软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(3):869−882 [doi: 10.13328/j.cnki.jos.005440]

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

应对倾斜数据流在线连接方法*

王春凯, 孟小峰

(中国人民大学 信息学院,北京 100872)

通讯作者: 孟小峰, E-mail: xfmeng@ruc.edu.cn

摘 要: 并行环境下的分布式连接处理要求制定划分策略以减少状态迁移和通信开销.相对于数据库管理系统而言,分布式数据流管理系统中的在线θ连接操作需要更高的计算成本和内存资源.基于完全二部图的连接模型可支持分布式数据流的连接操作.因为连接操作的每个关系仅存放于二部图模型的一侧处理单元,无需复制数据,且处理单元相互独立,因此该模型具有内存高效、易伸缩和可扩展等特性.然而,由于数据流速的不稳定性和属性值分布的不均衡性,导致倾斜数据流的连接操作易出现集群负载不均衡的现象.针对倾斜数据流的连接操作,模型无法动态分配查询节点,并需要人工干预数据分组的参数设置.尤其是应对全部历史数据的连接查询,模型效率更低.基于上述问题,提出了管理倾斜数据流连接的框架,使用基于键值和元组混合的划分样式,有效应对二部图模型的各侧倾斜数据.设计了重新动态分配查询节点的策略和状态迁移算法,以支持全历史数据的连接查询和自适应的资源管理.针对合成数据和真实数据的实验结果表明,该方案可有效应对倾斜数据的连接操作,并进一步提升分布式数据流管理系统的吞吐率,特别是降低云环境中的计算成本.

关键词: 分布式数据流管理系统;在线连接;数据倾斜;状态迁移;二部图连接模型

中图法分类号: TP311

中文引用格式: 王春凯,孟小峰.应对倾斜数据流在线连接方法.软件学报,2018,29(3):869−882. http://www.jos.org.cn/1000-9825/5440.htm

英文引用格式: Wang CK, Meng XF. Online join method for skewed data streams. Ruan Jian Xue Bao/Journal of Software, 2018,29(3):869−882 (in Chinese). http://www.jos.org.cn/1000-9825/5440.htm

Online Join Method for Skewed Data Streams

WANG Chun-Kai, MENG Xiao-Feng

(School of Information, Renmin University of China, Beijing 100872, China)

Abstract: Scalable distributed join processing in a parallel environment requires a partitioning policy to transfer data while minimizing the size of migrated statement and the number of communicated messages. Online theta-joins over data streams are more computationally expensive and impose higher memory requirement in distributed data stream management systems (DDSMS) than standalone database management systems (DBMS). The complete bipartite graph-based model can support distributed stream joins, and has the characteristics of memory-efficiency, elasticity and scalability. This is because each relation is stored in its corresponding processing units without data replicas and the units are independent of each other. However, due to the instability of data stream rate and the imbalance of attribute

* 基金项目: 国家自然科学基金(61532016, 61379050, 61532010, 91646203, 61762082); 国家重点研发计划(2016YFB1000602, 2016YFB1000603); 中国人民大学科学研究基金(11XNL010); 河南省科技开放合作项目(172106000077)

Foundation item: National Natural Science Foundation of China (61532016, 61379050, 61532010, 91646203, 61762082); The National Key Research and Development Program of China (2016YFB1000602, 2016YFB1000603); The Research Funds of Renmin University (11XNL010); the Science and Technology Opening up Cooperation project of He'nan Province (172106000077)

本文由基于图结构的大数据分析与管理技术专刊特约编辑林学民教授、杜小勇教授、李翠平教授推荐.

收稿时间: 2017-07-31; 修改时间: 2017-09-05; 采用时间: 2017-11-07; jos 在线出版时间: 2017-12-05

CNKI 网络优先出版: 2017-12-06 15:23:09, http://kns.cnki.net/kcms/detail/11.2560.TP.20171206.1522.005.html

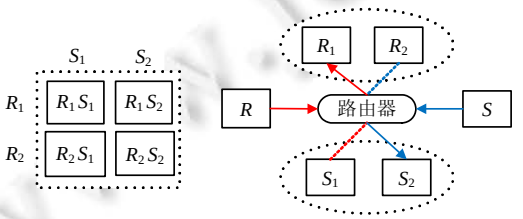
value distribution, the online theta-joins over skewed data streams can lead to the load imbalance of cluster. In this case, the bipartite graph-based model is unable to allocate the query nodes dynamically, and requires to set parameters about the grouping manually. The more serious issue is that the effect of the full-history join is worse. In this paper, a framework for handling skewed stream join is presented for enhancing the adaptability of the join model and minimizing the system cost based on the varying workloads. The proposal includes a mixed key-based and tuple-based partitioning scheme to handle skewed data in each side of the bipartite graph-based model, a strategy for redistribution of query nodes in two sides of this model, and a migration algorithm about state consistency to support full-history joins and adaptive resource management. Experiments with synthetic data and real data show that the presented method can effectively handle skewed data streams and improve the throughput of DDSMS, and it also effective especially on reducing the operational cost in the cloud environment.

Key words: distributed data stream management system; online join; data skew; state migration; bipartite graph-based join model

近年来,随着数据类型的增多和数据密集型应用的不断涌现,数据数量和流速在快速增加,这使得数据流的实时分析和流式处理成为当今热点研究领域之一[1−3].因此,分布式数据流管理系统(distributed data stream management system,简称 DDSMS)被广泛应用于大规模数据流的实时处理和查询分析.分布式数据流管理系统往往由上层的关系查询系统(relational query system,简称 RQS)和下层的流处理系统(stream processing system,简称 SPS)组成.目前,有许多开源的流处理系统,如 S4[4]、Storm[5]等.为提高其易用性和处理能力,提供查询语言的关系查询系统也相继推出,如 Squall[6]、Calcite[7]等.当用户向关系查询系统提交查询请求时,查询任务被转换成由多个子任务构成的有向无环图(directed acyclic graph,简称 DAG),并运行于流处理系统.

