arXiv:2303.07787v1 [cs.DB] 14 Mar 2023

一刀切并不适用：共享无架构关系型数据库管理系统中倾斜哈希连接的自适应调度器

杨金鑫 李辉 司一鸣 西安电子科技大学 中国西安
y
angjx
@stu.xidian.edu.cn
hli@xidian.edu.cn Si_
ym@hotmail.com

张辉
赵侃侃
魏科伟
浪潮软件集团有限公司
中国北京
zhanghui@inspur.com
zhaokankan@inspur.com
weikewei@inspur.com

宋文龙
刘英凡
崔江涛
西安电子科技大学
中国西安
wlsong_xdu@foxmail.com
{liuyingfan,cuijt}@xidian.edu.cn

摘要

无共享架构已广泛应用于各种商用分布式关系数据库管理系统。得益于这一架构，查询可以并行处理，并可按需横向扩展集群以提升处理速度。尽管如此，负载均衡仍是所有分布式关系数据库管理系统面临的一项挑战，其中也包括共享无架构系统；这类系统深受数据分布倾斜的影响。在本研究中，我们聚焦于一种代表性算子——哈希连接，并探究集群节点间的倾斜如何影响共享无架构关系数据库管理系统中任意查询的负载均衡及最终效率。我们发现，当某个值在探测表和构建表中都出现倾斜时，现有的分布式哈希连接方案可能无法提供令人满意的性能。

为解决这一问题，我们提出了一种新颖的分布式哈希连接方案，即分区与复制（PnR）。尽管 PnR 在某些倾斜场景下效率最高，但我们对一组共享无架构关系数据库管理系统进行的全面实验表明，在所有（数据倾斜）场景中，没有一种分布式哈希连接方案能够始终胜出。因此，我们进一步提出了一种自适应分布式哈希连接方案，该方案内置子算子成本模型，可根据目标查询的数据倾斜情况，在运行时动态选择最佳的分布式哈希连接实现策略。我们在商用共享无架构关系数据库管理系统 KaiwuDB（原名 ZNBase）中实现了该方案，实证研究证实，与许多现有关系数据库管理系统采用的一系列方案相比，该自适应模型的性能最佳。

引用格式：
杨金鑫、李辉、司一鸣、张辉、赵侃侃、魏科伟、宋文龙、刘英凡和崔江涛。《一刀切并不适用：面向共享无架构关系型数据库管理系统中倾斜哈希连接的自适应调度器》 doi:XX.XX/XXX.XX

工件可用性：
源代码、数据和/或其他工件已在 https://github.com/lihuixidian/SkewHJs 上提供。

本作品采用知识共享署名-非商业性使用-禁止演绎 4.0 国际许可协议授权。请访问 https://creativecommons.org/licenses/by-nc-nd/4.0/ 查看该许可协议副本。如需进行该许可协议未涵盖的任何用途，请发送电子邮件至 以获取许可。版权归所有者/作者所有。出版权已授权给 。doi:XX.XX/XXX.XX

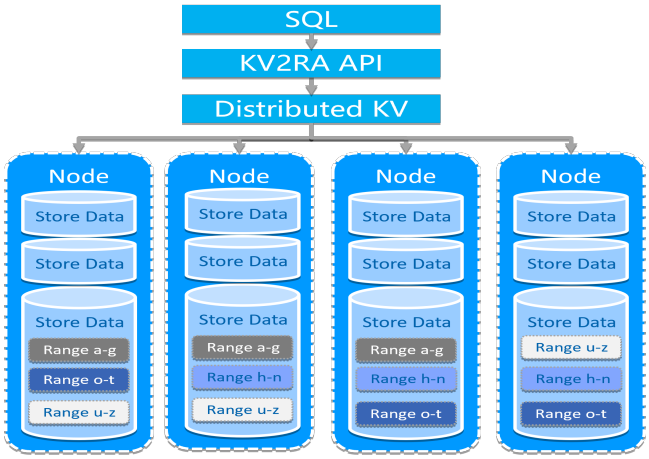


图1：KaiwuDB（原名 ZNBase）的架构

1 引言

由于原生支持并行查询处理且易于扩展，无共享架构已广泛应用于各种商用分布式关系数据库管理系统 [12, 27]。这些关系数据库管理系统采用分片技术，并在数据库集群中对数据进行水平分区。只需添加服务器，即可轻松加速并行查询处理。遵循无共享架构 [26]，浪潮软件集团开发了一款新的商用关系数据库管理系统 KaiwuDB（原名 ZNBase[1]），其系统架构如图 1所示。KaiwuDB 的每个节点都拥有独立的计算、内存和存储资源，其存储引擎基于一款流行的键值存储 RocksDB 构建 [5]。

表存储在一个由键值对构成的单体映射中。单个表的键在键空间中连续排列，据此，表被划分为若干连续的片段，即“范围”。每个范围会复制到 3 个副本（默认值），并分别存储在 3 个不同的节点上。在所有副本中，有一个被称为“租约持有者”，负责协调该范围的读请求和写请求。理想情况下，读请求始终发送给租约持有者，其元组始终保持最新状态。值得注意的是，租约持有者并非静态不变，而是通过选举产生的

