Bala-Join：一种用于平衡地理分布式 SQL 数据库中通信与计算的自适应哈希连接

Wenlong Song HuiLi Bingying Zhai网络空间工程学院 计算机科学与技术学院 计算机科学与技术学院 西安电子科技大学 西安电子科技大学 西安电子科技大学西安，中国 西安，中国 西安，中国 song wl@stu.xidian.edu.cn hli@xidian.edu.cn zhaibingying@stu.xidian.edu.cn

Jinxin Yang
网络空间工程学院
西安电子科技大学
西安，中国
yangjx@stu.xidian.edu.cn

Pinghui Wang
网络空间科学与工程学院
西安交通大学
西安，中国
phwang@mail.xjtu.edu.cn

Luming Sun
云析科技有限公司
上海，中国
sunluming@inspur.com

李明山东浪潮数据库技术有限公司济南，中国 liming2017@inspur.com

崔江涛计算机科学与技术学院 西安电子科技大学西安，中国
cuijt@xidian.edu.cn

arXiv:2603.05405v1 [cs.DB] 5 Mar 2026

摘要——**CockroachDB 等无共享架构的地理分布式 SQL 数据库，对于需要数据韧性和数据本地性的企业应用而言日益重要。然而，我们发现客户侧的性能显著下降，尤其是在其部署横跨广域网上的多个数据中心时。经调查，我们确定瓶颈在于分布式哈希连接（Dist-HJ）算法的性能，该算法的表现取决于通信开销与计算负载之间至关重要的平衡。处理来自真实客户工作负载的倾斜数据时，这种平衡会遭到严重破坏，进而导致观察到的性能下降。**

为应对这一挑战，我们提出 Bala-Join，这是一种在 Dist-HJ 执行过程中平衡计算负载与网络负载的自适应解决方案。我们的方法由均衡分区与部分复制（BPPR）算法和分布式在线倾斜连接键检测器组成。前者通过多播机制实现倾斜数据的均衡重分布，以提升计算性能并降低网络开销；后者则为 BPPR 提供实时倾斜连接键信息。此外，我们还引入主动信令与异步拉取（ASAP）机制，使检测器与数据重新分配过程能够以最低开销实现高效的实时同步。实证研究表明，Bala-Join 优于常用的 Dist-HJ 解决方案，吞吐量提升了 25%–61%。

索引词—**分布式哈希连接、无共享、数据倾斜**

I. 引言

无共享架构的 [1] 数据库管理系统，例如 CockroachDB [2],TiDB [3],Spanner [4] 和 OceanBase [5] ，正日益被部署用于支持地理分布式应用，这些应用将数据分散存储在不同的服务器或数据中心。在基于 CockroachDB 的客户部署中，我们与山东浪潮数据库技术有限公司联合组成的数据库研发团队，遇到了一项重大挑战：如何在广域网（WAN）上进行分布式查询处理。具体而言，对于涉及跨城市数据中心的分布式哈希连接（Dist-HJ）的复杂分析型查询，现实世界中的数据倾斜，加上广域网的高延迟和有限带宽，构成了严重的性能瓶颈。

这一挑战是分布式哈希连接执行模型的直接结果。作为实现连接算子的关键方法，HashJoin 依赖哈希表来高效匹配两个关系中的行。较小的关系被构建为哈希表，而较大的关系则通过探测来匹配元组。此时，这两个关系分别称为**构建表** 和**探测表**。通常，分布式哈希连接包括以下三个步骤：

重新分配：构建表和探测表分别位于多个数据 节点上，需要将元组重新分配到多个计算 节点，以执行哈希连接任务。

计算：每个计算 节点对接收到的元组分片执行本地哈希连接计算。

聚合：响应节点在没有后续任务时聚合每个计算节点的结果，例如，另一个哈希连接。否则，计算节点随后

转变为数据节点，为后续操作提供输入。

底层 CockroachDB 基础架构采用的默认哈希分区策略虽然适用于局域网，但往往无法妥善平衡网络通信负载与整体查询响应时间之间的权衡，尤其是在存在数据倾斜的情况下。以下以 TPC-H 基准测试中的一个典型连接查询为例， $customer$ 表和 $orders$表通过客户键（客户键）进行连接。同一客户的订单会分配到同一个计算节点。如果某些客户拥有大量订单，即出现即，数据倾斜，那么相应的计算节点就要承担比其他节点多得多的计算任务。

```
选择计数(*)
从 客户 连接 订单
开启 o_客户键 = c_客户键;
```

图 1 展示了一个实际示例：在 CockroachDB 中执行上述查询，并在连接 $customer$ 表和 $orders$ 表时引入 10% 的数据倾斜。三个计算节点（哈希连接器）的执行时间分别为*72ms*，***9.1s***和 *69ms*。数据倾斜导致各节点间的计算负载分布不均，其中一个节点承担了显著更高的计算开销，进而成为整体查询执行的性能瓶颈。

大量研究 [6]–[10] 已证实，上述问题在实践中普遍存在。已有多项工作 [8]，[10]–[12]致力于解决这一问题。常见思路是分别对倾斜数据和非倾斜数据采用不同的重新分配策略。这依赖于识别倾斜数据，可以通过预先收集的统计信息或在线检测来实现。此外，在高延迟、带宽受限的广域网上跨节点传输海量数据会导致性能下降[13], [14]，，尤其是在采用现有解决方案普遍使用的广播策略时。尽管这些方法在缓解负载不均衡方面取得了可观进展，但整体性能仍受节点间原始数据分布 [8] 或网络条件是否理想 [12]的制约。例如，在浪潮集团这家提供高性能计算基础设施的中国大型企业中，有许多查询需要连接分布于北京和上海（两地距离超过 $1000km$）的表。我们观察到，在这些跨区域连接中，即使将默认的 Dist-HJ 实现替换为一系列旨在实现负载均衡的替代方案 [8], [10]–[12]，网络开销仍然是影响连接性能的关键因素。原因在于，这些解决方案（即PRPD [8] 和 PnR [12]）通常只关注网络开销或负载均衡其中之一，从而限制了整体性能。

本文提出了 Bala-Join，该方法由一种新颖的重新分区策略和一个在线倾斜连接键检测器组成。前者能够适应各种倾斜场景，并始终保持良好的性能；后者则可以在运行时识别中间连接表中的倾斜连接键。

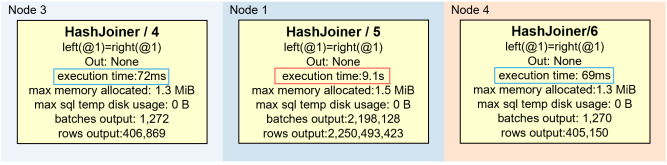


图 1：数据倾斜对分布式哈希连接的影响

总而言之，我们的贡献如下 s.

- 我们为 Dist-HJ 提出了一种新颖的重新分区策略（BPPR），并从理论上保证了负载均衡。通过引入平衡因子和多播机制，该策略能够在保证负载均衡的同时最大限度地降低网络开销（第四节），
- 我们提出了一种分布式倾斜检测机制，同时尽可能减少额外开销。该检测机制还与 BPPR 深度集成。（第五节），
- 实证研究展示了我们的方法在平衡负载和网络开销方面的性能。（第六节）。

II. 相关工作

A. 倾斜数据检测

倾斜数据检测策略通常分为两种范式：静态与动态。静态检测通常在查询规划阶段进行。它依赖于通过分析直方图或采样列值收集的预计算统计信息 [6], [11], [15], [16]。然而，在处理中间结果时，静态方法会失效，因为在执行之前数据分布是未知的。

相比之下，动态检测在运行时进行，用于从分布不可预测的数据流中识别频繁项。该问题在形式上等价于频繁项集挖掘。现有的动态算法可大致分为：1）**基于采样的算法** [17]–[19]，，通过检查数据流的一个子集来估计频率；2）**基于计数器的算法** [7], [18]，，例如 Space Saving，它利用确定性计数器来跟踪项的出现次数；3）**基于草图的算法** [20]，[21]，，例如 Count-Min Sketch [21]，，它采用概率数据结构进行频率估计。

分布式动态检测方面也已有多项研究。Parallel Space Saving [22]和 Double-Anonymous sketch [23] 等方案会聚合局部估计，以形成全局视图。然而，简单的聚合往往需要多次遍历数据，或产生较高的同步开销；Bala-Join 通过其单遍 ASAP 机制解决了这一挑战。

B. 倾斜数据上的连接

分布式连接中的数据倾斜有多种表现形式，包括元组放置倾斜（TPS）、选择率倾斜（SS）、重分布倾斜（RS）和连接结果规模

倾斜（JPS） [6], [24]。虽然哈希分区通常可以缓解 TPS，且 SS 很大程度上取决于查询选择率，但 RS 和 JPS 仍是并行连接处理的主要瓶颈 [8], [10]。现有的缓解策略可分为三种不同的方法：

任务调度方法将倾斜处理建模为传统的任务调度问题。在 Dist-HJ 中，连接表被划分为较小的分区（例如，范围分区[6] 或哈希桶 [15], [25]–[30]），并分配给处理单元。由于任务调度问题是 NP完全问题，此类算法中采用了各种知名的启发式算法（例如，LPT [31]和 MULTIFIT [32]）[6],[15], [25]–[29]。然而，这些方法通常依赖根据元组计数得出的静态成本估算，往往无法考虑动态的运行时波动。

运行时自适应调度方法会在执行过程中动态调整工作负载分布，以处理负载不均衡 [6], [26], [30], [33]–[36]。这些方法不预先预测执行时间，而是通过将任务从过载节点转移到利用率较低的节点，实现实时自适应。常见的调度策略包括虚拟处理器调度（VPS） [6], 通过共享虚拟内存进行工作窃取 [33], ，以及由中央处理器管理的多阶段调度 [35]。为指导这些决策，此类方法会采用各种成本指标，例如估算的连接成本 [6], 输出和工作函数 [26], ，或实时处理速度 [34]。此外，有些解决方案专为异构配置 [30] 或流处理环境设计，例如 FastJoin [36]。尽管这些方法具有自适应能力，但往往会产生显著的状态迁移开销。