在社交网络的微博分析、金融领域的高频交易监控和电商领域的实时推荐服务等应用中,往往涉及多个数据流的连接查询和分析.连接操作需要维护大量的状态信息,并依赖于全部历史(full-history)数据[8,9].在这些应用中,数据流速往往发生波动且属性值的分布也较不均衡.这使得倾斜数据流的连接操作易出现集群负载不均衡的现象,导致了连接查询的效率降低和云环境下计算成本的升高.由于数据流速和分布的不均衡性引起了属性值倾斜(attribute value skew,简称 AVS)[10]和由数据划分带来的元组放置倾斜(tuple placement skew,简称 TPS)[10]等问题.因此,如何应对倾斜数据流的高效连接和集群的负载均衡,是本文重点关注的问题.

为支持任意连接谓词和应对数据倾斜的问题,基于矩阵的连接模型[11]和基于完全二部图的连接模型[12]是最具代表性的两种模型.矩阵连接模型利用划分规则将数据流随机分裂成不重叠的子流.如图 1(a)所示,数据流 R(或 S)被随机划分为 R_1 和 R_2(或 S_1 和 S_2),由于各子流需要复制到多个不同的处理单元,该模型在每个矩阵单元格内实现了全历史数据任意谓词的 θ连接.作为替代矩阵模型的代表,二部图连接模型将处理单元组织成完全二部图的形式,每侧对应一个数据流.如图 1(b)所示,流 R 和 S 被划分至不同的两侧,并根据基于键值的划分方法(如哈希函数),各元组被分配至同一侧的不同节点进行存储,如 R_1 和 R_2(或 S_1 和 S_2).与此同时,元组通过相同的哈希函数被发送至对面一侧并完成连接操作,待完成连接操作后,丢弃该元组.图 1(b)展示了该连接的处理过程:从 R 发出的元组存储于 R_1,并同时发送至 S_1 完成连接操作;从 S 发出的元组存储于 S_2,并发送至 R_2 完成连接操作.



(a) 矩阵连接模型 (b) 二部图连接模型

Fig.1 Representative join models

图 1 代表性的连接模型

矩阵连接模型和二部图连接模型可有效处理分布式数据流的在线连接操作,但是面临如下问题和挑战.

1) 矩阵连接模型需要使用较多额外内存用于整行和整列数据的复制与存储;二部图连接模型尽管节省了内存空间,但无法根据数据流的倾斜情况动态调整处理单元的分布;

2) 由于数据流分布的不一致性和数据操作的非对称性,导致集群的负载不均衡,严重影响了数据流管理系统的性能;

3) 具有状态信息的连接操作需要数据流管理系统具有较好的可扩展性.当某一节点的压力过大(或过小)时,集群规模可根据应用负载动态增加(或减少).

本文基于二部图连接模型提出了应对倾斜数据流在线连接的数据划分策略,以实现集群的负载均衡并保证分布式数据流管理系统的高吞吐率.本文的主要贡献如下:

1) 关于二部图连接模型中单侧节点的数据倾斜问题,提出了基于键值和元组混合的数据划分模式.并结合动态迁移策略中涉及的不同代价类型,提出归一化的优化目标.通过周期性地监控各处理单元的负载情况,动态制定迁移策略,以实现最小化的总体代价,从而实现单侧节点间的负载均衡;

2) 根据数据流速的倾斜情况,设计了二部图连接模型两侧处理节点之间的重分布策略.通过逻辑重划分查询节点,动态实现连接算法的负载均衡.并在开源流处理系统 Storm 上实现该处理流程,利用合成数据和真实数据的不同查询任务,证实了算法的有效性和可行性;

3) 为支持全历史数据的连接查询和自适应的系统资源管理,给出了确保状态一致性的迁移算法.使用算子状态管理器动态迁移不同节点之间的处理单元,以保证连接算子的状态一致性和可扩展性.

本文引言部分给出倾斜数据流在线连接的问题描述.第 1 节介绍相关工作.第 2 节给出一些预备知识.第 3 节提出应对倾斜数据流的处理框架.第 4 节重点描述算法的设计过程和实例说明.第 5 节给出实验验证及分析结果.第 6 节进行总结.

1 相关工作

针对分布式环境下数据流在线连接的相关研究工作,可从以下 4 个方面归纳概括.

1) 连接类型

为处理不同查询请求,在线连接类型分为等值连接和θ连接两种.由谷歌公司开发的 Photon[13]是针对操作网络查询数据流和用户广告点击数据流的等值连接算子.D-Stream[14]是 Sparking Streaming[15]定义的操作对象,可支持多数据流的θ连接.利用 Spark[16]提供的 RDD[17]机制,确保查询处理的正确性和容错性.DYNAMIC[11]算子利用矩阵连接模型支持多数据流的θ连接.通过设计重组器(reshuffler)动态定义划分样式,确保最小化的数据输入装载因子(input-load factor).JB 算子[12]利用二部图连接模型也支持多数据流的θ连接,并可根据数据的加载程度扩展处理单元的数量.

2) 连接模型

针对不同类型的连接算子,其连接模型各不相同.Photon[13]利用中心协调器(central coordinator)模型实现多数据中心的容错和扩展连接.通过向中心协调器注册查询事件的方法和多数据中心的分布式架构,确保数据的完备性.D-Stream[14]利用 RDD 转换(transformation)模型,将数据流切分成一系列的微批次(mini-batch)进行处理.DYNAMIC[11]利用矩阵连接模型构造(n,m)映射模式,自动将处理节点划分成 $J(J=n\times m)$个矩阵区域.JB 算子利用二部图连接模型将集群划分成两个部分,降低了数据备份的冗余度并提高了资源利用率.

3) 处理方式

数据流的处理方式一般分为非阻塞的元组处理和阻塞的批处理.Photon[13]、DYNAMIC[11]和 JB 算子均属于非阻塞的元组处理,可确保实时获取数据流的连接查询结果.D-Stream[14]使用数据阻塞的批处理方式,需要将数据流切分成微批次的方式在 Spark Streaming[15]上处理.