[1]https://www.znbase.com/website/

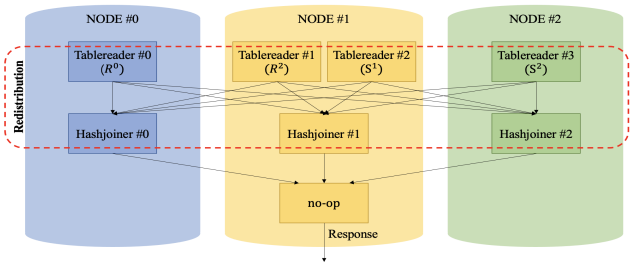


图 2：KaiwuDB 中分布式哈希连接的一种潜在执行计划

并按照 Raft 协议 [20] 进行管理。大多数共享无架构关系型数据库管理系统 [12, 27] 都普遍采用类似的设置。

1.1 KaiwuDB 中的分布式哈希连接

给定一条 SQL 查询，关系型数据库管理系统会借助查询优化器对其进行编译，并相应地生成查询执行计划。查询执行计划由一组物理算子组成，这些算子按树结构组织，展示了给定查询将如何一步步处理并返回结果（即，逐个算子处理）。传统关系型数据库管理系统包含数十种预定义算子，其中哈希连接被广泛使用和研究 [6, 18, 19, 23, 28, 30, 31]。

通常，哈希连接算子包含两个子步骤：哈希重分布和连接；具体来说，在无共享分布式（例如，KaiwuDB）关系型数据库管理系统中，哈希连接必须以分布式方式执行。与 [12, 27], 给定一条 SQL 语句时类似，KaiwuDB 中的查询计划生成通常可分为两个阶段。第一阶段称为逻辑规划，将从给定 SQL 解析出的相应抽象语法树（ast）输入优化器，以生成优化后的逻辑计划。逻辑计划采用树结构，其中节点表示逻辑算子。第二阶段称为物理规划，系统将优化后的逻辑计划转换为物理计划，指定如何在集群中分布式并行执行。举例来说，逻辑扫描操作（在第一阶段生成）由一系列 Tablereader（物理上）执行，每个节点对应一个。Tablereader 合并对位于同一节点上的范围的访问。同样，后续的逻辑算子（例如，JOIN）也会在此阶段由物理算子（例如，哈希连接）在各节点上实现并分布执行。

由于范围局部性在各节点间交错分布（即 横跨若干节点而不严格遵循顺序），且范围的分布并非静态（即 可能随元组变化而改变），因此表读取器的物理对齐是在运行时动态决定的。因此，作为后续算子，分布式哈希连接（简称分布式哈希连接）的具体工作负载高度取决于前一阶段的表读取器输出了什么。

1.2 分布式哈希连接中的数据倾斜

图 2 展示了 KaiwuDB 在包含 $k \geq 3$ 个节点的集群上运行时，分布式哈希连接的一种潜在执行计划。不失一般性，假设以下 SQL 任务在集群中的 3 个节点上执行，因此分布式哈希连接也将在这 3 个节点上执行。

```
SELECT * FROM R,S WHERE R.a=S.b;
```

根据分布式执行计划，节点 0 和节点 1 访问 R 的分区，而节点 1 和节点 2 分别访问 S 的分区。默认情况下，系统会进一步采用基本哈希重分布，将连接任务分配到各节点上，从而使其能够并行运行 [15]。

假设 $R.a$中存在一个倾斜值 x，且 $hash(x) \bmod 3 = 1$。形式上，给定表 R，如果某个值 x在列 a中出现的次数超过任意列条目数的 $p|R|$（例如 20%），则该值被称为**倾斜的**，其中 p 称为倾斜阈值。因此，节点 1 将过载，因为具有 $R.a = x$的元组都会被分配给它。文献中有两种解决倾斜哈希连接的代表性方案，即部分重分布与部分复制（PRPD） [29] 和 Flow-Join [21]。在 PRPD 中，来自 R 且值为 x 的元组保留在节点 0 和节点 1 本地，而来自 S 且值为 x 的元组则广播到全部三个节点。PRPD 的思路是平衡所有三个节点的工作负载。然而，将 R 的倾斜元组保留在本地，势必会导致节点 2 利用不足，因为最初没有与节点 2 对齐的 R 倾斜元组。另一方面，对于 Flow-Join，如果 x 在 S中不倾斜，其行为与 PRPD 相同。否则，将采用对称片段复制（SFR） [25]来平衡各节点的工作负载。然而，正如我们将在本文后文所见，SFR 在许多实际场景中同样无法平衡工作负载。

具体而言，KaiwuDB 中分布式哈希连接的倾斜可能由非均匀数据分布或有偏的物理规划造成。直观来看，对于这两种情况，假设可以将来自R 的倾斜元组按轮询方式分配给所有节点，并将来自 S的相关元组进行广播，那么所有节点从两张表中接收的倾斜元组数量可能大致相同。因此，这能平衡各节点之间的工作负载。受上述见解启发，我们在 KaiwuDB 中进行了新的尝试，提出了一种新的分布式哈希连接解决方案，即分区与重分布（PnR）。

值得注意的是，与 PRPD 和 Flow-Join 相比，由于 PnR 采用轮询方式，许多倾斜元组现在会通过网络进行分发，这可能会增加额外的通信成本。在实践中，我们发现，在 KaiwuDB 中，这些解决方案没有一种能够始终胜出（本文的实证研究和理论研究表明，这一现象不仅存在于 KaiwuDB 中，也存在于其他共享无架构关系型数据库管理系统中）。这直接促使我们进一步开发适用于实际系统的自适应解决方案。本文后续将详细介绍 PnR 和自适应解决方案。