倾斜感知分区方法与我们的工作最为相关，通过优化物理数据放置来缓解计算不均衡。PRPD [8]采用一种策略，使倾斜元组保留在本地，同时广播对侧表中对应的分区，这一策略依赖于随机数据分布的假设。Flow-Join [10]进一步应用对称分片复制（SFR）策略来处理双边倾斜，并证明了其在高带宽环境中的有效性。其他研究则专注于特定的优化目标：Track Join [11] 通过优化唯一值的传输计划来减少流量，而 FGSD [37] 则专门为 MapReduce 工作流提供细粒度负载均衡。最近，PnR [12] 改变了优化范式，从最小化网络传输转向实现近乎完美的计算均衡，在高速网络中优先考虑 CPU 效率。尽管这些策略已取得显著进展，但它们通常会以网络效率换取负载均衡（例如，PnR），或依赖于均匀倾斜分布的假设（例如，PRPD）。相比之下，Bala-Join 旨在将负载均衡与原始数据分布解耦，同时最大限度地减少与广播相关的网络开销。

III. 预备知识与研究动机

A. 无共享数据库管理系统

无共享架构广泛应用于分布式数据库管理系统中，其中每个节点都独立运行，拥有独立的中央处理器、内存和存储。在这些数据库管理系统中，Dist-HJ 以并行方式执行。由于数据分片位于不同节点上，具有相同连接键的元组需要重新分配到同一个计算节点。随后，每个计算节点执行本地哈希连接，结果再由响应节点进行聚合。尽管哈希分区是实践中最常用的重新分区策略，但在数据倾斜情况下可能导致严重的不均衡。在这种情况下，新添加的节点无法共享热点节点上的倾斜数据，从而限制可扩展性。例如，作为具有代表性的无共享数据库管理系统，CockroachDB [2] 将数据组织为范围，并在整个集群中进行分布和复制。在执行 Dist-HJ 时，CockroachDB 会动态选择范围副本来生成物理执行计划。因此，给定连接查询所涉及的节点并非预先确定，并且每个节点贡献的数据量可能存在显著差异。此外，CockroachDB 中默认的 Dist-HJ 将哈希分区作为唯一的重新分区策略。因此，当发生数据倾斜时，可能会导致整个集群的负载不均衡。

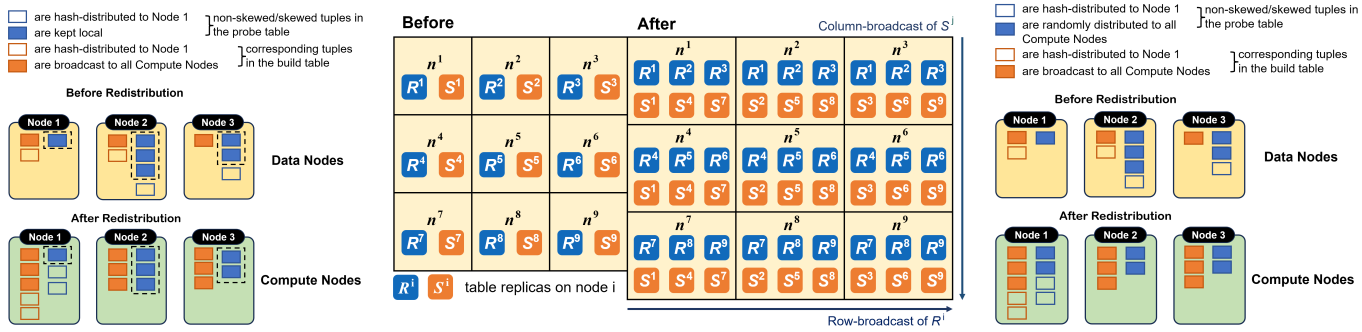
B. 分布式哈希连接

本节探讨分布式哈希连接的实现，重点介绍若干在不同数据分布场景下优化性能的关键算法。

Grace哈希连接：作为无共享架构的标准实现，GraHJ [38]使用一致的哈希函数，将构建表 R 和探测表 S 分区到各个节点上。这种重新分配确保具有相同连接键的元组被放置在同一处理单元上。因此，全局连接操作被分解为一组相互独立、并行执行的本地哈希连接，其并集构成最终结果。

PRPD：在数据倾斜情况下，GraHJ 的效率可能会下降。PRPD 分别处理倾斜元组和非倾斜元组：前者保留在本地，后者则采用哈希分区策略。构建表中与倾斜连接键对应的元组会广播到集群中的所有节点。PRPD 旨在最大限度地减少网络开销并实现均衡负载。其最佳性能建立在倾斜元组均匀分布于各节点的假设之上。然而，由于倾斜元组的分布无法预测，因此难以持续实现预期的良好性能。

SFR：SFR 是 Flow-Join [10] 所采用的重新分区策略。对于仅在构建表或探测表中发生倾斜的元组，Flow-Join 采用 PRPD 策略。对于在两张表中均发生倾斜的元组，Flow-Join 采用对称分片复制（SFR）策略，即将集群中的节点在逻辑上排列成一个矩阵。来自构建表的元组在该矩阵中



(a) PRPD 算法　(b) SFR 算法　(c) PnR 算法

图 2：重新分配策略概览

沿水平方向进行复制，而来自探测表的相应元组则沿垂直方向复制到各节点。

PnR：PnR [12] 的行为与 Flow-Join 类似，但它会选择在集群中的所有节点上随机或以轮询方式分发倾斜元组。探测表中的对应元组会广播到整个集群。

图 2 概述了针对数据倾斜设计的重新分配策略。接下来，我们将讨论这些方法的适用场景和局限性，从而阐明我们的目标。

C. 观察与目标

PRPD 和 SFR 专注于优化网络开销，而 PnR 则强调负载均衡。这两组方法在特定场景下各有优势，但在更为普遍的场景中可能会受到局限。

由于 PRPD 将倾斜元组保留在本地，当倾斜元组的原始分布在各节点之间不均匀时，某些节点的负载仍会高于其他节点。同样，在 SFR 中，当矩阵中的某一行或某一列包含的倾斜元组远多于其他行或列时，该行或该列上的节点将会过载。也就是说，PRPD 和 SFR 不适用于倾斜元组的原始分布极度不均衡的场景。

PnR 以巨大的网络开销为代价，在各种场景下实现均衡重分布。在重新分布阶段，包含更多倾斜元组的节点会承担更重的负载。此外，PnR 也存在与 PRPD 相同的问题，尤其是在两张表的规模几乎相同的情况下。

PRPD、Flow-Join 和 PnR 在计算阶段或重新分布阶段都存在局限性。总而言之，以往的策略面临以下挑战：

挑战。 ❶ 依赖原始分布：任何将倾斜元组保留在本地的策略，当倾斜元组在各节点上的原始分布不均匀时，都会导致性能下降。

挑战。 ❷ 统计依赖：以往的方法要求识别倾斜元组。此外，Flow-Join 和 PnR 还必须确定构建表和探测表中的倾斜元组数量。这类统计数据在流式场景或中间结果中无法获取，因此这些方法不适用。

挑战。 ❸ 非自适应策略：对于给定的元组，一旦获取必要的统计信息，重新分配就会按照特定策略进行。因此，无法在集群内实现动态负载均衡。在流式场景中，最初被判定为非倾斜的元组，随着数据流推进，最终可能变为倾斜元组。因此，分布式连接的正确性面临挑战。

为应对上述挑战，我们提出了 Bala-Join，它能够动态检测并处理倾斜元组。Bala-Join 在元组到达时立即进行分发，无需等待完整的统计信息。Bala-Join 提出了一种新颖的倾斜感知分发策略 BPPR，可确保集群内的连接负载动态均衡，且不受原始数据分布的影响，从而缓解**挑战。** ❶。两张表中具有相同连接键的元组，无论是否被判定为倾斜元组，始终都会分发到同一组节点，从而解决**挑战。** ❸中提出的动态适配一致性问题。最后，将 BPPR 与分布式检测器和 ASAP 机制结合，可实现实时检测与分发，大幅提升分布式哈希连接的性能，并解决**挑战。** ❷。

IV. BALA-JOIN

A. 系统概述

Bala-Join 面向地理分布式数据库中的分布式哈希连接，重点关注两项主要成本：跨区域通信开销，以及负载不均衡导致的计算成本。

Bala-Join 由一种新颖的重新分区策略 BPPR 和在线倾斜连接键检测器组成。构建表和探测表以数据分片或数据流的形式，在各数据节点间参与连接。探测元组到达后，部署在各节点上的分布式检测器会检查每个元组，判断其是否倾斜，然后根据 BPPR 对其进行重新分配。

为确保元组能够正确连接，当一个表中的元组被分发到某个节点时，另一个表中所有具有相同连接键的元组也必须分发到该节点。由于探测表通常大得多

表 I：符号说明

符号	说明
R, S	构建表和探测表
R^i, S^i	最初位于 R 和 S 的本地数据分片 节点 i（数据流）
S_{skew}, S_{non}	S 中的倾斜元组和非倾斜元组集合
R_{skew}, R_{non}	键与以下对象中的键匹配的构建元组集合 S_{skew} 和 S_{non}
S^i_{skew}, S^i_{non}	节点上的偏斜/非偏斜探测元组的本地子集 节点 i
R^i_{skew}, R^i_{non}	节点 i 上的本地构建元组子集
$U(x)$	候选目标计算节点集合，用于 倾斜键 x
$C(t)$	将倾斜元组 t 路由到某个节点的选择函数 U（$t.key$）中的节点
$\widehat{S}^j_{skew}$	收到的倾斜探测元组的总量 计算节点 j
$\widehat{S}^{ij}_{skew}$	来自源节点 i 的倾斜元组总量 分发到目标节点 j
B	表示负载差异的平衡因子（公式 1）
ϵ	用于衡量负载不均衡的用户定义阈值（$B \leq \epsilon$）