4) 倾斜数据流的负载均衡

为确保集群的负载均衡,有效应对数据流倾斜的问题,各种连接模型均有应对措施.DYNAMIC[11]根据数据倾斜变化,提出自适应的重划分机制.Aleksandar 等人引入等重(equi-weight)直方图的概念[18],推出了多级负载均衡算法.然而,文献[11,18]要求处理单元的划分个数必须为 2 的幂次方.因此,矩阵模型的伸缩性较差,资源浪费严重.为降低计算成本,文献[19]利用构建不规则的矩阵模式,提出了提高成本效益的数据流连接算法.然而,其基

本思想仍是基于矩阵模型,存在一定的数据冗余问题.JB 算子可显著节省资源使用率,并通过引入混合路由策略 ContRand,在一定情况下解决了负载不均衡的问题.但是该策略需要人工干预数据分组的参数设置,并且不能根据数据流速动态分配处理单元.此外,在查询任务倾斜和变化的场景下,需要对处理单元中的数据做重新划分的迁移操作.文献[20]在设定相同键值元组未超过同一处理单元存储上限的情况下,将各处理单元的数据倾斜问题规约至装箱问题[21]进行处理.但数据倾斜程度较高时,会存在相同键值元组超过同一处理单元存储上限的情况.此时,需要对同一处理单元的相同键值元组分裂至不同的处理单元.该问题无法用文献[20]提出的方法解决.

2 预备知识

本节给出全历史数据在线连接算子的相关预备知识.

2.1 基本定义

全历史数据在线连接操作是数据流 R 和 S 的各个元组对满足连接谓词的操作,表示为 $R\bowtie S$.在线连接是一种需要额外内存空间记录中间结果和迁移信息的状态算子(stateful operator).为便于形式化定义和明确优化目标,本文涉及到的相关符号说明见表 1.

Table 1 Table of notations

表 1 符号表

符号	描述
B	完全二部图
R,S	数据流 R 和 S
K_{tup}	元组的键值
pu	处理单元(Storm 中称为任务实例(task instance))
PU	处理单元集合
m,n	二部图两侧的处理单元个数
$L(pu)$	处理单元 pu 的总负载
$\theta(pu)$	处理单元 pu 的负载均衡因子
$\theta_{\max}$	负载均衡因子的临界值

定义 1. 时刻 t,处理单元 pu 的负载均衡因子可定义为

$$\theta_t(pu)=|L_t(pu)-\bar{L}_t|/\bar{L}_t \tag{1}$$

其中,$\bar{L}_t$ 为处理单元集合(PU)总负载的平均值.$\bar{L}_t$ 定义为

$$\bar{L}_t=\sum_{pu=1}^{N_{pu}}(L_t(pu))\Big/N_{pu} \tag{2}$$

那么,如果$\theta_t(pu)\leqslant\theta_{\max}$,我们可认为处理单元 pu 在时刻 t 的负载是相对平衡的.

定义 2. 时刻 t,当存在 $pu(pu\in PU)$,使得$\theta_t(pu)>\theta_{\max}$ 时,我们认为处理单元 pu 在时刻 t 负载不均衡,需要给出 3 种数据迁移的机制:(1) 数据迁入,不同处理单元具有相同 K_{tup} 的元组迁入至起始处理单元;(2) 数据迁出,具有相同 K_{tup} 的全部元组从当前处理单元迁出至其他处理单元;(3) 数据分裂,具有相同 K_{tup} 的部分元组迁移至其他处理单元,剩余元组保持在当前处理单元.

定义 3. 根据数据流的分布和倾斜程度,我们需要动态制定迁移策略.根据不同策略,涉及到 3 种类型的代价:(1) 路由代价 $C_{routing}$,数据迁移后,为记录 K_{tup} 和处理单元的映射关系而维护迁移路由的代价;(2) 复制代价 $C_{duplication}$,数据分裂后,执行连接操作时复制相同 K_{tup} 元组带来的代价;(3) 迁移代价 $C_{migration}$,元组从某一处理单元迁移至其他处理单元的代价.

由定义 2 和定义 3 可知:数据迁入仅涉及到迁移代价;数据迁出涉及到路由代价和迁移代价;数据分裂涉及到路由代价、复制代价和迁移代价.

2.2 优化目标

时刻 t,所有迁移函数的集合 $F=\{f_1,f_2,f_3,\ldots\}$.根据定义 3,每个迁移函数 f_i 的代价可表示为

$$C_t(f_i)=\alpha\times C_{routing}(f_i)+\beta\times C_{duplication}(f_i)+\gamma\times C_{migration}(f_i),\alpha+\beta+\gamma=1 \tag{3}$$

其中,α,β和γ是 3 种代价的权重.数据迁移的优化目标可表示为

$$\min_{f_i \in F} C_t(f_i) \quad \text{s.t.} \quad \theta_t(pu) \leqslant \theta_{\max}, \forall pu \in PU \tag{4}$$

当满足负载均衡的条件下,该优化目标是最小化迁移函数的总体代价.这涉及到 K_{tup} 的范围、全部处理单元的数目和最大非平衡因子的边界值$\theta_{\max}$.由于具有相同 K_{tup} 的数据在同一处理单元内部也有可能被分裂,这比文献[20]中的装箱问题更为复杂.因此,本文的优化目标可规约为 NP 问题.接下来,本文将给出解决该问题的系统架构和一系列的启发式规则.

3 系统架构

本节详细介绍应对倾斜数据流的动态负载均衡机制.基于二部图连接模型,在流处理系统上构建控制器(controller),用于周期性地监控各处理单元的负载情况.系统整体架构与工作流程如图 2 所示.

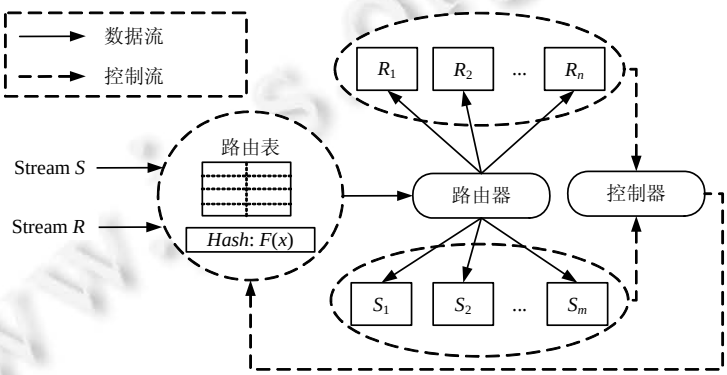


Fig.2 Architecture of workflow

图 2 工作流程架构图

- 首先,数据流 R(或 S)使用基于键值的哈希函数进行划分,分别存储在 n(或 m)个处理单元中,并同步将数据元组发送至另一侧处理单元以完成在线连接的操作;
- 然后,以固定时间间隔(本文的实验设置为 5s),周期性地监控二部图模型每侧节点的负载统计信息,并搜集发送至控制器:若控制器监控到某些处理单元超过负载均衡因子的临界值,我们则根据本文提出的启发式规则(见第 4.1 节)动态制定迁移策略,以实现最小化总体代价的优化目标;
- 接下来,在数据迁移之前,将新产生的数据流暂存在 Kafka[22]中,暂缓新数据的连接操作.此时,我们按照迁移策略进行数据流和连接状态信息的迁移,并同步更新路由表(routing table);
- 最后,继续发送 Kafka 中暂存的和新到来的数据,完成后续的在线连接操作.

