye12@suda.edu.cn)

1 引言

随着互联网技术的发展，光纤网络和 5G 等新技术得到广泛应用，网络传输速度达到了前所未有的高峰，同时分布式网络被广泛应用于云计算、物联网、人工智能、区块链等领域，为用户提供高效、灵活、可扩展的服务[1]。然而，极快的网络传输速度和分布式网络的复杂性给网络管理带来了新的问题，在这一背景下，网络流量测量工作变得越来越重要。

网络流量测量的目标是获取网络流的统计信息，网络流指流标签相同的所有数据包，其中流标签通常被定义为数据包头部的某些字段或组合，如源 IP 地址、目的 IP 地址、源端口、目标端口和协议等，一条流包含的数据包数量即为该条流的频数。在实际网络中，流的频数分布呈现高度倾斜，即绝大多数流都是小频数流，而只有极少数流的频数非常大[2]。例如，根据对 CAIDA2019 数据集[3]的统计，频数大小排名前 2%的流占据了 80%以上的总流量，而排名后 80%的流频数都不超过 10。这表明频数按大小排序靠前的 K 条流，即 top-K 频繁流，承载了网络中的关键信息。

分布式网络中的 top-K 频繁流又称为全局 top-K 频繁流，通常分为两种情况。第一种是节点流无交集的情况，即每个流标签仅由一个节点监测，如在广域网自治系统的不同边界路由器上，以源地址为流标签，每个节点记录的流彼此之间没有交集[4]，此种情况直接比较所有节点记录的频数，最大的 K 个频数对应的流即为全局 top-K 频繁流；第二种情况是节点流有交集的情况，即每个流标签会被多个节点监测，如在内容分发网络和数据中心，以请求内容为流标签，不同地理位置的服务器上记录的流存在交集，在这种情况下需要比较每条流在所有节点上的累积频数，累积频数最大的 K 条流为全局 top-K 频繁流。全局 top-K 频繁流可以揭示网络的热点信息，但仅有热点信息对网络相关应用的帮助有限，为了监测网络中潜在的异常行为和安全威胁[5]以及更好地规划和管理分布式网络资源[6]，还需要获得网络热点信息的变化趋势，而这就需要测量连续时间周期的全局 top-K 频繁流。

目前的 top-K 频繁流测量工作通常采用 sketch 等紧凑的概率数据结构来记录流信息，然后排序或使用最小堆得到 top-K 频繁流，但这些工作主要集中在单节点测量的场景[7-8]。而分布式环境中流量测量的工作目前相对较少，它们大多采用编码或摘要技术压缩 sketch 后将其发送到中心节点进行处理[9-12]，但由于压缩后无法直接准确还原流标签，不适用于测量全局 top-K 频繁流。分布式环境下专注于全局 top-K 频繁流测量的研究更为稀少。文献[4]研究了节点流无交集情况下的 top-K 公平性问题，但未考虑传输开销；文献[13-14]研究了节点流有交集情况下获取全 top-K 频繁流的最小传输开销方案，但是仅适用于单周期测量，无法扩展到连续时间周期下的全局 top-K 频繁流测量。

分布式网络中测量连续时间周期的全局 top-K 频繁流面临的挑战可以总结为如下 4 点：1）高速网络处理芯片存储资源有限，不足以支持海量网络流量的详细记录；2）流量信息分散在多个节点，需要多节点协同测量，相比单节点测量更为复杂，同时协同测量过程中通信会占用大量的带宽资源；3）通信延迟可能导致测量结果与目标测量时间段不匹配，这点在网络拥塞和单点故障时更为严重；4）存在单节点频数不高但合并后频数足够大成为全局 top-K 频繁流的“全局冰山流”[15]，这类流难以检测。目前还没有完全应对这 4 个挑战的相关工作。

鉴于上述背景和挑战，本文致力于分布式网络中全局 top-K 的测量工作，解决了现有工作未考虑到的连续时间周期全局 top-K 测量问题，全面考虑了节点流量无交集和有交集两种情况，并分别设计了测量方案。具体地，每个分布节点上布置 sketch 结构，在极小的空间内记录流信息；数据包到来后，在记录的同时简单处理得到需要发送给中心节点的必要信息，不需要重复处理流信息以满足连续时间周期的实时测量需求；使用布隆过滤器[16]表示限制信息，同时积极利用旧时间周期的信息来缩小传输列表，以此控制传输开销；适当放宽限制条件，以检测强隐匿的“全局冰山流”；使用历史平均增值预测新时间周期阈值对削弱通信延迟的影响。最后在真实网络流量数据集 CAIDA2019 上进行仿真模拟，验证了我们的测量方案有很好的准确度表现，设计的传输机制可以极大地节省传输开销，同时应对通信延迟方法得到的预测阈值与真实阈值的平均相对误差也减小了。

本文第 2 章介绍了 top-K 频繁流测量与分布式流量测量的相关工作；第 3 章给出了网络模型以及问题的定义和设计目标；第 4 章详细介绍了两种情况下的测量方法和传输机制；第 5 章对使用的方法进行了仿真模拟验证；最后总结全文并展望未来。

2 相关工作

网络流量测量领域中单节点环境下的 top-K 频繁流挖掘已经有广泛的研究，尤其是 sketch 相关的技术方法。然而，在分布式网络中进行多节点测量并致力于挖掘全局 top-K 频繁流的工作较为有限。本章将介绍一些典型的工作，包括单节点的 top-K 频繁流测量、多节点流量测量以及多节点全局 top-K 频繁流测量的研究。

在单节点环境下，已有许多工作研究了 top-K 频繁流测量。例如，Unbiased Space Saving[17]于 Space Saving 算法，引入了随机化，并且可以对数据流每个频繁流提供无偏的估计计数，得到估计后对频繁流排序从而获得 top-K 频繁流。WavingSketch[18]利用一种称为 Waving Counter 的新的数据结构来提供频数的无偏估计，并将频繁流记录在 heavy part 中，当一个新元素到来时，使用 waving counter 估计它的频数，如果频数较高就代替 heavy part 中频数最低的项目，通过这种机制，WavingSketch 能够高效并准确地维护数据流中 top-K 频繁流集合。HeavyKeeper[7]通过指数衰减的策略移除大量小流，只关注大流，在记录大流的同时维护一个最小堆来得到 top-K 频繁流，并且在小内存的情况下表现效果更佳。在分布式网络中，由于有全局冰山流的存在，我们不能只关注频数足够大的流，因此这些工作只适用于单节点测量的情况，但是对我们分布式网络中多节点的测量工作有很大的启发作用。

关注分布式测量情景的工作较少，现有的工作通常是

本地使用较大的空间，尽可能记录详细的信息，然后在传输时将信息压缩，以此降低传输开销。SF-Sketch[9]使用一个大容量的 Fat-subsketch 在本地节点进行频繁的更新操作，并定期生成一个小容量的 Slim-subsketch 发送到中心节点，这种设计符合分布式环境下的带宽和存储限制，但是只能满足查询流频数的需求，无法得到全局 top-K 频繁流。Dai 等研究了从分布式数据集中找出在多个数据集中频数都超过某个阈值的项目[10]，提出了 DISPERS 算法来解决这个问题，其基本思想是每个节点用 Raptor 编码对数据集中的每个项目 ID 进行压缩，中心节点汇聚后通过再次通信解码项目 ID。Li 等研究了从多个分布式数据流中找出全局频数大于某个阈值的流的问题[11]，使用 L-CCF 和 H-CCF 方法分别处理低阈值和高阈值的情况，也使用了 Raptor 编码压缩流标签。