本文其余部分安排如下：第 2 节回顾并归类一系列具有代表性的分布式哈希连接解决方案，展示它们面对不同倾斜场景时的优缺点。随后，我们从理论上讨论并比较 GraHJ（KaiwuDB 和其他一系列关系型数据库管理系统 [17, 27] 中的默认解决方案）与 PRPD [29] （分布式哈希连接中解决倾斜问题的最先进方案之一）针对倾斜值的分布式哈希连接所采用的策略，并揭示二者的局限性。为此，我们在第 3 节提出一种新的替代策略，即分区与重分布（PnR）。我们在 KaiwuDB 中实现了这三种策略（将它们视为分布式哈希连接的不同子算子），并通过理论研究和实证测试发现，没有一种策略能在所有倾斜场景中胜出。为解决这一问题，我们在第 4 节提出一种原生嵌入查询优化器的自适应子算子调度器，通过在运行时识别倾斜场景，动态选出最佳策略

。我们在 KaiwuDB 中实现了所提出的自适应调度器，并在第 5 节测试其性能。最后，第 6 节总结本文的贡献。

2 相关研究与系统

分布式关系型数据库管理系统中的并行查询处理已对数据倾斜进行了广泛研究。[4], 中通常将数据倾斜分为四种类型：元组放置倾斜（TPS）、选择率倾斜（SS）、重分布倾斜（RS）和连接结果倾斜（JPS）。元组放置倾斜是指元组在 N 副本间的初始分布不均衡，这种情况可以通过谨慎选择分区列和副本放置位置来避免。选择率倾斜是指由过滤谓词选择率不同所导致的情况，由于查询中的谓词各不相同，因此对此类倾斜的研究较少。重分布倾斜是指重新分配后不同节点接收的数据量不均衡。连接结果倾斜是指各节点的连接结果出现倾斜。大多数研究关注重分布倾斜和连接结果倾斜。本文聚焦于重分布倾斜，因为重新分配是分布式哈希连接中的关键步骤，也是最容易受到倾斜影响的步骤。下文将探讨几种针对这一问题的代表性解决方案。

关于 RS 的既有研究可分为三类。第一类算法将倾斜问题视为传统的任务调度问题：针对特定分析阶段扫描并物化两个连接表，并相应地将两张表划分为更小的单元。 [4] 引入范围分区，以取代哈希分区，从而平衡各节点接收的元组数量。范围分区基于在单独的采样阶段计算出的分割向量 。 [1, 11, 14], 采用基于哈希的方法，通过哈希分区将构建表和探测表划分为多个桶。划分后的单元通过最长处理时间优先（LPT） [7] 和 MULTIFIT [13]等任务调度模型进行分配。尽管任务调度算法可以缓解重新分配步骤中的负载不均衡，但它会给连接步骤带来显著开销，即使表不存在倾斜也是如此。第二类算法会动态处理倾斜问题 [9, 24, 32]。过载节点会将超出负荷的工作负载迁移到其他空闲节点。在 KaiwuDB 这类无共享系统中，节点间无法进行任务窃取，原因如下：（a）优雅的无共享架构中，每个节点都拥有独立的存储和计算能力[2],，需要经过重构才能支持任务窃取。（b）任务窃取会带来额外的系统开销，因为节点需要耗费更多时间进行通信和协调。这会增加延迟并降低整体系统吞吐量，即使使用远程直接内存访问（RDMA）也是如此，因为它仍需要一系列单边 RDMA 通信，导致较高的窃取延迟[16]。（c）任务窃取还可能使任务迁入的节点承受显著的额外负载 [10]。第三类算法是选择性广播方案。选择性广播算法最早由 [4], 提出，称为子集复制。选择性广播算法的核心思想是将一张表中的倾斜元组保留在本地，同时复制并广播另一张表中与之相关的元组。前两类算法在宏观层面解决倾斜问题，并未结合连接算法的特性。相比之下，处理选择性广播算法中的倾斜值的方法更有针对性，也更容易适配 KaiwuDB 的架构。我们将在下文详细介绍两种具有代表性的选择性广播算法，即 PRPD [29] 和 Flow-Join [21], 。

2.1 部分重分布与部分复制

PRPD 假设 R 和 S 采用精心选择的列进行分区，使每个节点在同一张表中持有数量几乎相同的倾斜元组和非倾斜元组。PRPD 利用统计信息，并假设事先已知每张表的偏斜值集合。PRPD 的核心思想是分别处理倾斜元组和非倾斜元组。具体而言，非倾斜元组通常经过哈希重分布，而一张表中的倾斜元组则保留在本地（即生成这些元组的节点），另一张表中对应的元组则进行广播。因此，在连接之前，每张表都被划分为三个互不相交的集合，即 分别包含要进行哈希重分布、保留在本地和广播的元组。在将这些集合重新分布或广播到特定节点后，对其执行连接操作；各节点连接结果的并集构成分布式哈希连接的输出。PRPD 还提出了一种变体（称为 PRPD-u[29]），用于处理数据分布不均的情况，即大部分倾斜数据集中在集群中的一个或少数几个节点上。为说明这一点，假设一张表经过均匀分区，且其中大部分倾斜元组位于一个“热点节点”上。按照 PRPD-u 的方法，每个节点上的倾斜数据会被随机重分布 到其他节点，而不是保留在本地。对于另一张表，重分布计划保持不变。据称，这种随机重分布能够大致平衡所有节点的工作负载。