4 数据迁移算法

在处理单元内部,数据出现负载过重的情况下,需要对内部数据进行划分和迁移.这比将处理单元当作整体进行数据迁移的情况更为复杂.因此本节需要制定一系列的启发式规则优化该目标函数,并提出两种优化策略:一种是针对二部图中单侧内部节点的数据迁移(internal-side-migration,简称 ISM);另一种是针对二部图两侧节点的逻辑迁移(side-to-side migration,简称 S2SM).

4.1 启发式规则

时刻 t,假设存在某一处理单元的负载超过非平衡因子的临界值上限,我们将其标记为 $pu_{\max}$,即 $L_t(pu_{\max}) > (1+\theta_{\max}) \times \overline{L_t}$,或者存在某一处理单元负载低于非平衡因子的临界值下限,我们将其标记为 $pu_{\min}$,即 $L_t(pu_{\min}) < (1-\theta_{\max}) \times \overline{L_t}$.为满足二部图连接模型各处理单元的平衡性,并尽量减少数据迁移的情况,我们设定的启发式规则

如下.

1) H1.数据需要迁出的处理单元,如果迁出负载键值的元组后可直接满足非平衡因子阈值的要求,则直接进行迁出操作,并在路由表中记录迁移键值;
2) H2.数据需要迁出的处理单元,如果迁出某些键值的元组后仍不满足非平衡因子阈值的要求,则需要切分具有较高元组数的键值,将切分后的部分数据进行迁出操作,并在路由表中记录迁移键值;
3) H3.数据需要迁入的处理单元,如果存在键值在路由表中,则优先将该键值的元组合并至哈希函数映射的处理单元,并清空路由表中的记录.

根据上述 3 种规则,我们分别给出迁出元组和迁入元组的基本算法.其中,迁入元组的具体流程见算法 1.算法首先判断迁出集合中迁出元组的键值范围(第 1 行~第 3 行),并确定待迁入元组的处理单元(第 4 行~第 6 行),然后针对各个迁出键值按照启发式规则 H1 和 H2 完成数据迁出,并更新路由表(第 7 行~第 10 行),最终确定迁移计划.

算法 1. MoveOut 算法.

输入:迁出处理单元集合 PU_{out},迁入处理单元集合 PU_{in},路由表 RT;

输出:迁移计划 MP.

1. **for** (i=1; i<$number\_of\_PU_{out}$; i++)
2. 计算迁出元组的键值范围 R_k;
3. **end for**
4. **for** (j=1; j<$number\_of\_PU_{in}$; j++)
5. 确定待迁入元组的处理单元 pu_{in};
6. **end for**
7. **for** (k=1; k<R_k; k++)
8. 利用 H1 和 H2 迁出元组;
9. 更新路由表 RT;
10. **end for**

迁入元组的具体流程见算法 2 描述.算法首先判断迁入集合中迁入元组的键值范围(第 1 行~第 3 行),并确定待迁出元组的处理单元(第 4 行~第 6 行),然后针对各个迁入键值,按照启发式规则 H3 完成数据迁入,并更新路由表(第 7 行~第 10 行),最终确定迁移计划.

算法 2. MoveIn 算法.

输入:迁入处理单元集合 PU_{in},迁出处理单元集合 PU_{out},路由表 RT;

输出:迁移计划 MP.

1. **for** (i=1; i<$number\_of\_PU_{in}$; i++)
2. 计算迁入元组的键值范围 R_k;
3. **end for**
4. **for** (j=1; j<$number\_of\_PU_{out}$; j++)
5. 确定待迁出元组的处理单元 pu_{out};
6. **end for**
7. **for** (k=1; k<R_k; k++)
8. 利用 H3 迁入元组;
9. 更新路由表 RT;
10. **end for**

4.2 ISM算法

时刻 t,为满足二部图连接模型单侧处理单元的平衡性,并尽量减少数据迁移的情况,单侧的数据迁移的

ISM 算法见算法 3 描述.算法首先统计时刻 t 每个计算单元的负载 $L_t(pu)$,并计算出平均负载 $\overline{L_t}$ (第 1 行~第 4 行);然后,对于需要迁出数据的处理单元,调用 MoveOut 算法(第 6 行~第 8 行);最后,对于需要迁入数据的处理单元,调用 MoveIn 算法(第 9 行~第 11 行).

算法 3. ISM 算法.

输入:二部图单侧处理节点 *PU*,路由表 *RT*,非平衡因子的临界值 $\theta_{\max}$;

输出:迁移计划 *MP*.

1. **for** (i=1; i<*number_of_PU*; i++)
2. 计算 $L_t(pu_i)$;
3. **end for**
4. 计算 $\overline{L_t}$;
5. **for** (i=1; i<*number_of_PU*; i++)
6. **if** ($L_t(pu_i)$>(1+$\theta_{\max}$)×$\overline{L_t}$) **then**
7. *MoveOut*(*PU*,*PU*,*RT*);
8. **end if**
9. **if** ($L_t(pu_i)$<(1−$\theta_{\max}$)×$\overline{L_t}$) **then**
10. *MoveIn*(*PU*,*PU*,*RT*);
11. **end if**
12. **end for**

我们用实例说明 ISM 算法的处理过程.假设每个处理单元的负载要求完全一致,即 $\theta_{\max}$=0.如图 3(a)所示,时刻 t_1:处理单元 pu_1 包含键值 k_1 和 k_2,元组数分别为 25 和 5;pu_2 包含键值 k_3 和 k_4,元组数分别为 5 和 5.根据启发式规则 H1 和 H2,需要将 k_1 元组切分成 15 和 10,并将其中 10 个元组迁移至 pu_2,同步更新路由表.如图 3(b)所示:在下一时刻 t_2,由于数据的倾斜原因,导致 k_4 的元组增加至 45 个,其他不变.根据启发式规则 H3,我们首先需要将 pu_2 中的 k_1 元组移回 pu_1,并清空路由表;然后,再将 k_4 元组切分成 35 和 10,并将其中 10 个元组迁移至 pu_1,同步更新路由表.