在分布式情况下进行全局 top-K 挖掘的工作更为稀缺。Zhao 等讨论了节点流无交集情况下的全局 top-K 问题[4]，使用的 Double-Anonymous Sketch 可以做到对流频数的无偏估计，以消除节点数据量不平衡造成的不公平估计误差，但是它将所有数据发往中心节点进行汇聚，并没有考虑获取全局 top-K 时的传输开销问题。TPUT[13]讨论了节点流有交集情况下准确高效地获取全局 top-K 的方法，它使用了一种固定三阶段的方法，先估计下界，然后剪枝不可能进入全局 top-K 集合的元素，最后查找剩余对象的准确值，这样可以得到准确的全局 top-K 频繁流，但它基于已经提前获取准确的数据流信息，并把它们排序的强假设。KLEE[14]提出了一个框架，用于解决最小化网络通信开销和响应时间同时生成高质量全局 top-K 结果的问题。它使用直方图和布隆过滤器结合的方法来大致表示流的分布，通过部分流分布信息得到阈值，每个分布节点得到大于阈值的候选者集合频数信息并发送给中心节点，中心节点再根据一定规则缩小候选者集合，最后分布节点将缩小的候选者集合发送给中心节点，中心节点汇聚得到全局 top-K。它引入了可接受的误差，在 TPUT 的基础上大幅缩减了传输开销，但是也需要对数据进行更复杂的处理以得到流分布的情况。并且 TPUT 和 KLEE 只考虑了一个时间周期的情况，这两种方法均不适用于多个连续时间周期。

综上，目前还没有在分布式网络环境中对连续时间周期全局 top-K 频繁流测量问题的研究。现有相关工作存在只考虑单节点情形、无法在分布式环境中准确还原流标签、计算和通信成本过高、需要完整流数据分布支持以及仅适用于单周期静态流数据等局限，这些限制使得它们难以直接应用于真实的大规模动态分布式网络。

3 问题模型

3.1 网络模型

本文考虑的网络模型如图 1 所示，它主要由 1 个中心节点和 N 个分布节点组成。在这种网络中，中心节点负责整个系统的协调和管理，维护分布节点的状态，分配任务给分布节点。分布节点之间无直接通信，所有交互均通过中心节点进行，每个分布节点负责一定范围的网络，它们接收中心节点分配的任务，独立地执行处理并返回中心节点结果。其中分布节点直接处理网络流量，需要使用高速网络芯片，而中心节点没有直接处理高速流量的任务，可以使用存储资源更丰富的片下空间。

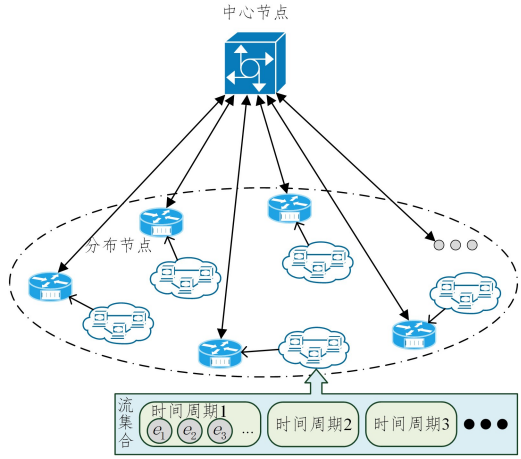


图 1 分布式网络模型

Fig. 1 Distributed network model

每个分布节点负责的网络中有不同的流量信息，所有网络中的流数据集合表示为 $S=\{S_1 S_2 S_3,\cdots,S_n\}$。每个分布节点处理的流数据集合都可以根据时间周期划分为 T 个子集，即 $S_i=\{D_{(i,1)},D_{(i,2)},D_{(i,3)},\cdots,D_{(i,T)}\}$。第 i 个节点的第 j 个时间周期的流数据 $D_{(i,j)}$ 包含 $m_{(i,j)}$ 个数据包，$D_{(i,j)}=\{e_1,e_2,e_3,\cdots,e_{m_{(i,j)}}\}$，其中每个 e 代表一个数据包的流标签，流标签可以是数据包头字段（源 IP 地址，目的 IP 地址，源端口，目的端口，协议类型）或它们的组合。第 i 个节点的流标签集合为 $U_i=\{u_{(i,1)},u_{(i,2)},u_{(i,3)},\cdots,u_{(i,n_i)}\}$，共 n_i 种不同的流标签。根据流标签的定义，同一种流标签可能只会出现在一个分布节点，也可能出现在多个分布节点，即任意两个分布节点 i 和 j 记录的流可能有交集（$U_i\cap U_j\neq\varnothing$），也可能没有交集（$U_i\cap U_j=\varnothing$）。

将一个时间周期内相同流标签的数据包数量定义为这个流标签的频数，若干个时间周期内所有时间周期的频数之和定义为这个流标签在这段时间内的累积频率。不同节点流无交集时，任意一个流标签 $u_{(i,h)}$ 从开始到第 j 个时间周期的全局累积频数为：

$$f_{(u_{(i,h)},j)}=\sum_{t=1}^{j}\sum_{k=1}^{m_{(i,t)}}\mathbf{1}_{\{e_k=u_{(i,h)},e_k\in D_{(i,t)}\}} \tag{1}$$

不同节点流有交集时，任意一个流标签的全局累积频数为在所有分布节点上累积频数之和，流标签 $u_{(i,h)}$ 从开始到第 j 个时间周期的全局累积频数为：

$$f_{(u_{(i,h)},j)}=\sum_{g=1}^{n}\sum_{t=1}^{j}\sum_{k=1}^{m_{(i,t)}}\mathbf{1}_{\{e_k=u_{(i,h)},e_k\in D_{(g,t)}\}} \tag{2}$$

从测量开始到第 j 个时间周期全局累积频数最大的 K 个流标签集合即为全局 top-K 频繁流，将其表示为 F_j，在整个测量过程中每个时间周期结束时全局 top-K 频繁流为集合 $\Psi=\{F_1,F_2,F_3,\cdots,F_T\}$。

3.2 设计目标

本文致力于提出一种高效挖掘分布式网络流量信息中全局 top-K 频繁流的测量方案。该方案需要连续若干个周期在每个网络分布节点中记录流信息，然后将必要信息传输到中心节点，中心节点汇聚得到全局 top-K 频繁流。该方案需要

同时考虑到网络流量转发高速、可使用的片上空间紧缺、可占用的带宽资源有限等限制条件，因此该方案应该满足以下4个设计目标。

1）分布节点记录流信息空间开销小：在网络设备上高速存储空间资源紧缺的限制下，应该使用紧凑的数据结构来压缩记录流信息。

2）传输的冗余信息尽可能少：测量任务不应该影响网络的状态，分布节点与中心节点之间每个时间周期都会有通信，如果通信内容过大，则会占用带宽资源，影响网络性能。

3）分布节点对数据包的处理简单：为了匹配数据包在网络中的传输速度，分布节点对数据包的处理不应该对数据包正常转发速度产生明显影响，因此对数据包不可以进行过于复杂的操作。