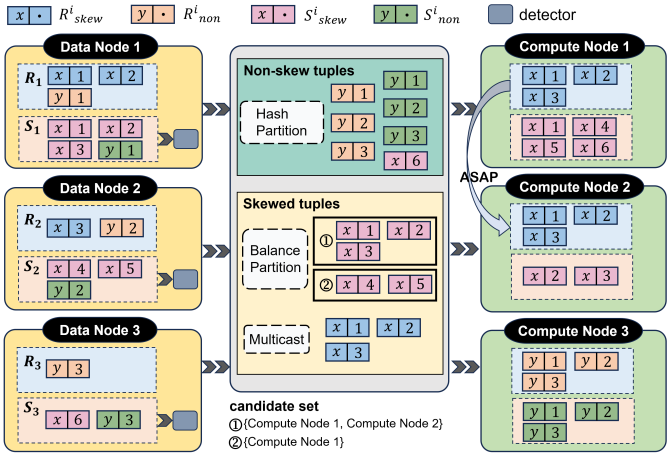


图3：Bala-Join 概览

于构建表，若要平衡构建表中的倾斜元组，就必须对海量探测表元组进行多播或广播。此外，我们关注的是涉及中间结果或数据流的场景，在这些场景中没有可用的统计信息来❶比较倾斜构建元组与对应探测元组的数量❷并决定如何处理它们（PnR 采用的策略）。因此，我们只研究探测侧。

在 BPPR 中，每个数据节点都会为倾斜元组的每个连接键维护一个可扩展的目标节点集。新到达的倾斜元组会被分发到集合中的适当节点，以满足预定义的平衡因子；相关细节将在第 IV-C 节中说明。当现有节点均不适用时，就会扩展节点集合。随后，构建表中具有相同连接键的元组很快就会被多播到该节点集合。

示例：图3展示了 Bala-Join 的概览。数据节点持续接收构建元组流和探测元组流，每个节点上的检测器会动态地实时识别倾斜探测元组。我们使用下标 p 和 b 分别表示来自探测表和构建表的元组。非倾斜元组（例如，$\langle x、6\rangle_p$、$\langle y、1\rangle_p \sim \langle y、3\rangle_p$）会通过哈希分区分发到相应的计算节点（例如，计算节点 1 和 3）；默认情况下，构建表也采用这一策略。相较之下，倾斜探测元组会根据其连接键分发到特定的目标节点集；例如，来自数据节点1、键为 x 的元组（即 $\langle x、1\rangle_p \sim \langle x、3\rangle_p$）会分发到 {计算节点 1、计算节点 2}；而来自数据节点 2 的元组（即 $\langle x、4\rangle_p$、$\langle x、5\rangle_p$）则会发送到 {计算节点 1}。为确保正确性，相应的构建元组（即 $\langle x、1\rangle_b \sim \langle x、3\rangle_b$）会多播到所有这些目的节点（即并集 {计算节点 1、计算节点2}）。由于构建元组本身并不知道探测元组的目的节点，因此这种匹配由第五节所述的 ASAP 机制来实现。此外，即使同一个连接键（例如，x）在不同节点上被不一致地识别为倾斜或非倾斜，目标集更新算法也能确保所有相关元组始终被分发到同一组一致的节点。

B. 符号说明

通常，我们考虑在一个由 n 个节点组成的无共享集群上执行分布式哈希连接操作$R \bowtie S$。其中，构建表 R 和探测表 S 最初被划分为分片R^i 和 S^i，位于节点 i（$0 \leq i < n$）。这种抽象将输入视为分布式数据流，无论它们是原始表还是中间结果。

如果探测表 S 中某个元组的连接键频率超过预定义阈值 θ，我们就将其识别为倾斜元组。基于此，S 被划分为倾斜元组集合（记为 S_{skew}）和非倾斜元组集合 S_{non}。相应地，构建表 R 在逻辑上被划分为R_{skew} 和 R_{non}，其中 R_{skew} 由连接键出现在 S_{skew} 中的元组组成（无论这些连接键在 R 中的固有频率如何）。在每个节点 i 上，本地分片进一步划分为四个不同的子集：S^i_{skew}、S^i_{non}、R^i_{skew}和 R^i_{non}。

C. 平衡分区与部分复制　　n

BPPR 工作流。

BPPR 遵循 Dist-HJ 的通用处理工作流，但增加了一个元组级分区阶段，旨在将倾斜元组进一步细分为粒度更细的分区，以改善负载均衡。在实时倾斜检测器（详见第五节）的辅助下，输入流会被动态分区并处理。对于形式为 $R \underset{R.a=S.b}{\bowtie} S$ 的连接操作，通用流程如下。

步骤 1：重新分配。根据上述内容，R^i 和 S^i 分别被划分为四部分：S^i_{non}、S^i_{skew}、R^i_{non}和 R^i_{skew}。这些部分随后按如下方式进一步分区并重新分配：

如第四节所定义，本地分片 S^i 和 R^i 在逻辑上被划分为四个子集，这些子集在图3的数据节点中以不同颜色区分。随后，这些子集按如下方式进一步分区并重新分配：

- S^i_{non}：此子集中的元组经过哈希分区并分发，在 n 个节点之间实现近乎均匀的分布。

  示例：$\langle x,\ 6\rangle_p$、$\langle y,\ 1\rangle_p \sim \langle y,\ 3\rangle_p$ 分别分发到计算节点 1 和计算节点 2。请注意，尽管 x 是 S 中的倾斜键，但由于数据节点3上的检测器仅监控本地流 S_3，因此将其识别为非倾斜键。即使在这种情况下，BPPR 和 ASAP 机制也能保证连接结果的正确性。

- S^i_{skew}：每个元组 $t \in S^i_{skew}$ 都会被路由到一个目标节点，该节点由候选集 $U(x)$ 中的选择函数 $C(t)$ 确定。为量化平衡程度，我们将平衡因子 B定义为：

$$B = (\max_j |\widehat{S}^j_{skew}| - \min_j |\widehat{S}^j_{skew}|)/\max_j |\widehat{S}^j_{skew}| \quad (1)$$

  其中，$\widehat{S}^j_{skew}$ 表示计算 节点 j 接收的倾斜元组量。

  示例：如图3所示，计算节点 1 接收的倾斜元组数量最多（4 个），而计算节点 3 接收的数量为零。因此，全局平衡因子计算为 $B = (4-0)/4 = 1$。请注意，尽管图中仅显示了 x 这一倾斜键，但 $\widehat{S}^j_{skew}$ 表示所有倾斜键，而不是某个特定的键，例如 x。

  自然地，倾斜元组的均衡重分布应满足 $B \le \epsilon$，其中 ϵ 是预定义的阈值。对于每个到达的倾斜元组，$C(t)$ 会尝试从节点集合中选择一个目标节点，并测试其是否足够均衡。如果 $U(x)$ 中没有节点满足要求，则会扩展 $U(x)$，以纳入其他节点供选择（稍后将在算法 2 中讨论）。

  示例：$\langle x,\ 1\rangle_p \sim \langle x,\ 3\rangle_p$和 $\langle y,\ 4\rangle_p \sim \langle y,\ 5\rangle_p$被分发到集合 $U(x)$ ={计算节点 1、计算节点 2}。请注意，$U(x)$ 实际上是由数据节点 1 和 2 维护的$U^1(x)$ 与 $U^2(x)$ 的并集，第四 IV-C 节将详细介绍这一机制。

- R^i_{skew}：为确保正确性，R_{skew} 中的每个元组都必须将连接键 $\boldsymbol{x}$ 多播至指定的节点
  由 $U(x)$，我们将其称为部分复制。
  示例：$\langle x, 1\rangle_b \sim \langle x, 3\rangle_b$ 多播到 {计算节点 1、计算节点 2}，因为 $\langle x,\ 1\rangle_p \sim \langle x, 3\rangle_p$ are 分发到该集合 .

- R^i_{non}：与 S^i_{non} 类似，这些元组通过以下方式分发标准哈希分区。
  示例：$\langle y, 1\rangle_b \sim \langle y, 3\rangle_b$ 被分发到计算节点
  节点 3 与 $\langle y, 1\rangle$ 匹配 $_p \sim \langle y, 3\rangle_p$.

步骤 2：计算。每个计算节点处理从重新分发步骤收到的元组（例如，计算节点 2 处理倾斜探测元组 $\langle x, 2\rangle_p$和 $\langle x, 3\rangle_p$ 与多播构建元组 $\langle x,\ 1\rangle_b \sim \langle x,\ 3\rangle_b$) 进行连接。节点 j在倾斜流和非倾斜流上分别执行两个独立的本地哈希连接：
$R^j_{non} \bowtie S^j_{non}$ 和 $R^j_{skew} \bowtie S^j_{skew}$。

步骤 3：合并。计算步骤完成后，每个计算节点都会生成一组本地部分结果。形式上，全局结果由汇总所有 n 个计算节点的输出得出。由于探测元组被不相交地分区（通过哈希或 $C(t)$ 函数），全局连接结果就是这些本地结果的并集：

$$R \bowtie S = \bigcup_{j=0}^{n-1} \Big((R^j_{non} \bowtie S^j_{non}) \cup (R^j_{skew} \bowtie S^j_{skew}) \Big) \quad (2)$$

其中，j 表示计算节点的索引。

与要求每个倾斜键在所有节点上均匀分布的策略（例如PRPD 和 PnR）不同，BPPR 在集群级别优化负载均衡。例如，PnR 将键为 x 的元组分散到所有 n 个节点，以尽可能减少本地倾斜，但会产生较高的多播开销。相反，BPPR 将 x的分发限制在最小子集 $U(x)$ 中。因此，尽管特定键 x的负载在整个集群中可能并非完全均匀，但所有倾斜元组的总负载是均衡的（即满足 $B \le \epsilon$）。这种设计权衡在维持计算均衡的同时显著降低了网络成本；我们在第六节的实证评估证实了这一点。

平衡分区算法

直观而言，根据上述策略，对于连接键为 $\boldsymbol{x}$ 的倾斜元组，每个数据节点都应将它们发送到同一组节点，从而减少多播构建表造成的网络开销。然而，在分布式环境中，实现这种共识需要大量网络通信。具体来说，对于每个到达的元组，系统都必须同步全局平衡状态，以决定是将其路由到当前的 $U(x)$，还是广播集合扩展。在处理中间结果或数据流时，这种巨大的同步开销难以承受。为此，我们提出了基于序列生成器的 平衡分区算法，该算法无需协调即可达成共识，同时确保集群中的负载均衡。