2.2 Flow-Join

与 PRPD 不同，Flow-Join 提出在运行时检测倾斜值，而不是预先使用统计信息。首先，Flow-Join 动态维护一个小型直方图，以识别倾斜数据。根据 Flow-Join 的实证结果，检测工作的耗时不超过总执行时间的 2%。其次，在获得两个输入（R and S）的（估计）倾斜值后，Flow-Join 选择采用与 PRPD 相同的策略：一个输入中的倾斜值保留在本地，并广播另一个表中的对应值。除了上述相似之处，Flow-Join 在运行时检测后处理倾斜元组（即，广播）的方式有所不同。Flow-Join 通过对称片段复制（SFR）重新分配方案重新分配倾斜值。事实上，如果我们依靠收集到的统计信息并关闭运行时倾斜检测器，同时禁用 SFR，那么 PRPD 和 Flow-Join 的行为相同。因此，本文后续将 PRPD 和 Flow-Join 统称为 PRPD。

由于倾斜值的统计信息很容易获取（通过收集频率和直方图），选择性广播算法是解决 KaiwuDB 中分布式哈希连接倾斜问题的一个有前景的方案。然而，如第1.2节所述，物理计划会根据集群上的一系列运行时负载动态生成，因此表读取器访问的数据量和数据分布并非固定不变，即使是同一个查询也如此。因此，KaiwuDB 中的分布式哈希连接可能存在 TPS，且 TPS 的倾斜程度可能会显著波动。这不符合现有解决方案所作的假设

表 1：符号说明。

符号	说明
R, S	探测表和构建表
R^i, S^i	R份额，S位于节点 i
$\|R\|, \|S\|$	R，S元组数量
N	参与分布式哈希连接的计算节点数量
p	倾斜阈值
$\rho_a^p(R)$，$\rho_b^p(S)$	R，S的偏斜值集合
$\rho_{(C)}(R, S)$	属于完全倾斜的 R，S中的元组集合
$\rho_{(C+)}(R, S)$	属于左侧主导的完全倾斜的 R，S中的元组集合 全倾斜
$\rho_{(C-)}(R, S)$	属于右侧主导的 R，S 中的元组集合 完全倾斜
R_1	R中的所有非倾斜元组集合
R_2	R中的部分倾斜 元组集合
R_3	属于 $\rho_{(C+)}(R, S)$ 的元组集合
R_4	属于 $\rho_{(C-)}(R, S)$ 的元组集合
S_2	$t \in S$ 中部分偏斜 的元组集合，位于 R
S_3	元组集合 $t \in S$ 和 $t.a \in \rho_{(C+)}(R, S)$
S_4	元组集合 $t \in S$ 和 $t.a \in \rho_{(C-)}(R, S)$
S_1	元组集合 $t \in S \setminus S_2 \setminus S_3 - S_4$
R_{hash}	元组是哈希分布 在 R
R_{loc}	元组保留在本地于 R
R_{rand}	元组被随机分布在 R
R^{repl}	元组被复制并发送到所有节点在 R
S_{hash}	元组被哈希分布在 S
S_{rand}	元组是 随机分布 在 S
S^{repl}	元组被 复制并发送到所有节点 在 S
$sel(R, x)$，$sel(S, x)$	x 在 R 或 S中的选择率
$M_h(x)$	如果进行哈希重分布，值 x 的代价
$M_h(R, x)$，$M_h(S, x)$	如果在 R 或 S 中进行哈希重分布，值 x 的代价
$M_l(R, x)$，$M_l(S, x)$	如果保留在本地，值 x 在 R 或 S 中的代价
$M_r(R, x)$，$M_r(S, x)$	值为 x 的元组在 R 或 S 中随机分布时的成本
$Q(R, x)$，$Q(S, x)$	值为 x 的元组在 $R.a$ 或 $S.b$中的数量

即哈希连接中的构建表和探测表在每个节点上都被均匀访问。本文提出了一种名为 PnR 的新算法。PnR 缓解了相关元组导致的 JPS 问题，而 [21] 中并未讨论这一问题。在实际工作负载中，我们发现 PRPD（分别是 Flow- Join）、PnR 和优雅的并行 GRACE 哈希连接 [15]（简称 GraHJ）都无法适用于所有工作负载。这一观察促使我们开发一种自适应解决方案，其中设计了一个基于代价的决策器，可根据给定的工作负载选择最优算法。

3 分区与重分布

本节提出一种针对倾斜工作负载的新型分布式哈希连接（Dist-HJ）解决方案，即分区与重分布（PnR）。在 PnR 中，我们聚焦于哈希连接的标准场景，其中 R（探测表）的规模远大于 S（构建表）。为便于讨论，我们先给出一组定义（主要符号汇总于表 1）。

定义 1（倾斜）。给定表 R 和倾斜阈值 p，如果值 x 在 $R.a$ 中（a 是 R中的一列）满足 $|\sigma_{R.a=x}(R)| \geq p|R|$，则称其为倾斜值。为便于讨论，令 $\rho_a^p(R) = \{x \mid |\sigma_{R.a=x}(R)| \geq p|R|\}$（简记为 $\rho(R)$）表示这些 x的集合。