4）有效识别强隐匿 top-K 频繁流：为了保证得到的 top-K 频繁流集合的准确性，必须有效识别具有强隐匿性的全局冰山流。

4 全局 top-K 频繁流挖掘机制

为实现分布式网络中连续周期的全局 top-K 频繁流挖掘，本文针对节点流有交集和节点流无交集的情况分别设计了一个高效的解决方案。本章先分别介绍了节点流有交集和节点流无交集情况下的解决方案，最后介绍了一种应对通信延迟的方法。

4.1 机制概述

节点流有交集和无交集两种情况下解决方案的关键思路相同：随着数据包的到来，每个分布节点记录流频数信息，同时进行一些不影响数据包正常传输的简单处理。当一个时间周期结束时，根据之前收到的限制信息快速得到需要传输给中心节点的必要信息；中心节点在收到每个分布节点发送的必要信息后，汇聚得到下一个时间周期的限制信息，然后将其传输给每个分布节点。该思路满足了传输冗余信息尽可能少、分布节点对数据包处理简单的设计目标，不同方法通过各自的机制如使用高效的概率数据结构 sketch（如 CM[19]，CU[20]）压缩记录流信息、使用布隆过滤器表示限制信息等设计，满足了其他设计目标，在后文每种方法的介绍中将具体说明。

4.2 应对节点流无交集情况的解决方案

在节点流无交集的情况下，一个流标签只会出现在一个分布节点中，全局 top-K 频繁流的频数只与一个分布节点有关，因此中心节点无须将不同分布节点的流信息汇聚，只需要简单地比较所有分布节点中流标签的累积频数，找到累积频数最大的前 K 个流标签即可。并且每个节点记录的所有流标签后一个时间周期的累积频数仅与自身前一时间周期记录的频数有关，不需要对其进行特别处理。并且每个时间周期分布节点记录的流标签累积频数信息仅依赖于自身前一个周期的信息，不需要与中心节点交换信息即可得到全局累积频数。

在应对节点流无交集情况时，我们在每个分布节点中使用哈希表记录流信息，使用最小堆维护每个节点处的 top-K 二元组，然后通过 s 分传输机制减少中心节点汇聚全局 top-K 的通信开销。下文介绍分布节点和中心节点在每个时间周期内的处理过程。

4.2.1 分布节点数据结构及处理过程

分布节点使用 CM，CU 等可以估计流频数的 sketch 来记录流信息，同时维护一个大小为 K 的最小堆来记录频数最大的前 K 条流。

当每个数据包到来时，首先将其流标签插入 sketch 中，在插入之后会得到这个流标签当前的频数估计，根据得到的频数估计结果维护最小堆。在每个时间周期结束后，最小堆中即为这个分布节点到这个时间周期为止的累积频数 top-K 频繁流。每个分布节点再将各自 top-K 频繁流的频数排序之后 s 分，即：

$$s=\sqrt{\frac{K*(n-1)*(x+y)}{x*n}} \tag{3}$$

其中，x 为传输一个整数的比特数，y 为传输一个流标签的比特数。将 s 分后每段末尾的频数发送给中心节点，然后等待中心节点发回阈值，收到阈值后将大于等于阈值的所有流标签及其频数发给中心节点。图 2 中，$K=7$，$s=4$，某个分布节点先使用最小堆在记录流信息的同时得到 top-7 流，对 top-7 流排序后 4 分，得到每个分割段的频数最小值，即 109，78，31，16，然后将它们发送给中心节点，之后收到了中心节点发来的阈值 80，再将频数大于 80 的流标签及其频数发往中心节点。

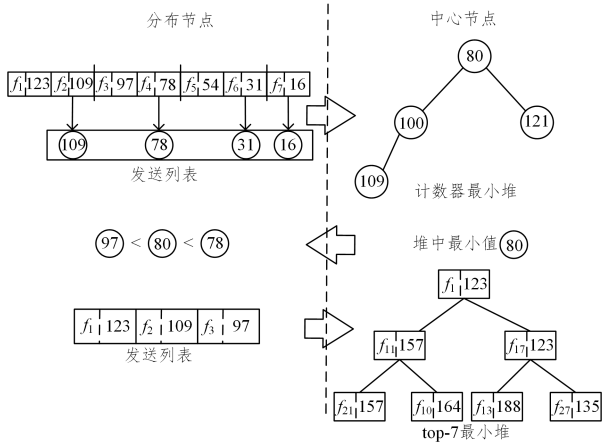


图 2 节点流无交集情况下解决方案的示例

Fig. 2 Example of a solution for node flows that do not intersect

在此简述式(3)s 取值的理由：所有节点发送它们 s 分最小频数的开销为 $s*n*x$，中心节点下发阈值的开销为 $n*x$，每个分布节点返回大于阈值的频数，除了返回目标 k 个二元组外，最多会有 $\frac{k}{s}*(n-1)$ 个多传的二元组，这部分开销共为 $\left(k+\frac{k}{s}*(n-1)\right)*(x+y)$，中心节点汇聚后得到全局 top-$K$ 阈值，相加后得到传输开销的上界为：

$$(s+1)*n*x+\left(k+\frac{k}{s}*(n-1)\right)*(x+y) \tag{4}$$

根据式(4)可解得传输开销上界最小时 s 的值为式(3)，此时在最坏的情况下传输开销最优。

该方案分布节点和中心节点的具体算法如算法 1 所示。

算法 1 节点流无交集情况下分布节点处理算法

输入：一个时间周期的流数据 D，最小堆 H，分段数目 s

```
输出:无输出
1. for each packet e in D:
2. /* 插入流标签得到估计频数并维护最小堆 */
3.    f=InsertLabel(e.label);
4.    if f>H.TopFrequency():
5.       UpdateHeap(H,f,e.label);
6. /* 排序后 s 分,发送每段最小频数 */
7. list=SplitS(H.sort());
8. SendToCentral(list);
9. /* 阻塞等待接收阈值后发送候选者列表 */
10. threshold=WaitForThreshold();
11. for each item in H:
12.    if item.frequency > threshold:
13.          candidate.insert(item);
14. SendCandidateList(candidate).
```

4.2.2 中心节点数据结构及处理过程

中心节点的数据结构包括一张哈希表和一个最小堆,哈希表存储流信息,最小堆维护全局 top-K 频繁流。

中心节点收到每个分布节点由式(3) s 分后每段的最小频数后,在所有频数排序后将第 i 位作为阈值发送给所有分布节点,然后等待返回候选者二元组列表,收到二元组列表后维护最小堆得到全局 top-K 频繁流。

$$i=s \tag{5}$$

图 2 中,中心节点使用最小堆得到第 $i=4$ 位的频数作为阈值下发给所有分布节点,之后收到所有分布节点发送的候选者二元组列表,重新维护一个大小为 7 的最小堆,得到全局 top-7 频繁流。具体算法如算法 2 所示。

算法 2 节点流无交集情况下中心节点处理算法

```
输入:最小堆 H
输出:全局 top-K 频繁流集合 S
1. L=WaitForSplitFrequency();
2. /* 排序后式(5)所示位置作为阈值下发 */
3. Sort(L);
4. threshold=L[i];
5. SendToDistributed(threshold);
6. /* 阻塞接收候选者列表后维护最小堆 */
7. candidateList =WaitForCandidate();
8. for each item in candidateList:
9.    if item.frequency > H.TopFrequency():
10.          UpdateHeap(H,f,e.label);
11. S=H.
```