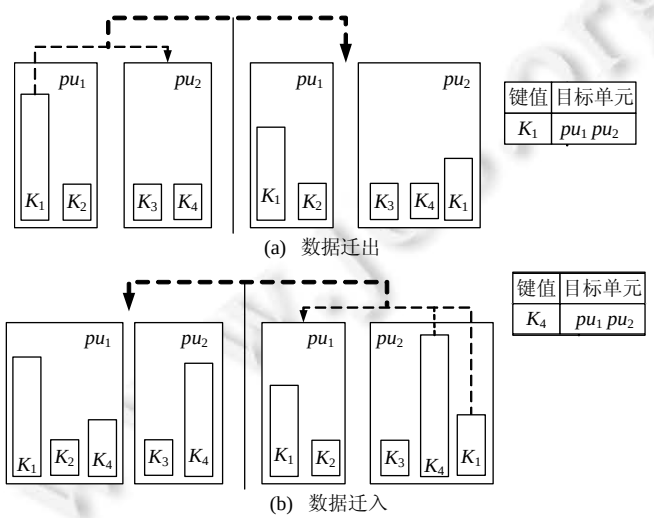


Fig.3 Example of ISM

图 3 ISM 算法举例

4.3 S2SM算法

由于数据流速的动态改变,导致二部图模型中两侧数据流的数量会有较大间隙,这严重影响到分布式数据

流管理系统的吞吐率.为此,我们给出两侧节点逻辑迁移的 S2SM 算法(见算法 4).该算法首先统计每个单元的负载 $L_t(pu)$,并分别统计各侧和整个集群的平均负载 $\overline{L}_{tm}$,$\overline{L}_{tm}$,$\overline{L}_t$ (第 1 行~第 4 行);然后,根据临界值判定迁出元组的一侧和迁入元组的一侧(第 5 行~第 14 行);最后,针对迁出侧,判断需要迁出的处理单元并调用迁出算法(第 15 行~第 19 行),针对迁入侧,判断需要迁入的处理单元并调用迁入算法(第 20 行~第 24 行).

算法 4. S2SM 算法.

输入:二部图两侧处理节点 PU_m 和 PU_n,路由表 RT,非平衡因子的临界值 $\theta_{\max}$;

输出:迁移计划 MP.

1. **for** (i=1; i<*number_of_*(PU_m+PU_n); i++)
2. 计算 $L_t(pu_i)$;
3. **end for**
4. 计算 $\overline{L}_{tm}$,$\overline{L}_{tm}$,$\overline{L}_t$;
5. **if** ($\overline{L}_{tm} > (1+\theta_{\max})\times\overline{L}_t$) **then**
6. PU_{out}=PU_m;
7. **else if** ($\overline{L}_{tm} < (1-\theta_{\max})\times\overline{L}_t$) **then**
8. PU_{in}=PU_m;
9. **end if**
10. **if** ($\overline{L}_{tn} > (1+\theta_{\max})\times\overline{L}_t$) **then**
11. PU_{out}=PU_n;
12. **else if** ($\overline{L}_{tn} < (1-\theta_{\max})\times\overline{L}_t$) **then**
13. PU_{in}=PU_n;
14. **end if**
15. **for** (j=1; j<*number_of_*PU_{out}; j++)
16. **if** ($L_t(pu_j)$>$(1+\theta_{\max})\times\overline{L}_t$) **then**
17. *MoveOut*(PU_{out},PU_{in},RT);
18. **end if**
19. **end for**
20. **for** (k=1; k<*number_of_*PU_{in}; k++)
21. **if** ($L_t(pu_k)$<$(1-\theta_{\max})\times\overline{L}_t$) **then**
22. *MoveIn*(PU_m,PU_{out},RT);
23. **end if**
24. **end for**

我们用实例说明 S2SM 算法的处理过程.假设每个处理单元的负载要求完全一致,即 $\theta_{\max}$=0.如图 4 所示,时刻 t,二部图连接模型共包括 4 个处理单元 pu_1~pu_4,其中,pu_1 和 pu_2 用于存储数据流 R,pu_3 和 pu_4 用于存储数据流 S.pu_1 和 pu_2 包含键值 R_{k1}~R_{k4},元组数分别为 5,5,5 和 5;pu_3 和 pu_4 包含键值 S_{k1}~S_{k4},元组数分别为 25,5,25 和 5,集群和两侧负载分别为 20,10 和 30.因此,我们需要将数据流 S 的部分元组迁移至 pu_1 和 pu_2.按照迁出规则 H1 和 H2,需要将 S_{k1} 元组数切分成 15 和 10,其中 10 个元组迁移至 pu_1,同步更新路由表;并将 S_{k4} 元组数切分成 15 和 10,其中 10 个元组迁移至 pu_2,同步更新路由表.

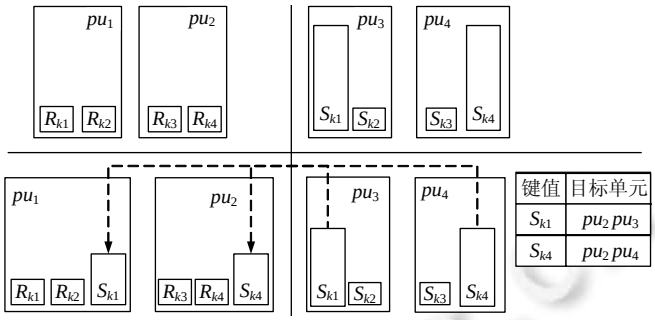


Fig.4 Example of S2SM

图 4 S2SM 算法举例

4.4 状态一致性迁移

基于二部图连接模型设计的 BiStream 系统[12]实现了动态扩展(或缩减)查询节点的资源管理策略,但其仅支持基于窗口的连接模型,无法对全历史数据的在线连接提供状态一致性的保证.本节引入内存文件系统 Tachyon[23]存储算子的状态信息,并使用算子状态管理器(operator states manager,简称 OSM)[24]动态迁移不同节点间的处理单元,以保证连接算子的状态一致性和可扩展性.我们利用该管理器实现了全历史数据在线连接操作的动态资源管理.状态管理器的架构图如图 5 所示.节点中的每个处理进程(storm 中称作 worker)管理自身的处理单元(pu_1~pu_n).状态管理器负责维护处理单元的数据不被当作垃圾回收,并将处理单元按照维持、迁出和迁入这 3 种状态进行管理.维持状态的处理单元仍运行在本进程中;迁出状态的处理单元将被迁移至其他节点,相关状态信息保存至文件系统(file server);迁入状态的处理单元由其他节点迁入该节点,并将其状态信息通过文件系统加载至恢复线程(retriever).该管理器确保了全历史数据在线连接操作的动态资源管理.