定义2（完全倾斜）。假设我们正在对 R 和另一张表 S 执行等值连接，即 $R \underset{R.a=S.b}{\bowtie} S$；如果 x 在 $R.a$ 和 $S.b$ 中都存在倾斜，即 $x \in \rho(R) \cap \rho(S)$，则称之为完全倾斜，记作 $x \in \rho_{C}(R, S)$。具体而言，给定 $\rho_{C}(R, S)$，如果 $|\sigma_{R.a=x} R| > |\sigma_{S.b=x} S|$ 右侧)主导的完全倾斜，记为 $\rho_{(C+)}(R, S)$ (相应地，$\rho_{(C-)}(R, S)$）。

定义3（部分倾斜）。假设我们正在执行 $R \underset{R.a=S.b}{\bowtie} S$，如果 x 在 $R.a$ 中存在倾斜，但在 $S.b$ 中不存在，即 $x \in \rho(R) \setminus \rho(S)$，则称之为 R 中的部分偏斜，记作 $\rho_{(P)}(R, S)$。

3.1 PRPD 的局限性

根据 PRPD 的假设，当 R^i 和 S^i 在集群的所有计算节点上均衡分布时，还假定每个节点上的倾斜值数量也相应均衡。然而，当我们将 PRPD 引入 KaiwuDB 并替换默认的 GraHJ 方案后，观察到以下问题：

(1) 假设我们正在执行 $R \underset{R.a=S.b}{\bowtie} S$，当连接列中的某个值 x 在 $S.b$中出现部分倾斜，即 $x \in \rho(S) \setminus \rho(R)$时，根据 PRPD，$x$在 R中的元组（即$\{t \mid t \in R,\ t.a = x\}$）将被广播，而 S 中的元组（即$\{t \mid t \in S,\ t.b = x\}$）将保留在本地。由于 $|R| \gg |S|$，很可能出现 $|\sigma_{R.a=x}(R)| \gg |\sigma_{S.b=x}(S)|$的情况，此时广播 R 中与 x 相关的元组必然会引入过多的计算负担。(2) 在 PRPD 中，由于 R^i 和 S^i的数量差异显著，各节点的计算工作负载并不均衡。在 KaiwuDB 中，每个 R^i 或 S^i 都通过 Tablereader 算子读取数据，而该算子读取的记录数并不总是相同，这进一步加剧了 R^i（分别为 S^i）之间的不均衡。(3) 在 PRPD-u 中，所有倾斜值的随机分布引入了不必要的数据传输任务。同时，另一张表中对应的值也需要广播，这必然会导致传输负担大幅波动。

为了解决上述问题，我们提出了一种新的分布式哈希连接方案，即 PnR。在 PnR 中，我们关注探测表中的倾斜值，因为它是上述问题（1）的关键。我们将大表中的倾斜值（即 $\rho(R)$）划分为两部分：1）对于完全倾斜值，即 $\rho(R) \cap \rho(S)$，我们将它们均匀划分，并对小表中与之对应的记录，也就是即$\sigma_{S.b \in \rho(R) \cap \rho(S)}(S)$，进行广播；2）对于 $R.a$ 中的部分倾斜值，即 $\rho(R) \setminus \rho(S)$，我们遵循 PRPD 策略，将它们保留在节点 i 上。我们主张并将在后文证明，这样可以解决上述问题（2）和（3）。

3.2 PnR 描述

假设优化器选择哈希连接来执行分布式关系型数据库管理系统中的 $R \underset{R.a=S.b}{\bowtie} S$ ($|R| > |S|$)。

如今，许多关系型数据库管理系统都会收集频率，因此，两个表中倾斜值的统计信息 [3, 12, 17, 27], 均可 预先获知。在 PnR 中，完全倾斜值，即 也就是 $\rho(R) \cap \rho(S)$，会根据定义2进一步划分为左侧主导和右侧主导两类，分别为 $\rho_{(C+)}(R, S)$ 和 $\rho_{(C-)}(R, S)$。

鉴于上述内容，我们将按以下步骤介绍 PnR 的关键流程。**步骤 1，重新分配**。与 PRPD 和 GraHJ 类似，在执行分布式哈希连接之前，集群中的每个节点 i 都会从前驱算子的输出中获取 R 的一份份额，例如 R^i。与

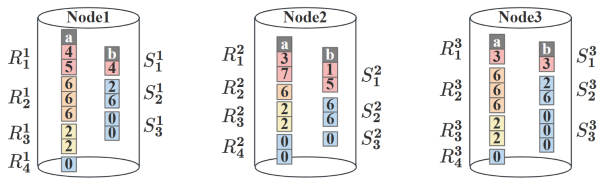


图3：三节点集群上 R 和 S 的分区，其中“6”是完全倾斜值，“2”和“0”分别是 R 和 S 中的部分倾斜值。图中使用的哈希函数为 $h(x) = (x + 1)$ 模 3。