4.2.3 时间复杂度分析

本文按照算法流程来分析一个时间周期内每步的时间复杂度。首先,分布节点在接收每个数据包的同时维护一个最小堆,处理单个数据包的时间复杂度为 $O(\log K)$,一个时间周期内共 N 个数据包,那么该操作总的时间复杂度为 $O(N\log K)$,在一个时间周期的数据包处理完后,对最小堆排序得到 s 分位置的值,其时间复杂度为 $O(K\log K)$,中心节点维护最小堆并找到第 i 位作为阈值的时间复杂度为 $O(N\log K)$,之后分布节点找到大于阈值的二元组时间复杂度为 $O(N)$,返回给中心节点后再次维护最小堆的时间复杂度为 $O(N\log K)$。综上,该方法在数据包到来时处理每个数据包的时间复杂度为 $(\log K)$,一个周期内所有操作整体的时间复杂度为 $O(N\log K)$。

4.3 应对节点流有交集情况的解决方案

在节点流有交集的情况下,全局 top-K 频繁流成员的全局累积频数与多个节点有关,受限于处理速度和传输开销,中心节点不可能收集每个节点所有流标签的信息,因此中心节点首先需要得到一个全局 top-K 频繁流候选列表,然后向每个分布节点请求候选列表中成员的频数,汇聚后比较所有候选成员,最后得到全局 top-K 频繁流。处理多个连续时间周期时,可以利用前一时间周期的信息得到粗略的限制信息,以此简化处理过程,节省传输开销。

本节介绍了多阶段无误差处理和单阶段快速处理两种解决方案。多阶段无误差处理方法多次传输数据以获得更精确的结果,单阶段快速处理方法合并省略多阶段方法的某些阶段,使用 Sketch 记录流信息,布隆过滤器表示限制信息,牺牲较小程度的准确率,提升了处理速度并且大幅减少了总传输开销,因此单阶段快速处理的解决方案适用于高速网络流量环境。

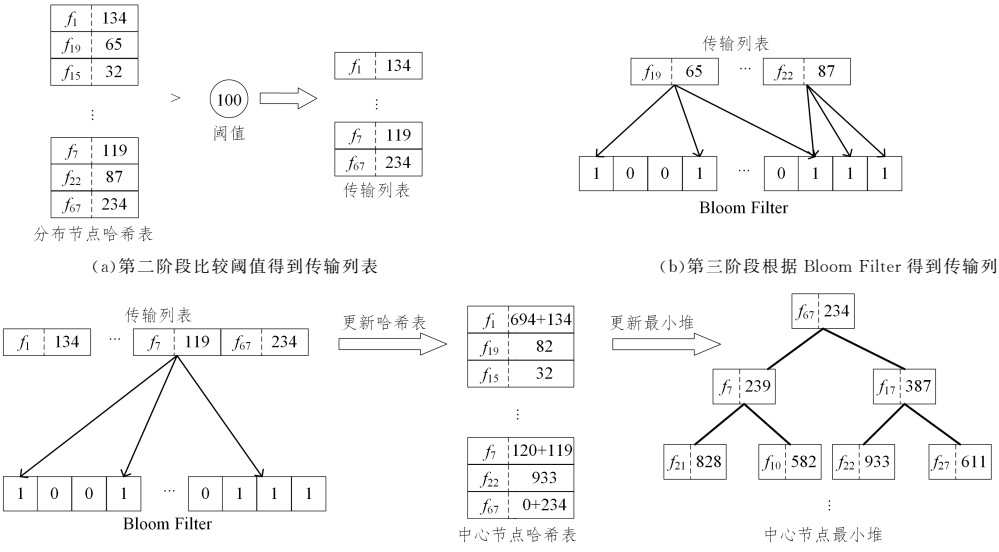
4.3.1 多阶段无误差处理方法

多阶段无误差处理方法是对 TPUT 进行改进,仍然使用和 TPUT 类似的多阶段处理方法,但是其可以用于连续多个时间周期,并且引入了布隆过滤器来表示候选者集合,以此削减传输开销。

在这个解决方案中,分布节点和中心节点都需要使用哈希表来记录流信息,使用最小堆来记录 top-K 信息,分布节点的最小堆只在第一个时间周期使用。

数据流不断到达分布节点,分布节点将其插入哈希表并得到该流的频数估计,使用频数估计来维护最小堆。第一阶段,每个分布节点将最小堆中的流标签及其频数发送给中心节点,中心节点收到信息后维护哈希表和最小堆,所有信息处理完毕后,将最小堆中的流标签插入到布隆过滤器,堆顶频数为 $value$,将 $\frac{value}{n}$ 作为阈值发送给所有分布节点;第二阶段,分布节点收到阈值后返回频数大于阈值但是第一阶段没有传输过的流标签及其频数,中心节点收到信息后维护哈希表和最小堆,同时把这些流标签都插入布隆过滤器中,处理完毕后将布隆过滤器发送给分布节点;第三阶段,每个分布节点再次检查每个没传输过的流标签是否在布隆过滤器中,如果在布隆过滤器中,则将流标签和频数发送给中心节点,中心节点汇聚所有信息维护最小堆和哈希表,此时最小堆中的元素就是第一个周期的全局 top-K 频繁流。在第一个时间周期之后的每个周期,我们可以根据前一个时间周期的结果得到这个时间周期较宽松的限制条件,由此除第一个时间周期外,其他时间周期只需要执行后两个阶段即可。图 3 给出了一个多阶段无误差处理的简单示例,图 3(a)展示了第二阶段分布节点的处理方法,其中 $f_1,\cdots,f_7,f_{67}$ 的频数大于阈值,因此加入了传输列表发送给中心节点,图 3(c)给出了第二阶段中心节点的处理方法,将收到的传输列表插入空白 Bloom Filter 中,然后更新哈希表,根据与之前记录频数相加的结果判断 f_1,f_7,f_{67} 进入了最小堆,堆顶频数 234 的 $\frac{1}{n}$ 则为下一时间周期的

阈值，图 3(b)给出了第三阶段分布节点判断流标签是否在 Bloom Filter 中的方法，f_{19}，…，f_{22} 在 Bloom Filter 中的对应位都为 1，因此它们加入了传输列表。分布节点和中心节点具体的处理过程如算法 3 和算法 4 所示。



(a)第二阶段比较阈值得到传输列表

(b)第三阶段根据 Bloom Filter 得到传输列表

(c)第二阶段更新哈希最小堆以及 Bloom Filter

图 3 多阶段无误差处理方法的示例

Fig. 3 Example of multi-stage error-free processing method

算法 3 多阶段无误差解决方案分布节点处理算法

输入：流数据集合 D，最小堆 H，哈希表 T

输出：无输出

```
1. /* 第一阶段 */
2. for each packet e in D[0]:
3.    if not first period:
4.       threshold=WaitForThreshold();
5.       break;
6.    InsertAndUpdateHeap(T,H,e);
7. TransToCentral(H);
8. threshold=WaitForThreshold();
9. /* 第二阶段 */
10. for each packet e in D[0]:
11.    if e not be sent and e.frequency > threshold:
12.    sendList.append(e);
13. SendToCentral(sendList);
14. blm=WaitForBlm();
15. /* 第三阶段 */
16. for each packet e in D[0]:
17.    if e not be sent and e.label in blm:
18.       sendList.append(e);
19. SendToCentral(sendList).
```