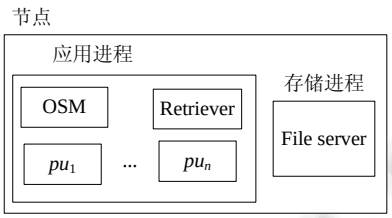


Fig.5 Architecture of OSM[24]

图 5 OSM 架构[24]

5 实验与结果分析

5.1 实验准备

1) 实验环境

本文实验平台用 1GB 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 各节点配置是:CPU:Intel E5-2620 2.00GHz,Memory:4GB;supervisor 各节点配置是:CPU:两个 Intel E5-2620 2.00GHz,Memory:64GB;操作系统:Ubuntu-14.04.3;Storm 版本 0.9.5.

2) 数据集

我们使用合成数据和真实数据进行实验对比.首先,以 TPC-H[25]作为基准测试,并利用 *dbgen* 产生数据集.所有数据在发送至 Storm 之前,均存放于数据发送节点的 Kafka 中.为测试数据集的不同倾斜程度,在连接属性上使用不同倾斜度 z 的 Zipf 分布.默认情况下,我们令 z=1,并产生 10GB 数据.其次,第 2 个数据集是 Linear Road

Benchmark(LRB)[26],该基准测试模拟高速公路的收费系统,统计不同路段、不同方向的车速信息.我们产生了1 200 万条车辆数据信息,如图 6 所示,车速分布具有明显的倾斜性.

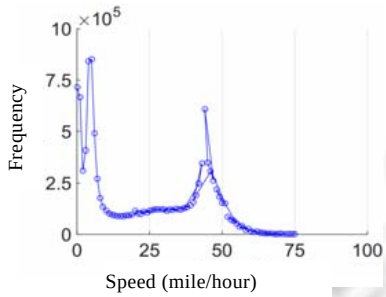


Fig.6 Speed distribution of LRB

图 6 LRB 车速分布图

3) 查询任务

本文共选取 4 个查询任务,其中两个是 TPC-H 提供的等值连接 Q3 和 Q5,一个是文献[11,12]中使用的范围查询(band).Band 查询描述为:

- SELECT *, FROM LINEITEM *L*1, LINEITEM *L*2
  WHERE *ABS*(*L*1.*orderkey*–*L*2.*orderkey*)≤1
  AND (*L*1.*shipmode*='TRUCK' AND *L*2.*shipinstruct*='NONE') AND *L*1.*Quantity*>48

另一个是 LRB 场景下,通过自连接操作统计同一车道下车速相同的车辆信息,其查询描述为:

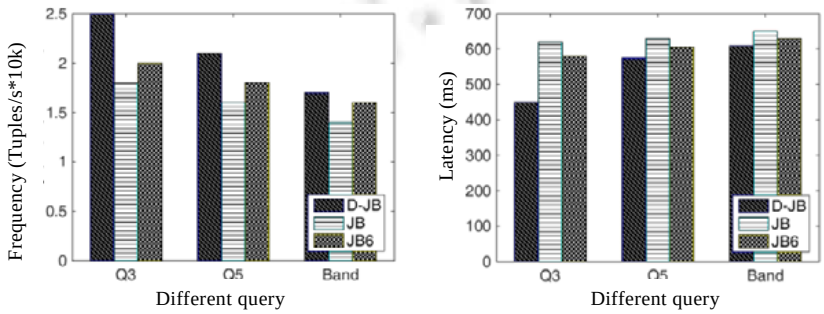
- SELECT *, FROM LRB *L*1, LRB *L*2
  WHERE *L*1.*Spd*=*L*2.*Spd* AND *L*1.*Lane*=*L*2.*Lane* AND *L*1.*Dir*=*L*2.*Dir* AND *L*1.*VID*!=*L*2.*VID*

4) 对比模型

本文使用 3 种算法对比分析查询性能:D-JB,JB 算子和 JB6.D-JB 是本文提出的算法,表示动态二部图连接模型.JB 算子表示平均分配集群节点至二部图的各侧.JB6 表示平均分配节点后,二部图各侧内部的处理单元分成 6 个子组做随机路由.

5.2 吞吐率和延迟

我们使用 TPC-H 中 *z*=1 的 10GB 数据,针对 3 个不同的查询任务,对比了 3 个模型的吞吐率和延迟.如图 7(a)所示,由于 D-JB 动态调整了处理单元的负载情况,其吞吐率最高.而 JB 算子需要做单侧的全网广播操作,通信量较大,故其吞吐率最低.针对不同查询任务的 topology 处理延迟而言,图 7(b)说明,D-JB 的延迟均低于 JB 算子和 JB6.



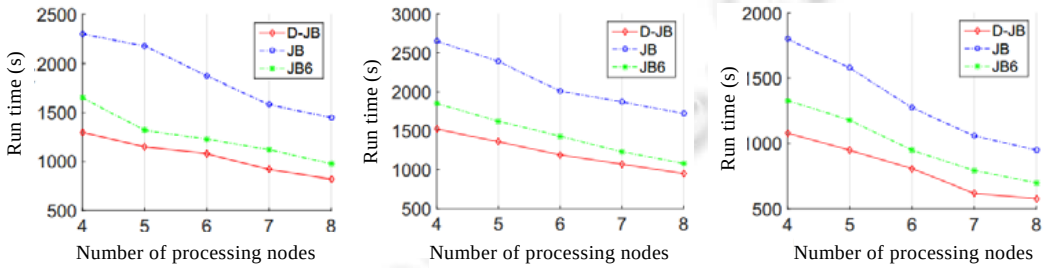
(a) Throughput (b) Latency

Fig.7 Throughput and latency

图 7 吞吐率和延迟

5.3 系统扩展性

当集群节点动态改变时,我们进一步分析 D-JB,JB 算子和 JB6 的可扩展性.由于 JB 算子和 JB6 不支持全历史数据连接的动态扩展,在增加处理节点时,我们将 TPC-H 数据暂存于 Kafka 的消息队列,待重启 Storm 的 topology 后继续发送数据(需减去重启 topology 的时间,以获得计算连接操作的实际运行时间).如图 8(a)~图 8(c)所示:D-JB 的运行时间是最短的,其扩展性最好;此外,由于 Q5 涉及到的连接操作最复杂,涉及的数据流最多,其运行时间高于其他两个查询任务.