在没有序列生成器的情况下，每个数据节点都会针对每个倾斜连接键 $\boldsymbol{x}$ 维护一个分歧的本地集合 $U^i(x)$。这带来了两个关键要求：

(1) **正确性**：有效的全局目标集必须是所有本地集合的并集，即 $U(x) = \bigcup_{i=0}^{n-1} U^i(x)$，以确保来自 R_{skew} 的构建元组能够到达所有潜在目标节点；

(2) **效率**：必须将基数 $|U(x)|$ 降至最低，以减少多播开销。

需求（1）由 ASAP 机制（详见第五节 V-B）保证。需求（2）促使我们设计了下文所述的确定性序列生成器。为确保满足（2），对于 $i = 0$、..... $n-1$，集合 $U^i(x)$ 必须以一致的方式构建和扩展，从而确保最大集合包含其他所有集合。这样可以最大限度地减少多播期间的网络开销。为此，我们提出了节点集更新算法，即算法1，

该算法采用了确定性序列生成器。对于倾斜连接键 $\boldsymbol{x}$，过程 GENSEQ 无需通信，即可在所有数据节点上生成一致的候选节点序列。该过程基于哈希的迭代机制，具体而言，它使用 $hash(\boldsymbol{x}+epoch) \bmod \boldsymbol{n}$ 计算候选节点索引，并通过迭代递增纪元来查找不重复的节点。值得注意的是，生成的第一个元素（其中 $epoch = 0$）始终为 $hash(\boldsymbol{x}) \bmod \boldsymbol{n}$，这对应于默认哈希分布下该元组的目标节点。这意味着，当 $\boldsymbol{x}$ 最初被识别为非倾斜、之后又被识别为倾斜时，其目标节点始终保持为$U(x)$。这解决了**挑战 ❸** 中提出的问题。

GENSEQ 确保为同一倾斜连接键在各个节点上生成的序列保持一致。基于此，每个数据节点都可以按照相同的序列独立扩展 $U^i(x)$。算法1中的过程 UPDATEU 展示了这一更新过程。每当节点集合需要扩展时（并且只要 $|U^i(x)| < n$），它就会调用 GENSEQ 生成序列中的下一个确定性节点，并将其添加到 $U^i(x)$。这确保了多播目标集在整个集群中得到一致扩展。

鉴于每个数据节点已就 $U^i(x)$ 的扩展达成共识，我们接下来追求集群内的负载均衡。回顾式 (1)， S^i_{skew} 应以均衡方式进行分区，使得传输到每个计算节点的聚合数据量保持均衡。我们将 $\widehat{S}^{ij}_{skew}$ 记为从数据节点 i发送到计算节点 j的倾斜元组分区。为确保分区后各分区保持均衡，我们定义局部均衡因子 B^i:

$$B^i = (\underset{j}{max}\,|\widehat{S}^{ij}_{skew}| - \underset{j}{min}\,|\widehat{S}^{ij}_{skew}|)/\underset{j}{max}\,|\widehat{S}^{ij}_{skew}| \quad (3)$$

平衡因子 B^i 衡量活跃目标节点之间聚合数据量的均衡程度。较小的B^i 表示节点 i 将其倾斜工作负载均匀地分配到目标计算节点。请注意，该指标考虑了所有倾斜键的总负载。例如，如果节点i 上键为 k_1 和 k_2 的倾斜元组分别分配到 {计算节点 1、计算节点 2} 和{计算节点 3}，且每个节点上的数据量相近，那么 max = min，因此 $B^i = 0$。在这种情况下，系统是均衡的，无需扩展候选集。我们设置阈值 ϵ，使得 $B^i \le \epsilon$。

定理1。 在均衡分区的情况下，如果平衡因子 $B^i \le \epsilon$，则重新分配后计算节点之间的整体平衡因子 B 也将满足 $B \le \epsilon$。

证明。 $\widehat{S}^{ij}_{skew}$ 与 $\widehat{S}^{j}_{skew}$ 之间的关系为:

$$|\widehat{S}^{j}_{skew}| = \sum_{i=0}^{n-1} |\widehat{S}^{ij}_{skew}| \quad (4)$$

令 $B_c = \epsilon$，将式 (3) 转换为以下不等式:

$$\underset{j}{max}\,|\widehat{S}^{ij}_{skew}| - \underset{j}{min}\,|\widehat{S}^{ij}_{skew}| \le B_c \cdot \underset{j}{max}\,|\widehat{S}^{ij}_{skew}| \quad (5)$$

对该不等式进行求和变换:

$$\sum_{i=0}^{n-1}(\underset{j}{max}\,|\widehat{S}^{ij}_{skew}| - \underset{j}{min}\,|\widehat{S}^{ij}_{skew}|) \le \sum_{i=0}^{n-1}(B_c \cdot \underset{j}{max}\,|\widehat{S}^{ij}_{skew}|) \quad (6)$$

重新排列求和式:

$$\underset{j}{max}(\sum_{i=0}^{n-1}|\widehat{S}^{ij}_{skew}|) - \underset{j}{min}(\sum_{i=0}^{n-1}|\widehat{S}^{ij}_{skew}|) \le B_c \cdot \underset{j}{max}(\sum_{i=0}^{n-1}|\widehat{S}^{ij}_{skew}|) \quad (7)$$

将两边同时除以 $\underset{j}{max}(\sum_{i=0}^{n-1}|\widehat{S}^{ij}_{skew}|)$:

$$\frac{\underset{j}{max}(\sum_{i=0}^{n-1}|\widehat{S}^{ij}_{skew}|) - \underset{j}{min}(\sum_{i=0}^{n-1}|\widehat{S}^{ij}_{skew}|)}{\underset{j}{max}(\sum_{i=0}^{n-1}|\widehat{S}^{ij}_{skew}|)} \le B_c \quad (8)$$

利用公式（4）将其代入不等式:

$$\frac{\underset{j}{max}\,|\widehat{S}^{j}| - \underset{j}{min}\,|\widehat{S}^{j}|}{\underset{j}{max}\,|\widehat{S}^{j}|} \le B_c \quad (9)$$

将其与公式（1）结合，可得:

$$B \le B_c \quad (10)$$

□

因此，确保满足全局要求 $B \le \epsilon$可归结为在局部满足 $B^i \le \epsilon$ 。为此，我们提出了平衡分区算法（算法 2）。对于每个到达的倾斜元组 t （其连接键为 $\boldsymbol{x}$），该算法执行以下步骤:

1) 暂定分配： 算法暂定将t 分配给 $C(t')$，其中 t' 是具有相同键 $\boldsymbol{x}$的前一个元组，从而保持分配局部性。**2) 平衡验证：** 随后计算局部均衡因子 B^i。如果满足条件 $B^i \le \epsilon$，则确认该分配。**3)重新分配（贪心调整）：** 如果不满足 $B^i \le \epsilon$ ，算法会尝试将 t 重新分配给 $U(x)$中当前负载最小的节点。**4) 集合扩展：** 如果分配给负载最小的节点仍无法满足 $B^i \le \epsilon$ （或者 $C(t')$已经是负载最小的节点），则表明$U(x)$的当前容量不足以在不破坏整体平衡的情况下容纳负载。只有在这种情况下，$U(x)$才会根据算法1进行扩展。

示例： 假设图3中的状态代表当前集群状态，其中数据节点 2 已将两个倾斜元组（$\langle x、4\rangle_p$、$\langle x、5\rangle_p$）路由到计算节点 1。当前候选集为 $U^2(x) = \{$计算节点 1$\}$。假设一个新的倾斜元组 $\langle x、7\rangle_p$ 到达数据节点 2。检测器将其识别为倾斜元组。算法暂定将其分配给计算节点 1（沿用之前的放置方式）。在此暂定分配下，计算节点 1 将持有 3 个倾斜元组（ 2 个现有元组 $+1$ 新元组），而计算节点 2 和 3 持有 0 个。因此，平衡因子计算为

$B = (3-0)/3 = 1$。由于 $B > \epsilon$，集合 $U^2(x)$ 扩展为 {计算节点 1、计算节点 2}。最后，$\langle x、7\rangle_p$ 被分配到计算节点 2，以缓解负载不均衡。

算法 2 展示了所提出策略的详细执行流程。值得注意的是，平衡因子的计算在实践中可以优化为常数时间复杂度（$O(1)$）。通过维护当前的最大和最小分区大小，每次进行元组分配时都可以增量更新平衡因子，从而避免重复进行线性扫描。

D. 算法复杂度

BPPR 算法由平衡分区算法和多播机制组成。前者将探测表中的倾斜元组分发到不同节点，后者则将构建表中具有相同连接键的元组复制到相应节点。在一个包含 n 个节点的集群中，节点 i 上的倾斜元组表示为 S^i_{skew}，其元组数量表示为 $m_p = |S^i_{skew}|$。来自 R_{Sskew} 的元组数量表示为 m_b。定义 $D(S^i_{skew})$ 为去重操作，其中去重后每个节点上的不同倾斜连接键数量表示为 $k = |D(S^i_{skew})|$。

在平衡分区算法中，倾斜元组以时间复杂度 $O(m_p)$进行遍历。平衡因子的优化计算具有常数时间复杂度$O(1)$，与扩展节点集合相同。因此，平衡分区算法的时间复杂度为 $O(m_p)$。算法平衡分区 为每个倾斜连接键 $\boldsymbol{x}$记录 m 元组和$U^i(x)$，其大小至多为 n 、至少为1。每个倾斜连接键的序列以大小 n保存。因此，平衡分区 算法的空间复杂度为 $O(m_p + k \cdot n)$。由于元组在实践中被直接分发，空间复杂度为$O(k \cdot n)$。

对于多播，R_{Sskew}中具有连接键 $\boldsymbol{x}$ 的每个元组被复制到 $U^i(x)$。因此，时间复杂度至少为$O(m_b)$，至多为 $O(m_b \cdot n)$。空间复杂度为 $O(1)$。

对于每个数据节点，时间和空间复杂度列于表 II。

总之，平衡分区算法通过以细粒度对倾斜元组进行分区，显著改善了集群范围内的负载均衡。同时，候选集合的增量扩展策略最大限度地降低了网络开销。至关重要的是，该方法的有效性不受特定数据分布的影响，能够动态适配倾斜工作负载，从而解决第 III-C 节所列的**挑战❶**和**挑战❸**。