算法 4 多阶段无误差解决方案中心节点处理算法

输入：最小堆 H，哈希表 T，分布节点数量 n

输出：全局 top-K 频繁流集合 S

```
1. /* 第一阶段 */
2. if not first period:goto 8;
3. L =WaitForTopkList();
4. for each item in L:
5.    InsertAndUpdateHeap(T,H,item);
6. threshold=H.TopFrequency()/n;
7. SendToDistributed(threshold);
8. /* 第二阶段 */
9. L=WaitForList();
10. blm=BloomFilter();
11. for each item in L:
12.    InsertAndUpdateHeap(T,H,item);
13.    blm.insert(item);
14. SendToDistributed(blm);
15. /* 第三阶段 */
16. L=WaitForList();
17. for each item in L:
18.    InsertAndUpdateHeap(T,H,item);
19. threshold=H.TopFrequency()/n;
20. SendToDistributed(threshold).
```

4.3.2 单阶段快速处理方法

多阶段的处理方法可以保证无误差地找到全局 top-K 频繁流，但是它空间占用极大并且在实际网络环境中并没有时间进行多阶段的处理。我们对多阶段的处理方案进行了简化，使之适用于高速网络环境的代价只是牺牲一点可接受的准确度。

在这个解决方案中，分布节点上的数据结构包括一个 CM，CU 或者 CMM 等可以测量频数的 sketch，以及一个只在第一个时间周期用到的 k-最小堆；中心节点上的数据结构包括一张哈希表和一个 k-最小堆。

每个分布节点将流插入 Sketch 并得到这条流的频数估计。在第一个时间周期，我们依然需要维护一个最小堆，每个节点得到各自的 top-K 频繁流并发送给中心节点，中心节点维护哈希表和最小堆，在维护最小堆的同时将每个大于当时堆顶 value 的 $\frac{1}{n}$ 的流标签插入到空布隆过滤器中，将汇聚完成后的最小堆作为第一个时间周期的全局 top-K。这个结果的准确度并不高，为了使之适应于网络环境，这是我们必须

进行的取舍，后续周期会达到较高的准确度。后续每个时间周期我们将前一个时间周期得到的最小堆堆顶 value 的$\frac{1}{n}$作为阈值，将布隆过滤器作为候选者集合的摘要传给分布节点，每个分布节点在动态地处理流信息插入 sketch 的同时加入判断，如果大于上一周期的阈值或者在布隆过滤器中，则加入传输列表，在时间周期结束后，发送给中心节点，中心节点汇聚收到的每个分布节点的传输集合，维护最小堆，以及将大于堆顶 value 的$\frac{1}{n}$的元素插入新的布隆过滤器中，并将其作为下一个周期的阈值和候选者集合摘要，全部处理完成后得到的最小堆就是全局 top-K 频繁流。图 4 中，分布节点使用 Count Min Sketch 记录流频数，流 f_1 和 f_2 到达分布节点，插入 CM 后得到各自频数的估计值 34，108，f_1 可以在 Bloom Filter 中得到，f_2 频数估计值大于阈值，因此它们都加入了发送列表。在这个时间周期结束后发送到中心节点，中心节点维护哈希表和最小堆得到下个时间周期的阈值和 Bloom Filter 并返回给每个分布节点，每个时间周期都是如此。分布节点和中心节点的具体算法如算法 5、算法 6 所示。

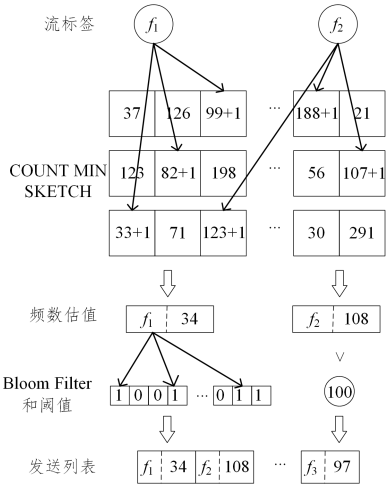


图 4 单阶段快速处理方法示例

Fig. 4 Example of single-stage rapid processing method

算法 5 单阶段快速解决方案分布节点处理算法

输入：流数据集合 D，最小堆 H，Sketch 结构 S

输出：无输出

```
1. if first period:
2.     for each packet e in D:
3.         f=S.insert(e);
4.         if f > H.TopFrequency():
5.             UpdateHeap(H,f,e.label);
6.     SendToCentral(H);
7. if not first period:
8.     for each packet e in D:
9.         f=S.insert(e);
10.        if f > threshold or e.label in blm:
11.            sendList.append(e);
12.    SendToCentral(set(sendList));
13. threshold=WaitForThreshold();
14. blm =WaitForBloomFilter().
```

算法 6 单阶段快速解决方案中心节点处理算法

输入：最小堆 H，哈希表 T，分布节点数量 n

输出：全局 top-K 频繁流集合 S

```
1. L=WaitForList();
2. blm=BloomFilter();
3. for each item in L:
4.    f=T.InsertLabel(item);
5.       if f > H.TopFrequency():
6.          UpdateHeap(H,f,item.label);
7.    if f >H.TopFrequency()/n:
8.       blm.insert(item.label);
9. threshold=H.TopFrequency()/n;
10. SendToDistributed(threshold);
11. SendToDistributed(blm).
```

4.3.3 时间复杂度分析

本小节将分别分析多阶段无误差处理方法和单阶段快速处理方法的时间复杂度。

在多阶段无误差处理方法中，第一阶段分布节点对每个数据包维护哈希表和最小堆的时间复杂度为 $O(\log K)$，若一个时间周期共有 N 个数据包，则该操作总共的时间复杂度为 $O(N\log K)$，中心节点收到所有数据包后维护最小堆的时间复杂度为 $O(N\log K)$；第二阶段分布节点返回频数大于阈值但没有传输过的二元组时间复杂度为 $O(N)$，中心节点将收到的流标签插入布隆过滤器的时间复杂度为 $O(N)$；第三阶段分布节点检查流标签是否在哈希表中的时间复杂度为 $O(N)$，中心节点汇聚信息后更新哈希表和最小堆的时间复杂度为 $O(N\log K)$。因此，在数据包到达时对每个数据包的处理时间复杂度为 $O(\log K)$，一个时间周期内所有操作的时间复杂度为 $O(N\log K)$。

在单阶段快速处理方法中，分布节点对每个数据包的操作除第一个时间周期需要维护最小堆时间复杂度为 $O(\log K)$外，后续时间周期对每个数据包处理的时间复杂度都为 $O(1)$，若一个时间周期有 N 个数据包，则分布节点总的时间复杂度为 $O(N)$，中心节点需要维护最小堆，时间复杂度为 $O(N\log K)$。综上，数据包到达时对每个数据包的处理时间复杂度为 $O(1)$，一个时间周期内所有操作的时间复杂度为 $O(N\log K)$。注意到时间复杂度较高的操作发生在中心节点，分布节点对数据包实时操作的时间复杂度仅为 $O(1)$，因此该方法适用于处理真实的高速网络实时流量。

4.4 应对传输延迟

在节点流无交集的情况下，每个分布节点测量的限制信息都在本地，在一个时间周期结束后把必要信息发送给中心节点，之后可以继续处理而不需要等待中心节点返回必要信息，因此传输延迟对这种情况没有影响。而在节点流有交集的情况下，在上一个时间周期结束后，分布节点传输完传输列表后进入下一个时间周期，继续更新流信息，但是中心节点处理完分布节点发来的信息并发送新时间周期的限制信息还需要一段时间。在这一段还未收到限制信息的时间里，分布节点不可能停止处理网络流量，在网络状况较差或者某个节点出现故障时，限制信息会在很长的时间内得不到更新。一种简单的方法是，使用上一个周期收到的限制信息，但是这样