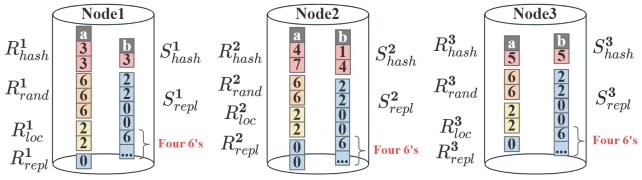


图4：PnR 重分布策略后的数据放置，其中哈希函数为 $h(x) = (x + 1)$ 模 3。

不同于将 R^i 划分为三个互不相交集合的 PRPD，PnR 选择将其划分为四个部分，如下所示。

R_1^i：此部分包含所有非倾斜元组 t $t.a \in$,即 $\backslash\rho(\,)$。它们会 $R^i.a$ R^i 进行哈希重分布（与 GraHJ 相同）。R_2^i：此部分是元组 t，其在 a 中的条目在 R 中呈部分倾斜。它们会保留在本地的节点 i。R_3^i：第三组包含元组 $t \in R^i$ $t.a \in$，其中$()$(,)。它 ρ $C+$ R S 们会随机分布。R_4^i：此部分包含元组 $t \in R^i$，其中 $t.a \in \rho(C-)(R,S)$。它们会广播到所有节点。

根据上述对 R^i 的分区及其对应的重分布策略，每个节点 i 将相应地接收四部分，分别记为 R^i_{hash}、R^i_{loc}、R^i_{rand} 和 R^i_{repl}。例如，R^i_{hash} 包含元组 t，对于所有 $i = 1$、……、N，这些元组在 R_1^i 中满足 $hash(t.a)\ mod\ N = i$；R^i_{loc} 与 R_2^i 相同，因为它保留在 i 本地；R^i_{rand} 包含从 $\cup_{i=1}^{N} R_3^i$ 中均匀随机抽样得到的元组 t 的随机份额；R^i_{repl} 与 $\cup_{i=1}^{N} R_4^i$ 相同，因为其中的任意元组都会广播到所有节点。

除了上述针对 R^i 的策略外，我们还需要相应地将 S^i 划分为互不相交的集合，得到 S_1^i、S_2^i、S_3^i、S_4^i。不同之处在于，这些集合的重分布策略应取决于它们在 R 中的对应项如何分布，即 R 中满足连接条件的那些元组如何分布。当元组$\{t|t.a = x、t \in R\}$被分布到多个节点时，S 上的对应值，即 $\{s|s.b = x、s \in S\}$，必须广播到所有节点，以确保连接结果的正确性。基于此，我们将详细说明 S_1^i、S_2^i、S_3^i、S_4^i 的重分布策略如下（运行示例见图3）：

S_1^i：这部分由元组 $t \in S^i \backslash S_2^i \backslash S_3^i \backslash S_4^i$ 组成。它们经过哈希重分布。

S_2^i：这部分包含元组 $t \in S^i$，其中 a 中的条目存在部分倾斜，位于 R。它们会广播到所有节点。S_3^i：第三组包含元组 $t \in S^i$，其中 $t.a \in \rho(C+)(R,\ S)$

它们被广播到所有节点。S_4^i：其中包含元组 $t \in S^i$，满足 $t.a \in \rho(C-)(R,S)$。它们是随机分布的。

最后，每个节点 i 将获得 S 的三个份额，即 S^i_{rand}、S^i_{repl} 和 S^i_{hash}。具体而言，S^i_{rand} 是通过随机分发 S_4^i（这些数据来自所有节点）得到的份额，也就是说，其中包含满足 $t.a \in \rho(C-)(R,S)$ 的元组 t；S^i_{repl} 是广播 $S_2^i \cup S_3^i$ 后得到的元组；S^i_{hash} 是从所有 $S_1^i (i = 1, \ldots, N)$ 哈希重分布得到的元组。图4展示了**步骤1**后的一个示例。

步骤2，连接。重新分配后，每个节点从 R 接收四组元组，从 S 接收三组元组。随后，每个节点都必须在本地执行以下连接，以获得正确结果：

$$\mathcal{R}^i_{hash} \underset{a=b}{\bowtie} \mathcal{S}^i_{hash}, \mathcal{R}^i_{loc} \underset{a=b}{\bowtie} \mathcal{S}^i_{repl}$$

$$\mathcal{R}^i_{rand} \underset{a=b}{\bowtie} \mathcal{S}^i_{repl}, \mathcal{R}^i_{repl} \underset{a=b}{\bowtie} \mathcal{S}^i_{rand}$$

第一个连接对 R 和 S 中的非倾斜元组执行哈希连接。在第二和第三个连接中，尽管 R^i_{loc} 和 R^i_{rand} 中的元组分布在第 i 个节点上，S 仍会广播可连接的元组，使每个节点都收到相同的 S^i_{repl}。因此，有效结果绝不会出现在第二和第三个连接的输出中。与上述情况类似，第四个连接也不会遗漏任何结果。因此，上述四个连接包含了 $R \underset{R.a=S.b}{\bowtie} S$ 的所有有效结果。

步骤 3，并集。最终结果由所有节点执行的四次连接的结果并集而成。

总之，PnR 的伪代码如算法1所示。第 4 至 11 行描述了重分发阶段。第 4 至 5 行分别初始化 R^i，S^i。第 7 行相应地将 R^i，S^i 划分为多个分区。第 8 至 11 行针对不同分区实施不同的重新分配策略。第 13 至 21 行对应连接阶段。第 23 行整合来自所有节点的结果。

3.3 局限性

在 PnR 算法中，我们假设探测表中的数据量远大于构建表中的数据量，因此只处理探测表中的倾斜值，而忽略构建表中的倾斜值。尽管这一假设很常见，也符合大多数实际场景，但仍存在以下例外情况：