然而，尽管算法1确保了本地视图之间的一致性，要正确多播构建元组，就必须将这些本地集合 $U^i(x)$ 在逻辑上聚合为一个全局目标集 $U(x)$。在下一节中，我们将介绍分布式检测器和 *ASAP*机制。这些组件旨在处理流式倾斜检测，并在无需大量同步的情况下，将构建元组精确地多播到 $U(x)$。

算法1 节点集更新算法

输入： 连接键 $\boldsymbol{x}$，序列 $\boldsymbol{seq}$ for $\boldsymbol{x}$，节点数量 $\boldsymbol{n}$
输出： 已更新的 $\boldsymbol{U^i(x)}$

过程 GENSEQ($\boldsymbol{x}, \boldsymbol{seq}, \boldsymbol{n}$)
  如果 $\boldsymbol{seq}.size < \boldsymbol{n}$ **则**
    $epoch \leftarrow \boldsymbol{seq}.size$
    $candidate \leftarrow hash(\boldsymbol{x} + epoch)$ mod $\boldsymbol{n}$
    当 $candidate \in \boldsymbol{seq}$ **时执行**
      $epoch \leftarrow epoch + 1$
      $candidate \leftarrow hash(\boldsymbol{x} + epoch)$ 模 $\boldsymbol{n}$
    循环结束
    $\boldsymbol{seq}.add(candidate)$
  条件判断结束
  返回 $\boldsymbol{seq}$
结束过程
过程 UPDATEU($\boldsymbol{x}$, $\boldsymbol{U^i(x)}$ $\boldsymbol{seq}$)
  $size \leftarrow \boldsymbol{U^i(x)}.size$
  $\boldsymbol{seq} \leftarrow$ GENSEQ($\boldsymbol{x}, \boldsymbol{seq}, \boldsymbol{n}$)
  如果 $size == \boldsymbol{seq}.size$ **则**
    返回 $\boldsymbol{U^i(x)}$
  结束 if
  $\boldsymbol{U^i(x)}.add(\boldsymbol{seq}[size])$
  返回 $\boldsymbol{U^i(x)}$
结束过程

表 II：BPPR 的时间和空间复杂度

	平衡分区	多播	BPPR
time	$O(m_p)$	$O(m_b \cdot n)$	$O(\max(m_p, m_b \cdot n))$
空间	$O(k \cdot n)$	$O(1)$	$O(k \cdot n)$

V. 即时分布式检测器

基于第 IV 节确立的分区策略，本节将详细介绍在分布式流处理环境中实现 Bala-Join 所需的运行时组件。我们将介绍两个核心模块：**分布式倾斜检测器**，它无需全局同步即可实时识别倾斜键；以及 **ASAP** 机制，它实现了将构建元组接收方驱动地同步到动态目标集 $U(x)$。

A. BPPR 的分布式倾斜检测器

由于无法获取数据流或中间结果的全局统计信息，我们采用一种基于 *SpaceSaving* 算法的局部检测策略 [7]。不同于需要线性空间的精确计数方法，*Space Saving*算法利用有界内存高效识别重频项（即 使用 $O(k)$ 个计数器跟踪 k 个元素）。Space Saving 算法面临的主要挑战在于高效更新元素，以及移除与最小计数对应的元素。遵循 [10],的建议，我们采用基于哈希表和有序数组的 Space Saving 算法作为单节点环境中的检测器，从而最大限度地降低时间和空间开销。

一种直接的 [22], [39] 分布式检测方法是在每个节点上进行局部计数，然后汇总

算法 2 平衡分区算法

输入： 倾斜元组 S^i_{skew}，阈值 ϵ，节点数量 n
输出： 平衡子分区集合 $partition$

过程CALCBALANCEFACTOR($partition$)
  $maxVal \leftarrow 0$
  $minVal \leftarrow \infty$
  对所有 $sub \in partition$ **执行**
    $size \leftarrow sub.size$
    如果 $size > maxVal$ **则** $maxVal \leftarrow size$
    结束条件判断
    如果 $size < minVal$ **则** $minVal \leftarrow size$
    结束条件判断
  结束循环
  如果 $maxVal == 0$ **则**
    返回 0
  结束如果
  返回 $(maxVal - minVal)/maxVal$
结束过程
过程 BALANCEDPARTITION(S^i_{skew}, ϵ, n)
  $partition \leftarrow$ 初始化 n 空列表
  $U^i, Seq \leftarrow$ 初始化空映射
  对于所有 $t \in S^i_{skew}$ **do**
    $x \leftarrow t.b$
    $ID \leftarrow 0$ ▷节点索引
    如果 $Seq_x == []$ **则**
      UPDATEU(x, $U^i(x)$ Seq_x)
      $ID \leftarrow Seq_x[0]$
    else
      $lastID \leftarrow$ 由 x 最后选择的子分区ID
      $partition[lastID].add(t)$ ▷ 试探性添加
      $balanceFactor \leftarrow$ 计算余额因子-TOR($partition$)
      $partition[lastID].delete(t)$ ▷ 回滚
      如果 $balanceFactor > \epsilon$ **则**
        $ID \leftarrow U^i(x)$ 中的最小规模子分区
        如果 $ID == lastID$ **则**
          UPDATEU($x, U^i(x)$ Seq_x)
          $ID \leftarrow Seq_x.last()$
        结束如果
      else
        $ID \leftarrow lastID$
      结束如果
    结束 if
    $partition[ID].add(t)$ ▷ 最终分配
  结束 for
  返回 $partition$
结束 procedure

列表以达成全局共识，再重新分发元组。然而，这种方法存在三个关键问题：

双重遍历 S：每个数据节点都必须遍历表 S^i 两次（一次用于本地检测，另一次用于重新分配）。在流式场景中，这种缓冲会导致显著的内存和 I/O 开销。

独立单机检测：节点间处理速度的差异会导致同步屏障。较快的节点在等待较慢的节点完成检测时会陷入空闲，从而降低整体吞吐量。

共识的通信开销：要获得全局认可的列表，数据节点之间不可避免地需要承担高昂的通信开销。

为克服这些限制，Bala-Join 放弃全局共识，转而采用与 BPPR 算法紧密集成的完全局部检测策略。系统将检测器视为守门人，从而在遍历数据流的**单次扫描**中完成检测和重新分配。当探测元组 t （其连接键为 x）到达时，系统会执行以下分类逻辑：

- **非倾斜流（默认）：** 如果 t 未在本地被识别为倾斜流，则沿用标准哈希分区路径，并直接路由至其哈希分布节点（计算方式为 q =hash(x)）。
- **倾斜流（BPPR）：** 如果 t 被识别为倾斜流，则绕过默认哈希路径，并交由平衡分区算法处理。该算法将 t 路由至动态候选集 $U^i(x)$ 中的一个目标节点，以维持平衡因子 $B^i \leq \epsilon$。

至关重要的是，该架构将检测准确性与路由一致性解耦。由于本地数据存在差异，连接键 x 可能在数据节点1上被识别为倾斜 （路由至 $U^i(x)$），但在数据节点3上被识别为非倾斜 （路由至哈希分布节点 q）。我们系统的稳健性依赖于序列生成器（算法1），它确保哈希分布节点 q 严格包含在候选集 $U^i(x)$（即 $q \in U^i(x)$）中。该保证确保无论某个键在本地被视为倾斜还是非倾斜，驻留在节点 q 上的构建元组始终可供整个集合 $U(x)$访问，从而无需同步延迟即可实现独立运行。

这种集成机制自然意味着一种**延迟构建分区** 策略。由于倾斜是动态处理的，Bala-Join 最初将构建表 R 中的所有元组视为非倾斜元组，并根据 hash(x)定义的默认哈希节点对其进行哈希分发。仅在需要时，才会将 R 分区（复制）到扩展集合 $U^i(x)$中，具体通过下一小节所述的机制实现。

*B. ASAP*机制

为了保证分布式环境中哈希连接的正确性，集群中的数据节点必须达成两种形式的共识：

共识 ***1***：关于倾斜键的共识，使数据节点能够以均衡的方式处理来自 S 的倾斜元组；
共识 ***2***：关于目标集合 $U(x)$ 的共识，用于多播相应的构建元组。

我们通过**主动信令与异步拉取 (ASAP)**机制达成共识。构建元组到达时始终按照哈希进行分布。探测元组到达后，非倾斜元组按照哈希分布，而

倾斜元组则通过 BPPR 路由。相关节点在收到倾斜元组时会收到通知。随后，它们通过 ASAP 异步拉取匹配的构建元组，从而无需预扫描或物化中间表即可确保连接结果正确。因此，Bala-Join 可以在单次遍历中完成倾斜检测和重新分配。

示例：图3还展示了 ASAP 机制的工作方式。构建表中的所有元组均经过哈希分布。当倾斜探测元组 $\langle x、2\rangle_p$ 和 $\langle x、3\rangle_p$ 到达计算节点 2 时，该计算节点会收到通知，得知其收到了连接键为 x 的元组。随后，该计算节点会异步拉取已哈希分布至计算节点 1 的匹配构建元组 $\langle x、1\rangle_b \sim \langle x、3\rangle_b$。

通过启用独立数据拉取，接收倾斜探测元组的节点集合隐式构成全局目标集 $U(x)$，并从中拉取对应的构建元组，从而有效实现构建数据的多播。

如第五节 A 部分所述，本地检测偏差可能导致连接键 $\boldsymbol{x}$ 在某些节点上被识别为倾斜，而在其他节点上被识别为非倾斜。ASAP 可无缝处理这种不一致情况。如果元组被路由到 q（假阴性），q 已在本地持有构建元组（通过标准哈希分布接收），因此无需拉取即可立即执行连接。如果元组被路由到 $U(x)$ 中的其他任一节点（真阳性），ASAP 机制可确保该节点能够从 q 拉取构建元组。由于序列生成器保证 $q \in U(x)$，构建目标 q 始终位于全局候选集中，因此本地检测错误只会影响效率（负载均衡），绝不会影响结果的正确性。