对流的过滤作用会小很多，在网络拥塞严重或者发生单点故障时会产生很多额外的传输开销。一种更好的方法是继续使用上一个时间周期的布隆过滤器，但是阈值用以往时间周期变化值的均值来更新。

5 实验仿真

本章对节点流无交集和节点流有交集情况下的全局 top-K 频繁流解决方案进行了仿真模拟分析。首先介绍了实验平台和使用的数据集，然后介绍了评估指标，最后给出了实验结果并对其进行了详细分析。

5.1 模拟数据及仿真环境

模拟实验由 python 语言编写，在 Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz 3.60GHz，16GB 内存，64 位 Windows 10 系统上运行。

数据集使用了 2019 年 CAIDA 在 Equinix NYC 监视器上监听的 60min 时长的匿名流量数据，每分钟的数据集包含 27 万余条流，超过 1×10^7 个数据包。在该数据集中，每个数据项为一个数据包的源 IP 地址、目的 IP 地址、源端口、目的端口、协议信息五元组。我们将数据进行处理，只使用源 IP 地址作为流标签。我们将 60 min 的数据集平分给每个节点的每个时间周期，需要注意的是，并不是所有实验都使用了完整的 60 min 数据，具体情况在每部分实验会有进一步说明。CAIDA 数据集为现实流量数据集，在实验中可以很好地模拟真实网络流量的大小及分布，本文算法中没有依赖于特殊硬件的操作，在普通商用交换机和可编程交换机中都支持这些操作，因此使用 Python 处理 CAIDA 数据集可以较好地模拟真实流量测量的场景。

5.2 性能评估指标

本文使用传输开销、准确度、平均相对误差(ARE)作为实验性能指标。传输开销是评价不同传输机制优劣的重要指标，传输开销越小越符合传输冗余信息尽可能少的设计目标。准确度是衡量是否准确检测到 top-K 频繁流的关键指标，只有在高准确度下才能有效识别强隐匿 top-K 频繁流，以满足第四条设计目标。平均相对误差可以表示测量值与实际值的差异，平均相对误差越小说明测量值与实际值越接近。召回率和 F1 得分也可以用于评价集合比较，但是对于 top-K 的测量工作，由于 top-K 真实数量和测得的数量都为 K，召回率和准确度在数值上是相等的，F1 得分作为两者的调和平均值在数值上也是相等的，因此不再使用召回率和 F1 得分来评价实验结果。各性能指标的详细计算方式如下：

1)传输开销：每个分布节点发送给中心节点以及中心节点发送给每个分布节点的字节数总和。

2)准确度：$precision=\frac{FN}{K}$。其中 FN 为得到的全局 top-K 频繁流集合与真实 top-K 频繁流集合交集的元素数量，K 为 top-K 的 K 值。

3)平均相对误差：$ARE=\frac{1}{K}\sum_{f_i\in\Psi}\frac{|e_{fi}-\hat{e}_{fi}|}{e_{fi}}$。其中，$\Psi$ 是测得的全局 top-K 频繁流集合，K 为 top-K 的 K 值。

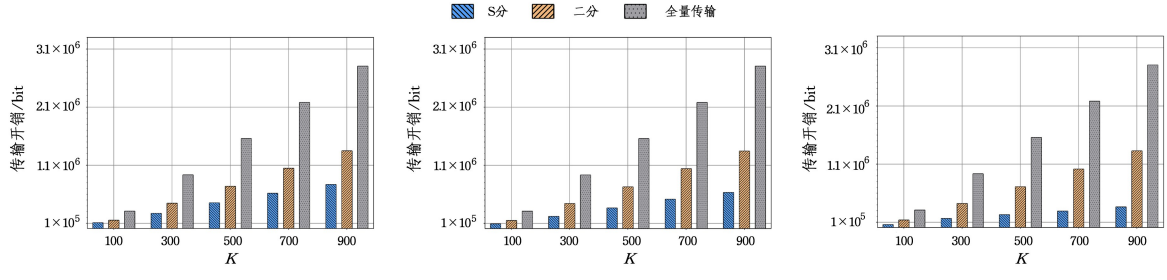
5.3 仿真模拟

本文先比较了节点流无交集情况下 s 分、二分和全量传输的传输开销；然后比较了节点流有交集情况下多阶段无误差处理方法、单阶段快速处理方法、TPUT 以及 KLEE 的传输开销，对比了单阶段快速处理方法和 KLEE 的准确度；之后展示了不同参数下单阶段快速处理方法的表现；最后对比了应对通信延迟和不应对通信延迟获得阈值的平均相对误差。

5.3.1 节点流无交集解决方案比较

这部分实验对比了 s 分传输、二分传输、全量传输的传输开销，模拟了 4 节点 15 个时间周期、5 节点 12 个时间周期、10 节点 6 个时间周期的情况，每种情况都完整使用了 60 min 的数据，依次设置 K 为 100，300，500，700，900。由于每种方案都是无误差的方案，它们的准确度都会是 100%，因此在这部分实验中没有比较准确度和平均相对误差。现有节点流无交集情况下 top-K 方法的绝大部分工作重点在其他方面，因此采用的方法都是全量传输，与全量传输进行对比即为与相关工作使用方法进行对比。

实验结果如图 5 所示，从实验结果可以看到，本文提出的先传输分段最小频数，得到阈值后再传输流标签的传输方案在 s 分和二分的情况下都可以减少 50%以上的传输开销，并且 s 分传输相比二分传输在所有情况下都可以减少更多的传输开销，这说明我们计算出的 s 取值确实可以减少更多的传输开销。同时也注意到，分布节点越多，本文方案相比全量传输节省的传输开销也越多，在 4 节点的情况下 s 分传输的开销约为全量传输的 30%，而 10 节点的情况中仅为 20%。相比全量传输，s 分传输开销大幅度减少，证明本文提出的节点流无交集的解决方案可以有效减少传输的冗余信息，又因为它是无误差方案，所以也可以有效识别强隐匿 top-K 频繁流。



(a)4 节点无交集传输开销　(b)5 节点无交集传输开销　(c)10 节点无交集传输开销

图 5　节点流无交集情况传输开销比较

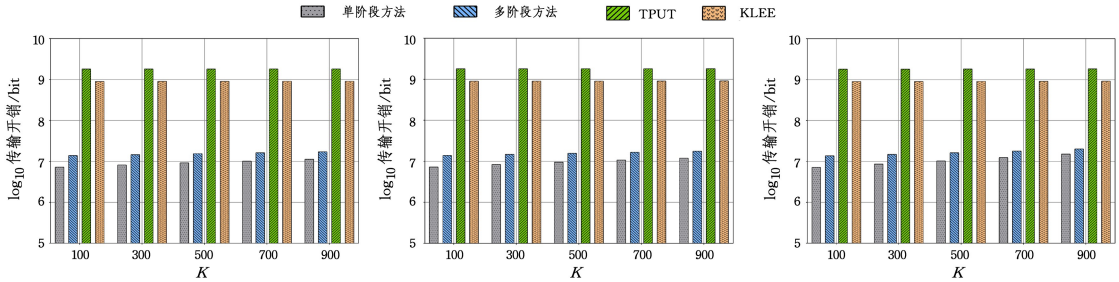
Fig. 5　Transmission overhead comparison when node flows have no intersection