(a) Q3 查询 (b) Q5 查询 (c) Band 查询

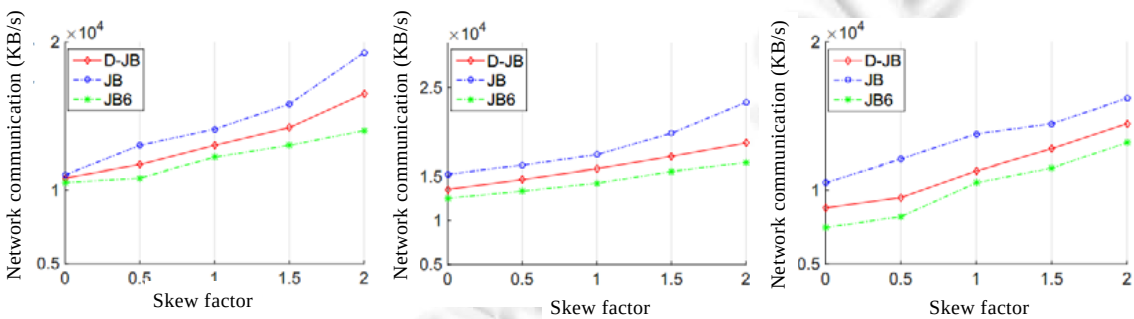
Fig.8 Run time

图 8 运行时间

5.4 通信代价

为验证 D-JB 模型应对不同倾斜度数据流的处理情况,我们通过改变 Zipf 的分布情况,测试 3 种模型的平均网络通信代价和数据迁移总量.对比图 9(a)~图 9(c)我们发现:

- JB 算子需要做全网广播操作,其平均网络通信代价最高;
- JB6 仅做子网广播操作,其通信代价最低;
- D-JB 由于做了数据迁移操作,其通信代价略高于 JB6;并且,由于 Q5 涉及到的数据流最多,其通信代价高于 Q3 和 Band.



(a) Q3 查询 (b) Q5 查询 (c) Band 查询

Fig.9 Network communication cost

图 9 网络通信代价

此外,根据不同倾斜度,图 10 给出了执行 D-JB 算法的数据迁移总量.可以发现:随数据倾斜度的增加,由数据迁移引起的网络通信总量也随之增加,但迁移总量相对于全部数据流量来讲相对较低.

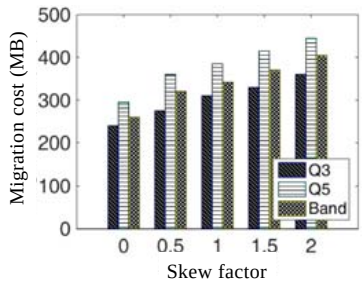


Fig.10 Total migration

图 10 迁移总量

5.5 归一化优化目标的选参设置

本文为设计归一化模型,为 3 个代价设置相关权重参数.首先,路由表仅在中心节点更新,维护代价最低;其次,在连接操作时,由于数据分裂导致的复制相同 K_{tup} 数据至多个节点,网络开销较高;最后,数据分裂导致多条数据在不同节点间迁移,网络开销最高.如图 11 所示,我们分别将α,β和γ设置为 0.2,0.3 和 0.5(conf_1);0.1,0.3,0.6(conf_2)和 0.1,0.4,0.5(conf_3).以 Q3 查询为例,我们可以发现,conf_1 的查询吞吐率最高.

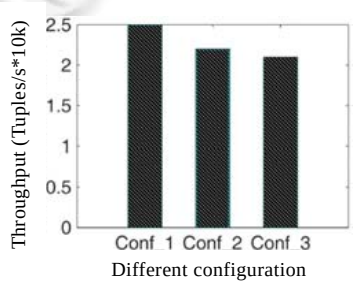


Fig.11 Throughput

图 11 吞吐率

5.6 真实数据流分析

除使用合成数据 TPC-H 外,我们还使用 LRB 的真实数据测试算法的有效性.执行第 5.1 节的查询任务,根据速度分布的倾斜性和数据量的不断增加,记录其执行时间.如图 12 所示,D-JB 算法的执行时间最短.

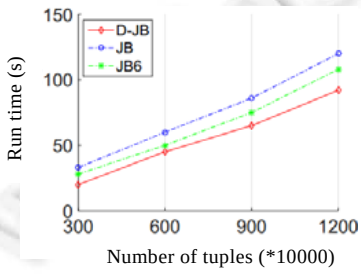


Fig.12 LRB run time

图 12 LRB 运行时间

6 总 结

本文提出了一种应对倾斜数据流的在线连接方法.基于二部图连接模型,在单侧节点内,我们设计了基于键值和元组的混合划分样式;在双侧节点间,我们制定了重新分布处理单元的策略;并通过引入状态管理器,实现

了可扩展的全历史数据连接操作.实验结果表明,本文提出的方法在系统性能、可扩展性和应对倾斜数据的能力等方面是可行且有效的.文中提到的数据迁移问题,文本通过制定归一化的优化目标并利用启发式规则以获取较优解.接下来,根据集群环境的不同,我们需要解决自动选取归一化优化目标的最优配置问题,并寻找更佳的优化方法使连接操作更加快速,系统更加稳定.