(1) $|R| \approx |S|$, $\rho(R) = \emptyset$。在这种情况下，$\rho(S)$ 不会被处理，PnR 会退化为 GraHJ。相比之下，PRPD 表现良好，因为 $\rho(S)$ 保留在每个节点的本地。(2) $|R| \approx |S|$, $\rho(R) \cap \rho(S) = \emptyset$。类似地，PnR 仍不会处理 $\rho(S)$。在某些情况下，这会导致重新分配后各节点的计算负载极不均衡。

一般而言，PnR 不会考虑 S 中的部分倾斜元组，这可能导致重新分配后各节点的计算负担不均衡。随着 S 和 R 的规模逐渐接近，这种不均衡会更加显著。

算法1 PnR 算法

输入： R； S；偏斜值集合 $\rho(R)$ $\rho(S)$；

输出： PnR 的结果集 *result*

1: $result \leftarrow []$
2: **对于**节点 i，在 n 个节点中**执行**
3: // 步骤1，重新分配。
4: $R^i \leftarrow R$ 位于第 i 个节点上
5: $S^i \leftarrow S$ 位于第 i 个节点上
6: // 分区(R^i, S^i)指的是第 3.2 节中的步骤1
7: $R_1^i\ R_2^i\ R_3^i\ R_4^i\ S_1^i\ S_2^i\ S_3^i\ S_4^i \leftarrow$ 分区(R^i, S^i)
8: 哈希重分布 R_1^i, S_1^i
9: 保留在本地 R_2^i
10: 随机分发 R_3^i, S_4^i
11: 广播 R_4^i, S_2^i, S_3^i
12: // 步骤2，连接。
13: $\mathcal{R}_{hash}^i \leftarrow \{t | hash(t.a) \bmod N = i,\ t \in \cup_{i=1}^N R_1^i\}$
14: $\mathcal{R}_{loc}^i \leftarrow R_2^i$
15: $\mathcal{R}_{rand}^i \leftarrow$ 从 $\cup_{i=1}^N R_3^i$ 中均匀采样的元组 t
16: $\mathcal{R}_{repl}^i \leftarrow \cup_{i=1}^N R_4^i$
17: $\mathcal{S}_{rand}^i \leftarrow$ 从 $\cup_{i=1}^N S_4^i$ 中均匀采样的元组 t
18: $\mathcal{S}_{repl}^i \leftarrow \cup_{i=1}^N (S_2^i \cup S_3^i)$
19: $\mathcal{S}_{hash}^i \leftarrow \{t | hash(t.a) \bmod N = i, t \in \cup_{i=1}^N S_1^i\}$
20: $result^i = (\mathrm{R}^i{}_{hash} \underset{a=b}{\bowtie} \mathcal{S}_{hash}^i) \cup (\mathrm{R}^i{}_{loc} \underset{a=b}{\bowtie} \mathcal{S}_{repl}^i)$
21: $result^i = result^i \cup (\mathrm{R}_{rand}^i \underset{a=b}{\bowtie} \mathrm{S}_{repl}^i) \cup (\mathrm{R}_{repl}^i \underset{a=b}{\bowtie} \mathrm{S}_{rand}^i)$
22: // 步骤 3，并集。
23: $result = result \cup result^i$
24: **结束循环**
25: **返回** *result*

4 KaiwuDB 中的自适应解决方案

针对分布式哈希连接的三种代表性解决方案，即 PRPD、PnR 和 GraHJ，上述理论研究表明，没有一种算法能在所有场景中都表现最佳。通常，当连接表划分均匀且倾斜数据分布均匀时，PRPD 是最优方案。PnR 同时考虑网络流量和连接结果的倾斜程度，在大多数倾斜工作负载下表现良好，但第 3.3 节讨论的场景除外。当大量倾斜元组被分配到网关节点（即最初接收查询的节点），且所有结果都需要汇集到该节点时，GraHJ 是最优方案。鉴于上述发现，为了更好地解决 KaiwuDB 中分布式哈希连接的倾斜问题，我们希望开发一种机制，使其能够在运行时针对特定工作负载自适应地选择最优方案。在本部分中，我们将先通过一些具体示例说明这一现象，并将不同的解决方案建模为附加到哈希连接的子算子，即 PRPD、PnR 和 GraHJ。此外，我们还提出了一种基于代价的子算子调度器，该调度器原生嵌入 KaiwuDB 的代价模型和优化器中。

4.1 没有一种方案适用于所有情况

为说明不存在一种解决方案能适用于所有场景这一问题，我们将考察三个具有代表性的案例。在每个案例中，我们将研究不同解决方案的时间复杂度（以待处理元组的数量衡量），并在之后展示相应的实证结果。为简单起见，假设 R 中包含一个倾斜值 x，而 $sel(R, x)$ 和 $sel(S, x)$ 分别表示 x 在 R 和 S 中的选择率。

情况 1. $sel(S, x) > 0$ **和** $sel(R, x) = 0$。根据以下内容，PRPD 在这种情况下更优。

GraHJ. 最慢节点需要处理的元组数量可估算为 $|S|sel(S, x) + \frac{|R|}{N}$。**PnR.** 它退化为 GraHJ，其复杂度与 GraHJ 相同。**PRPD.** 最慢节点的复杂度可估算为 $\max\{|S^i|\}sel(S, x) < |S|sel(S, x)$ $+ \frac{|R|}{N}$。显然，$\max\{|S^i|\}sel(S, x) < |S|sel(S, x)$。