5.3.2 节点流有交集解决方案比较

这部分实验先与经典的 TPUT 和 KLEE 算法进行对比，之后又展示了不同参数下单阶段快速处理方法的结果，验证了多阶段无误差处理方法和单阶段快速处理方法在连续时间周期的全局 top-K 测量中表现更好，达到了预期设计目标。需要说明的是，所有实验中分布节点上的 sketch 结构都采用了最经典的 CountMin Sketch，这也满足分布节点对数据包处理简单的设计目标。

算法对比实验中比较了单阶段快速处理方法、多阶段无误差方法、TPUT 和 KLEE 在 4 节点 15 个时间周期、5 节点 12 个时间周期、10 节点 6 个时间周期的情况下，K 为 100，300，500，700，900 的表现。由于 TPUT 和 KLEE 无法应对多周期的情况，也无法在流动态到来时进行处理，这两种算法在模拟时每个时间周期都进行了一次完整的算法过程，并且为了公平，4 种方案都在每个时间周期结束后统一处理。

本文首先比较了它们传输开销的表现。实验结果如图 6 所示，可以看出在处理 60min 的数据时，单阶段方法和多阶段方法相比 TPUT 和 KLEE 都大约可以节省两个数量级的传输开销，单阶段方法相比多阶段方法传输开销又有所减少。其中的原因有两点：1）TPUT 和 KLEE 都传输完整的流标签，而我们选择使用布隆过滤器传输限制信息，这点节省了一部分开销；2）本文方法不需要知道流分布情况，并且积极利用前一个时间周期的结果来减少新时间周期需要传输的信息，而 TPUT 和 KLEE 只适用于单个时间周期，它们在每个时间周期开始都需要通信传输冗余信息来重新获取流分布。节点流有交集的多阶段无误差处理方法和单阶段快速处理方法的传输开销更小，说明可以有效减少传输的冗余信息。

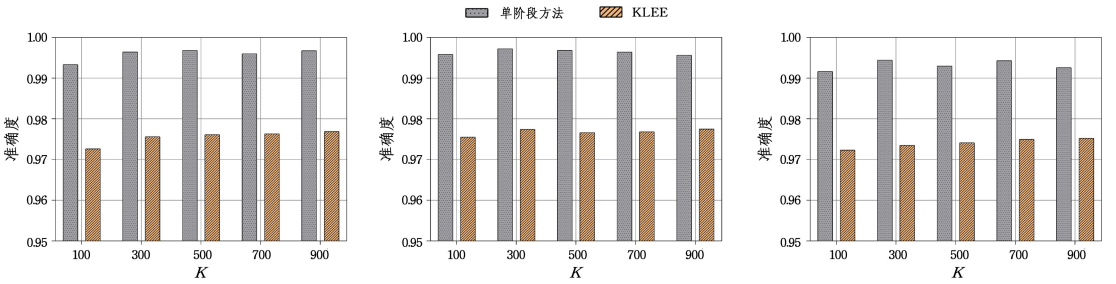


(a) 4 节点有交集传输开销　(b)5 节点有交集传输开销　(c)10 节点有交集传输开销

图 6 节点流有交集情况传输开销比较

Fig. 6 Transmission overhead when node flows have intersection

然后比较了不同算法的准确度表现。由于多阶段方法和 TPUT 都是无误差的方法，因此图 7 只给出了单阶段方法和 KLEE 所有周期的平均准确度。从图 7 中可以看到，在所有情况下单阶段快速处理方法的准确度都在 99% 以上，而 KLEE 的准确度为 97%～98%。这是因为 KLEE 是根据流数据的大致分布得到的全局 top-K 候选列表有误差，但我们的单阶段快速处理方法是把新出现的频数大于阈值的所有候选 top-K 频繁流都传输给中心节点后再处理得到全局 top-K，流的大致分布会有误差，但全局 top-K 频繁流在分布节点的频数一定大于计算得出的阈值。准确度更高说明单阶段快速处理方法可以有效识别强隐匿的 top-K 频繁流，这符合最初的设计目标。



(a)4 节点 15 周期平均准确度　(b)5 节点 12 周期平均准确度　(c)10 节点 6 周期平均准确度

图 7 单阶段快速处理方法和 KLEE 准确度对比

Fig. 7 Accuracy comparison of single-stage rapid processing method and KLEE

参数对比实验比较了单阶段快速处理方法在 4 节点 15 个时间周期条件下，K=100，300，500，700，900，CM 设置不同大小时的表现，CM 参数设置如表 1 所列，结果如图 8 所示。

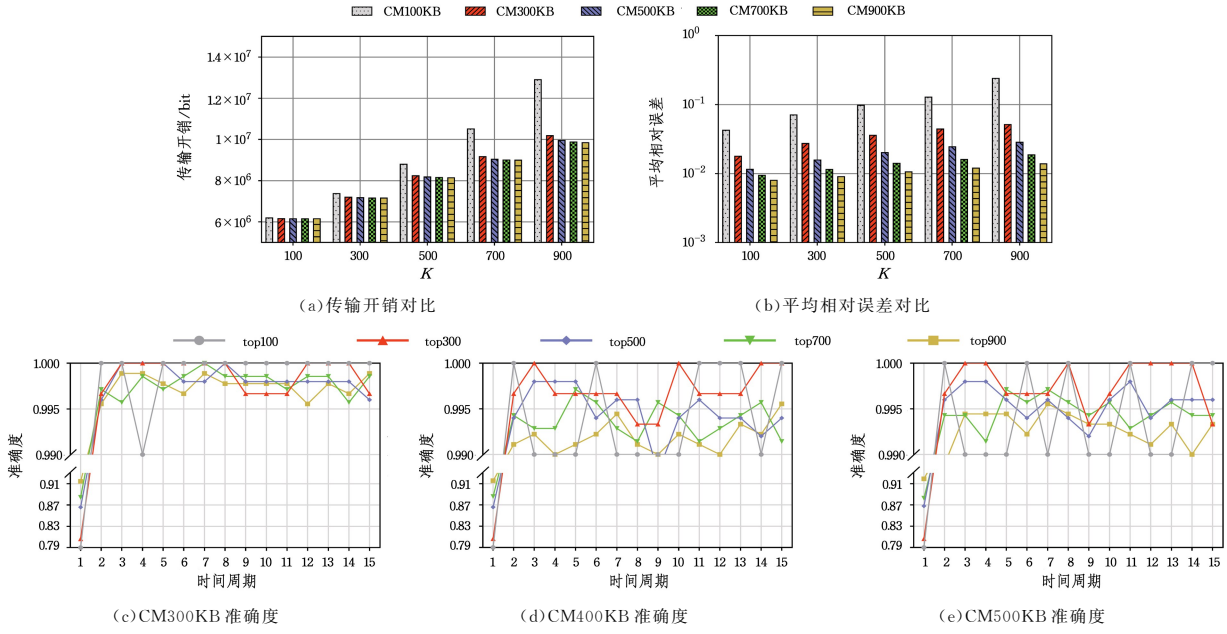
表 1 CM 不同空间大小的参数设置

Table 1 Parameter configuration for different sizes of CM

空间/kB	每层计数器数目	层数
300	19200	4
400	25600	4
500	32000	4

实验中 CM 的大小最大仅为 500 kB，这符合我们分布节点记录流信息空间开销小的设计目标。从图 8(a)和图 8(b)中可以看到，随着 CM 占用空间的增大，传输开销会减少，尤其是从 100 kB 增加到 200 kB 时传输开销显著减少，这是因为随着 CM 占用空间增大，其对流频数的估计会更准确，平均相对误差因此减小，同时会有更少的流因为频数估计错误而进入候选列表，从而进一步减少了传输开销。图 8(c)－图8(e)给出了 CM 占用 300 kB，400 kB，500 kB 时测得的全局 top-K