References:

[1] Sun DW, Zhang GY, Zheng WM. Big data stream computing: Technologies and instances. Ruan Jian Xue Bao/Journal of Software, 2014,25(4):839−862 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4558.htm [doi: 10.13328/j.cnki.jos.004558]

[2] Cui XC, Yu XH, Liu Y, Lü ZY. Distributed stream processing: A survey. Journal of Computer Research and Development, 2015, 52(2):318−332 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2015.20140268]

[3] Wang CK, Meng XF. Relational query techniques for distributed data stream: A survey. Computer Journal of Computers, 2016, 39(1):80−96 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2016.00080]

[4] Neumeyer L, Robbins B, Nair A, Kesari A. S4: Distributed stream computing platform. In: Proc. of the 10th EEE Int'l Conf. on Data Mining Workshops. Sydney: IEEE, 2010. 170−177. [doi: 10.1109/ICDMW.2010.172]

[5] Toshniwal A, Taneja S, Shukla A, Ramasamy K, Pater JM. Storm@Twitter. In: Proc. of the 2014 ACM Int'l Conf. on Management of Data. Snowbird: ACM Press, 2014. 147−156. [doi: 10.1145/2588555.2595641]

[6] Squall. https://github.com/epfldata/squall/

[7] Calcite. http://calcite.apache.org/

[8] Hwang JH, Balazinska M, Rasin A, Cetintemel U, Stonebraker M. High-Availability algorithms for distributed stream processing. In: Proc. of the 21st Int'l Conf. on Data Engineering. Tokyo: IEEE, 2005. 779−790. [doi: 10.1109/ICDE.2005.72]

[9] Fernandez RC, Migliavacca M, Kalyvianaki E, Pietzuch P. Integrating scale out and fault tolerance in stream processing using operator state management. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2013. 725−736. [doi: 10.1145/2463676.2465282]

[10] Walton CB, Dale AG, Jenevein RM. A taxonomy and performance model of data skew effects in parallel joins. In: Proc. of the 17th Int'l Conf. on Very Large Data Bases. Barcelona: Morgan Kaufmann Publishers, 1991. 537−548.

[11] Elseidy M, Elguindy A, Vitorovic A, Koch C. Scalable and adaptive online joins. Proc. of the VLDB Endowment, 2014,7(6): 441−452. [doi: 10.14778/2732279.2732281]

[12] Lin Q, Ooi BC, Wang Z, Yu C. Scalable distributed stream join processing. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Melbourne: ACM Press, 2015. 811−825. [doi: 10.1145/2723372.2746485]

[13] Ananthanarayanan R, Basker V, Das S, Gupta A, Jiang H. Photon: Fault-tolerant and scalable joining of continuous data streams. In: Proc. of the ACM 2013 Int'l Conf. on Management of Data. New York: ACM Press, 2013. 577−588. [doi: 10.1145/2463676.2465272]

[14] Zaharia M, Das T, Li H, Hunter T, Shenker S. Discretized streams: Fault-tolerant streaming computation at scale. In: Proc. of the ACM SIGOPS 24th Symp. on Operating Systems Principles. Farmington: ACM Press, 2013. 423−438. [doi: 10.1145/2517349.2522737]

[15] Spark streaming. https://spark.apache.org/streaming/

[16] Spark. http://spark.apache.org/

[17] Zaharia M, Chowdhury M, Das T, Dave A, Ma J. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proc. of the 9th USENIX Symp. on Networked Systems Design and Implementation. San Jose: USENIX, 2012. 15−28.

[18] Vitorovic A, Elseidy M, Koch C. Load balancing and skew resilience for parallel joins. In: Proc. of the 32nd IEEE Int'l Conf. on Data Engineering. Helsinki: IEEE, 2016. 313−324. [doi: 10.1109/ICDE.2016.7498250]

[19] Fang JH, Zhang R, Wang XT, Fu TZJ, Zhang ZJ, Zhou AY. Cost-Effective stream join algorithm on cloud system. In: Proc. of the 25th ACM Int'l Conf. on Information and Knowledge Management. Indianapolis: ACM Press, 2016. 1773−1782. [doi: 10.1145/2983323.2983773]

[20] Fang JH, Zhang R, Fu TZJ, Zhang ZJ, Zhou AY, Zhu JH. Parallel stream processing against workload skewness and variance. In: Proc. of the 26th Int'l Symp. on High-Performance Parallel and Distributed Computing. Washington: ACM Press, 2017. 15−26. [doi: 10.1145/3078597.3078613]

[21] Karmarkar N, Karp RM. An efficient approximation scheme for the one-dimensional bin-packing problem. In: Proc. of the 23rd Annual Symp. on Foundations of Computer Science. Chicago: IEEE, 1982. 312−320. [doi: 10.1109/SFCS.1982.61]

[22] Kafka. http://kafka.apache.org/

[23] Li HY, Ghodsi A, Zaharia M, Shenker S, Stoica I. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In: Proc. of the 2014 ACM Symp. on Cloud Computing. Seattle: ACM Press, 2014. 1−15. [doi: 10.1145/2670979.2670985]

[24] Ding JB, Fu TZJ, Ma RTB, Winslett M, Yang Y, Zhang ZJ, Chao HY. Optimal operator state migration for elastic data stream processing. HAL-INRIA, 2015,22(3):1−8.

[25] TPCH. http://www.tpc.org/tpch

[26] Arasu A, Cheniack M, Maier D, Maskey AS, Ryvkina E, Stonebraker M, Tibbetts R. Linear road: A stream data management benchmark. In: Proc. of the 30th Int'l Conf. on Very Large Data Bases. Toronto: Morgan Kaufmann Publishers, 2004. 480−491. [doi: 10.1016/B978-012088469-8.50044-9]

附中文参考文献:

[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.软件学报,2014,25(4):839−862. http://www.jos.org.cn/1000-9825/4558.htm [doi: 10.13328/j.cnki.jos.004558]

[2] 崔星灿,禹晓辉,刘洋,吕朝阳.分布式流处理技术综.计算机研究与发展,2015,52(2):318−332. [doi: 10.7544/issn1000-1239.2015.20140268]

[3] 王春凯,孟小峰.分布式数据流关系查询技术研究.计算机学报,2016,39(1):80−96. [doi: 10.11897/SP.J.1016.2016.00080]

王春凯(1981－),男,河南开封人,博士,主要研究领域为分布式数据流管理.

孟小峰(1964－),男,博士,教授,博士生导师,CCF会士,主要研究领域为Web数据管理,移动数据管理,XML 数据管理,云数据管理,隐私保护.