总而言之，所提出的检测器完全基于本地状态运行，有效消除了全局一致所带来的通信开销和同步屏障。通过利用序列生成器确保算法的稳健性，即使存在不一致的本地视图，系统仍能保证正确性。该设计明确解决了**Chal.** ❷，因为它消除了对全局状态协调的依赖，同时确保倾斜元组在有效候选集内得到处理。

VI. 评估

A. 实验设置

我们的设置使用专为分布式哈希连接构建的仿真框架，复现了我们与浪潮开展的工业项目中的真实跨区域拓扑。这为评估广域网上的性能建立了现实的测试平台，使我们能够通过系统地将网络带宽从拥塞水平调整至高性能水平，来评估该解决方案的稳健性。我们采用部署在北京、上海和贵阳的 $1 \sim 8$ 华为云服务器，它们共同组成一个包含 3 至 24 个节点的集群。每个节点配备 4 核中央处理器、8GB 随机存取存储器和通用固态硬盘，并运行 Ubuntu 18.04 操作系统。

我们评估了专为倾斜数据设计的各种重新分配策略的性能，包括 GraHJ [38]，PRPD [8]，PnR [12]，SFR [40],和 BPPR。

表 III：第五节-B 中所有实验的不同算法的平均吞吐量和排名

	带宽	平衡因子	Zipf 因子	$\lvert R\rvert/\lvert S\rvert$	综合排名
BPPR	**165M** (1)	**125M** (1)	**178M**(1)	**154M** (1)	**622M** (1)
PnR	$159M$ (2)	$101M$ (3)	$172M$ (2)	$137M$ (4)	$569M$ (2)
PRPD	$140M$ (3)	$94M$ (4)	$142M$ (3)	$148M$ (2)	$524M$ (3)
SFR	$127M$ (4)	$81M$ (5)	$131M$ (4)	$140M$ (3)	$470M$ (4)
GraHJ	$105M$ (5)	$103M$ (2)	$102M$ (5)	$103M$ (5)	$413M$ (5)

随后，我们将 Bala-Join 与采用倾斜检测器的其他分布式哈希连接基线方法进行性能比较，如下所列。

- GraHJ：一种仅使用哈希分区的策略
- PRPD：一种将倾斜元组保留在本地的策略
- PnR：一种将倾斜元组均匀分布的策略
- SFR：一种将倾斜元组对称复制到矩阵各行和各列上的节点的策略
- Flow-Join：一种综合解决方案，包含 SFR 策略以及基于 Space Saving 算法的检测器

对于实验数据，我们使用服从 Zipf [41] 分布的合成数据集，该分布在以往的研究中被广泛采用[10]，[15]，[33]，[39]，[42]。数据倾斜程度与 Zipf 因子 z呈正相关。此外，我们采用 SSB-skew 数据集 [43]，其规模因子设为 10；该数据集是星型模式基准测试 [44]的一个变体。我们关注查询中的连接操作，并通过分布式哈希连接执行这些操作。

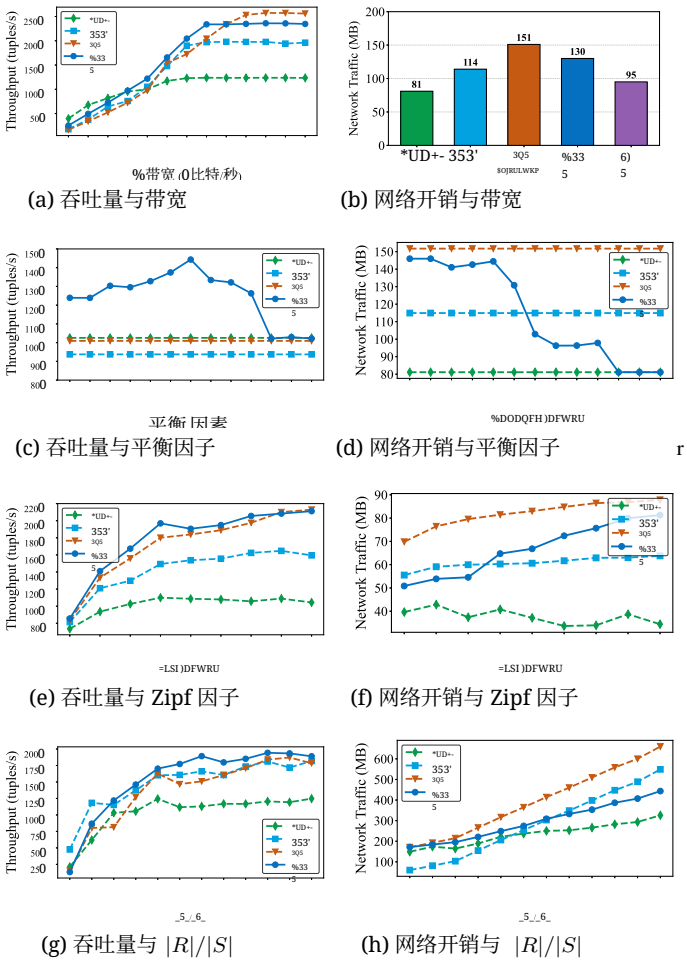
性能通过吞吐量和网络流量来衡量。吞吐量是指集群每秒生成的结果元组数，计算方法是将连接结果元组总数除以端到端耗时（启用倾斜检测时包括倾斜检测、重新分配和本地连接计算）。网络流量表示通过网络传输的数据量（以字节为单位）。前者反映系统的处理效率，后者则体现网络开销。

B. 重新分区策略

首先，我们进行了一系列实验来评估重新分配策略。在这些实验中，直接向每种算法提供倾斜信息。默认设置：带宽为 = 100 兆比特/秒，平衡因子为 = 0.2，Zipf 因子为 =1.25，且为 $|R|/|S|$ = 2/3。

图 4(a) 和图 4(b) 展示了五种策略在 10–300 兆比特/秒不同带宽下的性能。GraHJ 的网络流量最低，而 BPPR 的流量比 PnR 少 21%，但比 PRPD 多 14%，且高于 SFR。BPPR 在中等带宽下表现最佳，在高带宽下也具有竞争力，尤其是与 GraHJ 相比时。

图 4(c) 和图 4(d) 探究了 BPPR 平衡因子的影响。随着平衡因子增大，BPPR 的吞吐量起初会上升，并在 0.3 时达到最高值。之后，吞吐量下降，并在 0.7 时最终降至与 GraHJ 相近的水平。如图 4(d) 所示，较宽松的平衡阈值可以使 BPPR 针对每个倾斜键保留的节点集合 U（x）更小，从而减少



(a) 吞吐量与带宽 (b) 网络开销与带宽

(c) 吞吐量与平衡因子 (d) 网络开销与平衡因子

(e) 吞吐量与 Zipf 因子 (f) 网络开销与 Zipf 因子

(g) 吞吐量与 $|R|/|S|$ (h) 网络开销与 $|R|/|S|$

图 4：不同场景下分布策略的比较

构建端的组播范围，并显著降低整体网络传输开销。

图 4(e) 和图 4(f) 比较了不同 Zipf 因子下的算法表现，Zipf 因子用于控制数据倾斜程度。BPPR 在吞吐量方面优于所有其他算法，并且在网络开销方面排名 2nd 。

图 4(g) 和图 4(h) 评估了不同比例 $|R|/|S|$ 下的性能。BPPR 始终保持稳定；当 $|R|/|S|$ 小于 0.1 时，其表现略逊于 PRPD。

表 III 汇总了四组实验中的平均吞吐量和排名。与 PRPD、SFR 和 PnR 相比，BPPR 展现出更强的适应性，并在所有实验设置中始终取得最高排名。

C. 分布式倾斜检测器

本节评估第五节提出的分布式数据倾斜检测器的性能。由于该策略与 BPPR 紧密集成，构成 Bala-Join，为了公平起见，我们将其与另外两种方案进行比较：

BPPR（不含倾斜检测器）：此版本预先向 BPPR 提供倾斜值，以观察

表 IV：第六节 C 部分所有实验中不同算法的平均吞吐量及比较

	Bala-Join	BPPR+Det	BPPR
带宽（3 个节点）	$178M$	$172M(+3\%)$	$186M(-4\%)$
Zipf 因子（3 个节点）	$59M$	$55M(+7\%)$	$61M(-8\%)$
$\|R\|/\|S\|$ (3 个节点)	$194M$	$179M(+8\%)$	$199M(-3\%)$
带宽（6 节点）	$221M$	$200M(+11\%)$	$230M(-4\%)$
Zipf 因子（6 节点）	$274M$	$240M(+14\%)$	$292M(-6\%)$
$\|R\|/\|S\|$ (6 个节点)	$235M$	$205M(+15\%)$	$253M(-7\%)$

分布式运行时检测器引入的开销。因此，移除检测器可消除检测成本并提高吞吐量。

采用独立数据倾斜检测器的 BPPR（BPPR+Det)：该方案在 BPPR [22]旁采用独立的 Space Saving 检测算法。

通过比较这三种方案，我们可以评估分布式检测器引入的开销。由于所有系统都使用 BPPR 处理倾斜数据，因此网络开销没有差异，比较仅关注吞吐量。为便于说明，在后续实验中，我们将这些方案简称为 BPPR、BPPR+Det 和 Bala-Join（本文提出的解决方案）。

图 5(a) 和图 5(d) 展示了不同带宽下，3 节点和 6 节点集群中不同策略的吞吐量。在这两种情况下，Bala-Join 的表现介于 BPPR 和 BPPR+Det 之间。我们还考察了不同 Zipf 因子（图 5(c) 和图 5(f)）以及 $|R|/|S|$ 比率（图 5(b) 和图 5(e)）下的性能。

在所有实验中，Bala-Join 曲线始终位于 BPPR 和 BPPR+Det 之间，这表明分布式检测器会带来一些开销，但其性能显著优于 BPPR+Det 中使用的独立 Space-Saving 检测器。这印证了第 V 节讨论的挑战。平均而言，与 BPPR 相比，所提出的检测器会增加约 5% 的开销；与 3 节点配置相比，6 节点配置通常开销更大。