至每个时间周期结束的全局 top-K 频繁流。由于第一个时间周期没有额外的处理操作获得限制条件，所有参数设置下第一个时间周期的准确度都较低，这是为了提高处理速度的必要牺牲。CM 占用 300 kB 以上时，除第一个时间周期外绝大多数时间周期结束后准确度都达到了 99% 以上。这说明我们的测量机制在分布节点上使用较小空间测得的全局 top-K 频繁流的准确度较高，符合设计目标中分布节点记录流信息开销小和有效识别强隐匿流的要求。



(a)传输开销对比　(b)平均相对误差对比

(c)CM300KB 准确度　(d)CM400KB 准确度　(e)CM500KB 准确度

图 8　CM 不同参数下的准确度对比

Fig. 8　Accuracy comparison with different CM parameters

5.3.3　应对通信延迟比较

这部分实验比较了 4 节点 15 个时间周期情况下直接使用上一时间周期阈值和使用历史增值信息更新阈值的实际阈值的 ARE 平均值。从图 9 可以看到，在所有情况下使用历史增值信息更新阈值的方法的平均相对误差都小于直接使用上一时间周期阈值的方法，这说明本文方法可以有效减少由通信延迟导致的阈值更新不及时，进而造成更多流错误进入传输列表的情况，因此我们提出的应对通信延迟的方法也有效减少了传输的冗余信息。

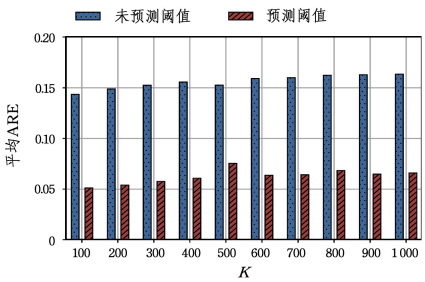


图 9　阈值平均相对误差对比

Fig. 9　Average relative error comparison of threshold

结束语　本文提出了在节点流无交集和节点流有交集的分布式网络环境中测量连续时间周期全局 top-K 频繁流的解决方案。两种情况下的解决方案都大大节省了传输开销，并且获得的全局 top-K 频繁流准确度也极高。本文的研究主要致力于通过传输机制减少传输开销，未来还可以通过对流标签的压缩等操作来进一步优化分布式网络全局 top-K 频繁流的测量工作。

参 考 文 献

[1] SHARMA H, THAKKAR P, BHARADWAJ S, et al. Optimizing Network Provisioning through Cooperation[C]// 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). Renton: USENIX Association, 2022: 1179-1194.

[2] HUANG H, SUN Y E, MA C Y, et al. Spread estimation with non-duplicate sampling in high-speed networks[J]. IEEE/ACM Transactions on Networking, 2021, 29(5): 2073-2086.

[3] CAIDA. Anonymized internet traces 2019 [EB/OL]. (2019-01) [2023-10-08]. https://catalog.caida.org/details/dataset/passive_2019_pcap.

[4] ZHAO Y K, HAN W C, ZHONG Z, et al. Double-Anonymous Sketch: Achieving Top-K-fairness for Finding Global Top-K Frequent Items[J]. Proceedings of the ACM on Management of Data, 2023, 1(1): 1-26.

[5] SONG W, BESHLEY M, PRZYSTUPA K, et al. A software deep packet inspection system for network traffic analysis and anomaly detection[J]. Sensors, 2020, 20(6): 1637.

[6] JIN K, XIE K, TIAN J Z, et al. Low cost network traffic measurement and fast recovery via redundant row subspace-based matrix completion [J]. Connection Science, 2023, 35 (1): 2218069.

[7] YANG T, ZHANG H W, LI J Y, et al. HeavyKeeper: an accurate algorithm for finding Top-k elephant flows[J]. IEEE/ACM Transactions on Networking, 2019, 27(5): 1845-1858.

[8] YANG T, JIANG J, LIU P, et al. Elastic sketch: Adaptive and

fast network-wide measurements[C]// Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. New York: Association for Computing Machinery, 2018:561-575.

[9] LIU L T, SHEN Y L, YAN Y B, et al. Sf-sketch: a two-stage sketch for data streams[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(10):2263-2276.

[10] DAI H P, LI M, LIU A, et al. Finding persistent items in distributed datasets[J]. IEEE/ACM Transactions on Networking, 2019, 28(1):1-14.

[11] LI M, DAI H P, WANG X Y, et al. Thresholded monitoring in distributed data streams[J]. IEEE/ACM Transactions on Networking, 2020, 28(3):1033-1046.

[12] BASAT R, EINZIGER G, TAYH B. Cooperative network-wide flow selection[C]// 2020 IEEE 28th International Conference on Network Protocols(ICNP). Madrid: IEEE Press, 2020:1-11.

[13] CAO P, WANG Z. Efficient top-k query calculation in distributed networks[C]// Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing. New York: Association for Computing Machinery, 2004:206-215.

[14] MICHEL S, TRIANTAFILLOU P, WEIKUM G. Klee: A framework for distributed top-k query algorithms[C]// Proceedings of the 31st International Conference on Very Large Data Bases. New York: ACM, 2005:637-648.

[15] ANCEAUME E, BUSNEL Y, RIVETTI N, et al. Identifying global icebergs in distributed streams[C]// 2015 IEEE 34th Symposium on Reliable Distributed Systems(SRDS). Montreal: IEEE Press, 2015:266-275.

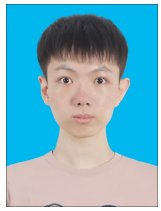
[16] TARKOMA S, ROTHENBERG C, LAGERSPETZ E. Theory and practice of bloom filters for distributed systems[J]. IEEE Communications Surveys & Tutorials, 2011, 14(1):131-155.

[17] TING D. Data sketches for disaggregated subset sum and frequent item estimation[C]// Proceedings of the 2018 International Conference on Management of Data. New York: Association for Computing Machinery, 2018:1129-1140.

[18] LI J Z, LI Z K, XU Y F, et al. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams[C]// Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York: Association for Computing Machinery, 2020:1574-1584.

[19] CORMODE G, MUTHUKRISHNAN S. An improved data stream summary: the count-min sketch and its applications[J]. Journal of Algorithms, 2005, 55(1):58-75.

[20] ESTAN C, VARGHESE G. New directions in traffic measurement and accounting[C]// Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. New York: New Directions in Traffic Measurement and Accounting, 2002:323-336.



MAO Chenyu, born in 2000, postgraduate, is a student member of CCF(No. L5662G). His main research interest is network traffic measurement.

SUN Yu'e, born in 1983, Ph.D supervisor, is a member of CCF(No. 28402M). Her main research interests include traffic measurement, Internet of Things, privacy preserving and crowd sensing.

(责任编辑:喻藜)