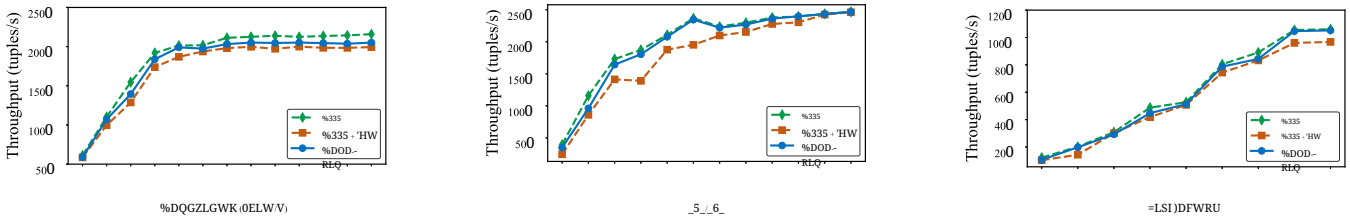
表 IV 汇总了所有场景下三种策略的每秒平均吞吐量。

D. 整体 *Dist-HJ* 解决方案

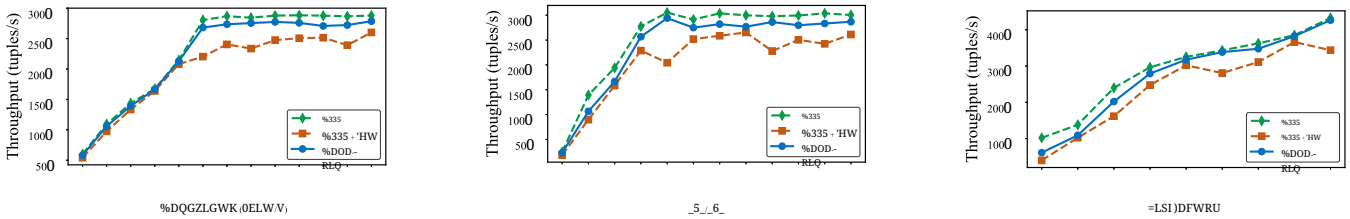
最后，我们评估完整的 Dist-HJ 解决方案，包括数据倾斜检测、重新分配和计算的整个流程。图 6 和图 7 分别比较了 3 节点和 6 节点集群中不同解决方案的性能。表 V 汇总了各种场景下每种解决方案的每秒平均吞吐量，以及我们的解决方案相较于 Flow-Join 和 GraHJ 的性能提升。

图 8 展示了在 24 节点集群上评估的两个最常见连接操作的结果，这两个操作在 SSB-skew 基准测试中出现频率最高（在其他集群规模下也观察到了类似趋势）。

这两种连接操作涉及的表都存在不同程度的数据倾斜。对于连接“LO ORDERDATE =D DATEKEY”，出现频率最高的连接键占比为 0.09%；而对于“LO SUPPKEY =S SUPPKEY”，该比例为 0.55%。随着

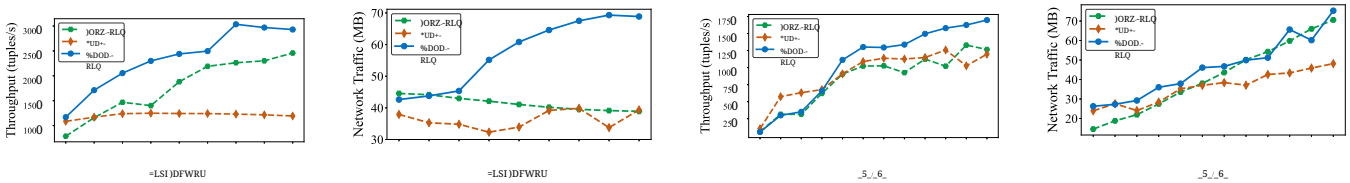


(a) 吞吐量与带宽的关系（3 个节点）　(b) 吞吐量与 $|R|/|S|$ 的关系（3 个节点）　(c) 吞吐量与 Zipf 因子的关系（3 个节点）



(d) 吞吐量与带宽的关系（6 个节点）　(e) 吞吐量与 $|R|/|S|$ 的关系（6 个节点）　(f) 吞吐量与 Zipf 因子（6 个节点）

图 5：运行时倾斜值检测器的开销



（a）吞吐量与 Zipf 因子　（b）网络开销与 Zipf 因子　（c）吞吐量与 $|R|/|S|$　（d）网络开销与 $|R|/|S|$

（e）吞吐量与 Zipf 因子的关系　（f）网络开销与 Zipf 因子的关系　（g）吞吐量与 $|R|/|S|$ 的关系　（h）网络开销与 $|R|/|S|$ 的关系

图 6：不同集群设置下 Dist-HJ 解决方案的比较：（a-d）3 个节点；（e-h）6 个节点

倾斜程度增加，Bala-Join 的性能优于 GraHJ 和 Flow-Join。值得注意的是，在情况 (a) 中，GraHJ 的性能优于 Bala-Join。这是因为在 SSB-skew 基准测试中，相同的连接键（LO ORDERDATE）被聚集在一起。因此，检测器可能会过早地将先到达的元组判定为倾斜元组，从而导致不必要的分区。这一发现对于实际部署数据倾斜检测机制具有重要价值。

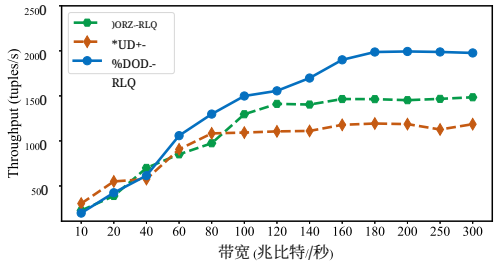
在所有实验场景中，我们的综合解决方案在吞吐量方面始终优于 Flow-Join。然而，在网络开销方面，它并不明显优于 Flow-Join。此外，采用 SFR 倾斜处理策略的 Flow-Join，在节点数量从 3 个增加到 6 个时，呈现出网络开销降低的趋势。不过，我们的结果表明，我们的解决方案在网络开销与计算性能之间实现了更好的平衡。尽管网络开销更高，我们的算法仍显著提升了执行效率。

表 V：第 VI-D 节所有实验中不同 Dist-HJ 实现的平均吞吐量及比较

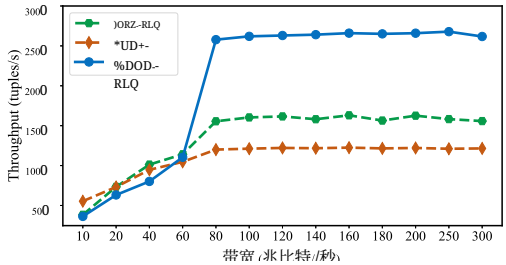
	Bala-Join	Flow-Join	GraHJ
带宽（3 个节点）	**140M**	$112M(+25\%)$	$97M(+44\%)$
Zipf 因子（3 个节点）	**235M**	$177M(+33\%)$	$121M(+94\%)$
$\|R\|/\|S\|$ (3 个节点)	**107M**	$83M(+29\%)$	$91M(+18\%)$
带宽（6 个节点）	**205M**	$135M(+52\%)$	$109M(+88\%)$
Zipf 因子（6 个节点）	**328M**	$208M(+58\%)$	$121M(+171\%)$
$\|R\|/\|S\|$ (6 个节点)	**163M**	$101M(+61\%)$	$99M(+65\%)$

VII. 结论

分布式哈希连接（Dist-HJ）在地理分布式数据库（如 CockroachDB）中的效率受到数据倾斜的严重制约，尤其是在广域网（WANs）上，这构成了一项严峻的工业挑战。在本文中，我们提出 Bala-Join，一种旨在缓解这一特定性能瓶颈的自适应解决方案。我们的核心

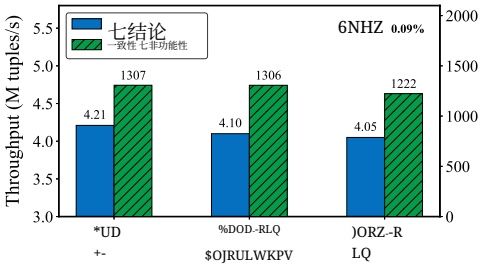


（a）吞吐量与带宽（3 个节点）

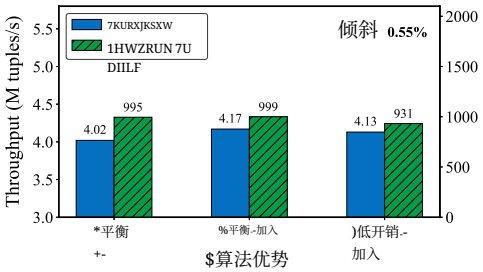


（b）吞吐量与带宽（6 个节点）

图 7：Dist-HJ 解决方案的吞吐量与带宽



(a) LO_ORDERDATE =D DATEKEY



(b) 低阶 SUPPKEY =S SUPPKEY

图 8：Dist-HJ 解决方案在 SSB倾斜 上的性能

方案，即 BPPR 算法，采用受控组播机制，审慎地在计算负载分布与网络开销之间取得平衡。在严格的平衡因子约束下，BPPR 对倾斜元组进行细粒度划分，从而在最小化构建表组播范围的同时实现有界的负载均衡。该算法与在线倾斜检测器紧密集成，后者利用主动信令与异步拉取机制，以促进数据流的高效实时处理。我们的实证评估表明，Bala-Join 实现了在网络与计算性能之间实现了更优平衡，使吞吐量提升 25%-61%，并证实了其在真实部署中的实际可行性。

参考文献

[1] M. Stonebraker，“无共享架构的理由”，《*IEEE* 数据库工程通报》，第 9 卷第 1 期，第 4–9 页，1986 年。[2] R. Taft、I. Sharif、A. Matei、N. VanBenschoten、J. Lewis、T. Grieger、K. Niemi、A. Woods、A. Birzin、R. Poss等，“CockroachDB：具备弹性的地理分布式 SQL 数据库”，载于*2020* 年 *ACM SIGMOD* 数据管理国际会议论文集，2020 年，第 1493–1509 页。[3] D. Huang、Q. Liu、Q. Cui、Z. Fang、X. Ma、F. Xu、L. Shen、L. Tang、Y. Zhou、M. Huang等，“TiDB：基于 Raft 的 HTAP 数据库”，*VLDB*基金会会刊，第 13 卷第 12 期，第 3072–3084 页，2020 年。[4] J. C. Corbett、J. Dean、M. Epstein、A. Fikes、C. Frost、J. J. Furman、S. Ghemawat、A. Gubarev、C. Heiser、P. Hochschild等，“Spanner：Google 的全球分布式数据库”，《*ACM* 计算机系统汇刊（*TOCS*）》，第 31 卷第 3 期，第 1–22 页，2013 年。[5] Z. Yang、C. Yang、F. Han、M. Zhuang、B. Yang、Z. Yang、X. Cheng、Y. Zhao、W. Shi、H. Xi等，“OceanBase：每分钟 7.07 亿事务处理性能的分布式关系数据库系统”，*VLDB*基金会会刊，第 15 卷第 12 期，第 3385–3397 页，2022 年。[6] D. J. DeWitt、J. F. Naughton、D. A. Schneider 和 S. Seshadri，“并行连接中的实用倾斜处理”，威斯康星大学麦迪逊分校计算机科学系，技术报告，1992 年。[7] A. Metwally、D. Agrawal 和 A. El Abbadi，“数据流中频繁项和 Top-k 元素的高效计算”，载于数据库理论——*ICDT 2005*：第 *10* 届国际会议，英国爱丁堡，*2005* 年 *1* 月 *5—7* 日，论文集第 *10* 卷。Springer，2005 年，第 398–412 页。[8] Y. Xu、P. Kostamaa、X. Zhou 和 L. Chen，“无共享系统中并行连接的数据倾斜处理”，载于*2008* 年 *ACM SIGMOD* 数据管理国际会议论文集，2008 年，第 1043–1052 页。[9] C. Barthels、S. Loesing、G. Alonso 和 D. Kossmann，“使用 RDMA 的机架级内存内连接处理”，载于*2015* 年 *ACM SIGMOD* 数据管理国际会议论文集，2015 年，第 1463–1475 页。[10] W. R¨odiger 、 S. Idicula 、 A. Kemper 和 T. Neumann，“Flow-Join：高速网络上分布式连接的自适应倾斜处理”，载于 *2016* 年 *IEEE* 第 *32* 届数据工程国际会议（*ICDE*）。IEEE，2016 年，第 1194–1205 页。[11] O. Polychroniou、R. Sen 和 K. A. Ross，“Track Join：网络流量最小化的分布式连接”，载于*2014* 年 *ACM SIGMOD* 数据管理国际会议论文集，2014 年，第 1483–1494 页。[12] J. Yang、H. Li、Y. Si、H. Zhang、K. Zhao、K. Wei、W. Song、Y. Liu 和 J. Cui，“没有一种方案适用于所有情况：无共享关系数据库管理系统中倾斜哈希连接的自适应调度器”，载于第 *29* 届高级应用数据库系统国际会议论文集，2024 年。[13] Q. Pu、G. Ananthanarayanan、P. Bodik、S. Kandula、A. Akella、P. Bahl 和 I. Stoica，“低延迟地理分布式数据分析”，《*ACM SIGCOMM* 计算机通信评论》，第 45 卷，434，2015 年 4，421第。页–。[14] A. Pradhan、S. Karthik 和 R. S，“大规模地理分布式数据分析的最优查询计划”，载于第七届数据科学与数据管理联合国际会议论文集（第 *11* 届 *ACM IKDD CODS* 和第 *29* 届 *COMAD*），2024 年，第 247–251 页。[15] M. Kitsuregawa 和 Y. Ogawa，“桶扩展并行哈希：超级数据库计算机（SDC）中用于处理数据倾斜的一种新型稳健并行哈希连接方法。”载于*VLDB*，1990 年，第 210–221 页。[16] A. Vitorovic、M. Elseidy 和 C. Koch，“并行连接的负载均衡与抗倾斜能力”，载于*2016* 年 *IEEE* 第 *32* 届数据工程国际会议（*ICDE*）。IEEE，2016 年，第 313–324 页。[17] P. B. Gibbons 和 Y. Matias，“用于改进近似查询答案的新型采样汇总统计”，载于*1998* 年 *ACM SIGMOD* 数据管理国际会议论文集，1998 年，第 331–342 页。[18] G. S. Manku 和 R. Motwani，“数据流上的近似频率计数”，载于*VLDB'02*：第 *28* 届超大型数据库国际会议论文集。Elsevier，2002 年，第 346–357 页。

[19] E. D. Demaine、A. L´opez-Ortiz 和 J. I. Munro，“有限空间下互联网数据包流的频率估计”，载于 Esa，第 2 卷。Citeseer，2002 年，第 348–360 页。[20] M. Charikar、K. Chen 和 M. Farach-Colton，“在数据流中查找频繁项”，《理论计算机科学》，第 312 卷，15，2004 年 1，3第。页– 。[21] G. Cormode 和 S. Muthukrishnan，“改进的数据流摘要：Count-Min Sketch 及其应用”，《算法杂志》，第 55 卷第 1 期，第 58–75 页，2005 年。[22] M. Cafaro 、 M. Pulimeno 和 P. Tempesta，“用于频繁项的并行 Space Saving 算法与赫尔维茨 ζ 分布”，《信息科学》第 329 卷，第 1–19 页，2016 年。[23] Y. Zhao、W. Han、Z. Zhong、Y. Zhang、T. Yang 和 B. Cui，“双匿名草图：实现全局 top-k 频繁项查找的 top-k 公平性”，《ACM 数据管理会议论文集》，第 1 卷第 1 期，第 1–26 页，2023 年。[24] C. B. Walton、A. G. Dale 和 R. M. Jenevein，“并行连接中数据倾斜效应的分类与性能模型”，载于第 17 届超大型数据库国际会议论文集，1991 年，第 537–548 页。[25] K. A. Hua 和 C. Lee，“使用分区调优处理多处理器数据库计算机中的数据倾斜。”载于 VLDB，第 91 卷，1991 年，第 525–535 页。[26] K. Alsabti 和 S. Ranka，“关系连接的倾斜无关并行算法”，《沙特国王大学计算机与信息科学杂志》，第 13 卷，第 79–110 页，2001 年。[27] J. L. Wolf、D. M. Dias 和 P. S. Yu，“存在数据倾斜时哈希连接并行化的有效算法”，载于论文集。第七届数据工程国际会议。IEEE 计算机学会，1991 年，第 200–201 页。[28] ——，“用于管理数据倾斜的并行排序归并连接算法”，《IEEE 并行与分布式系统汇刊》，第 4 卷第 1 期，第 70–86 页，1993 年。[29] J. L. Wolf、D. M. Dias、P. S. Yu 和 J. Turek，“存在数据倾斜时关系数据库连接并行化的新算法”，《IEEE 知识与数据工程汇刊》，第 6 卷第 6 期，第 990–997 页，1994 年。[30] H. M. Dewan 、 K. W. Mok、M. Hern´andez 和 S. J. Stolfo，“存在数据倾斜时异构处理器上并行哈希连接的预测式动态负载均衡”，载于第 3 届并行与分布式信息系统国际会议论文集。IEEE，1994 年，第 40–49 页。[31] R. L. Graham，“多处理时序异常的界限”，《SIAM 应用数学杂志》，第 17 卷第 2 期，第 416–429 页，1969 年。[32] E. G. Coffman, Jr、M. R. Garey 和 D. S. Johnson，“装箱问题在多处理器调度中的应用”，《SIAM 计算杂志》，第 7 卷第 1 期，第 1–17 页，1978 年。[33] A. Shatdal 和 J. F. Naughton，“使用共享虚拟内存进行并行连接处理”，载于 1993 年 ACM SIGMOD 数据管理国际会议论文集，1993 年，第 119–128 页。[34] M. Kitsuregawa，“无共享数据库系统中哈希连接的动态连接结果倾斜处理”，载于第四届高级应用数据库系统国际会议论文集，第 5 卷。世界科学出版社，1995 年，第 246 页。[35] X. Zhou 和 M. E. Orlowska，“使用两阶段调度处理并行哈希连接计算中的数据倾斜”，载于第一届并行处理算法与体系结构国际会议论文集，第 2 卷。IEEE，1995 年，第 527–536 页。[36] S. Zhou、F. Zhang、H. Chen、H. Jin 和 B. B. Zhou，“FastJoin：感知倾斜的分布式流连接系统”，载于 2019 年 IEEE 并行与分布式处理国际研讨会（IPDPS）。IEEE，2019 年，第 1042–1052 页。[37] E. Gavagsaz、A. Rezaee 和 H. Haj Seyyed Javadi，“MapReduce 系统中倾斜数据连接算法的负载均衡”，《超级计算杂志》，第 75 卷，第 228–254 页，2019 年。[38] M. Kitsuregawa、H. Tanaka 和 T. Moto-Oka，“哈希在数据库机器中的应用及其体系结构”，《新一代计算》，第 1 卷第 1 期，第 63–74 页，1983 年。[39] Q. Huang 和 P. P. Lee，“LD-Sketch：一种用于网络数据流中精准且可扩展异常检测的分布式草图设计”，载于 IEEE INFOCOM 2014——IEEE 计算机通信会议。IEEE，2014 年，第 1420–1428 页。[40] J. W. Stamos 和 H. C. Young，“一种用于分布式连接的对称分片复制算法”，《IEEE 并行与分布式系统汇刊》，第 4 卷第 12 期，第 1345–1354 页，1993 年。[41] G. K. Zipf，《人类行为与最省力原则：人类生态学导论》。Addison-Wesley，1949年。[42] W. R¨odiger、T. M¨uhlbauer、P. Unterbrunner、A. Reiser、A. Kemper 和 T. Neumann，“面向并行主存数据库集群的局部性感知算子”，载于*2014*年*IEEE*第*30*届数据工程国际会议。IEEE，2014年，第592–603页。[43] D. Justen、D. Ritter、C. Fraser、A. Lamb、A. Lee、T. Bodner、M. Y. Haddad、S. Zeuch、V. Markl 和 M. Boehm，“Polar：通过最小阻力计划实现自适应且非侵入式的连接顺序选择”，*VLDB*基金会会刊，第17卷第6期，第1350–1363页，2024年。[44] P. E. O’Neil、E. J. O’Neil 和 X. Chen，“星型模式基准测试（SSB）”，专利，第200卷第0期，第50页，2007年。