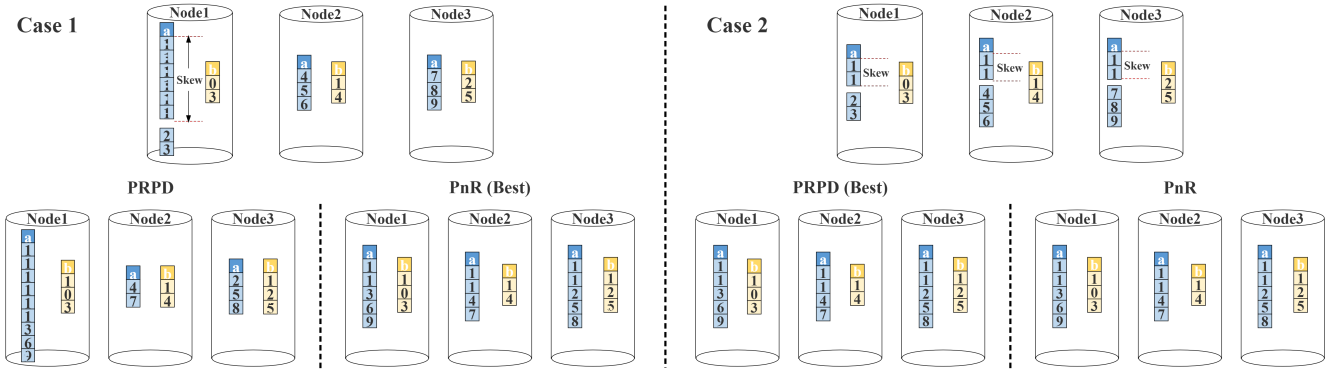
情况 2. $sel(S, x) > 0$ **和** $sel(R, x) > 0$。根据以下内容，PnR 在这种情况下更优。

GraHJ。最慢节点的时间复杂度可估算为 $|S|sel(S, x) + |R|sel(R, x) + |S|sel(S, x) \cdot |R|sel(R, x)$，其中 $|S|sel(S, x) \cdot |R|sel(R, x)$ 是物化时间。PnR。最慢节点的复杂度为 $\max\{|S^i|\}sel(S, x) + |R|sel(R, x)\frac{N}{N} + \max\{|S^i|\}sel(S, x) \cdot \frac{|R|sel(R, x)}{N}$。PRPD。复杂度可估算为 $\max\{|S^i|\}sel(S, x)|R^i|sel(R, x) + |S^i|sel(S, x)|R^i|sel(R, x) + \max\{|S^i|\}sel(S, x) \cdot |R^i|sel(R, x)$。

其他情况。 在上述两种情况下，我们都假设倾斜元组保留在网关节点上。否则，网关节点会生成大量结果。在这种情况下，GraHJ 算法将大幅降低网络开销，因此更具优势。

图 5 展示了在不同数据分布场景下，PnR 和 PRPD 的重新分配方案。为简化起见，表 R 和 S 的其他列未显示，只显示连接列，即 也就是 $R.a$ 和 $S.b$。在这两种场景中，都存在一个部分倾斜值，即 也就是 1。在情况 1 中，倾斜值 1 全部位于节点 *1* 上。按照 PRPD 的策略，这些值必须保留在节点 *1* 上。相比之下，按照 PnR 的方案，部分倾斜值（即也就是 1）的条目必须在集群中均匀重新分配，这样工作负载比采用 PRPD 时更加均衡。在情况 2 中，连接发生前，部分倾斜值（即也就是 1）已经均匀分布在三个节点上。无论采用 PRPD 还是 PnR，这些对应条目都会保留在原节点本地。因此，PRPD 和 PnR 下的工作负载都均衡，唯一的区别是，倾斜值 1 不会在 PRPD 中带来额外的网络开销。

除了上述理论分析和运行示例外，我们还据此进行了全面的实验。我们在 KaiwuDB 中实现了 PRPD、PnR 和 GraHJ，并在每个节点配备 Intel Xeon Gold 6240 和 64GB 内存的三节点集群平台上复现了上述三种场景。实验结果如图 6 所示。结果表明，在第一种情况下，PnR 比 PRPD 和 GraHJ 分别快 10% 和 121%。在第二种情况下，PRPD 比 PnR 和 GraHJ 分别快约 16% 和 28%。在前文讨论的最后一种情况下，GraHJ 比 PnR 和 PRPD 分别快 15% 和 13%。



Fig图 5：运行示例展示了不同倾斜场景下 PRPD 与 PnR 的差异 **s**

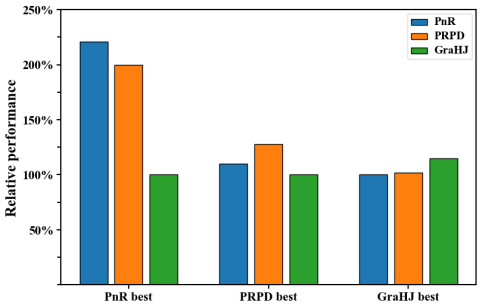


图 6：不同场景下三种算法的性能比较

4.2 自适应策略

由于 GraHJ、PRPD 和 PnR 中的任何单一算法都无法适用于所有工作负载，因此我们提出了一种基于成本的调度器，可根据目标工作负载的特征自动选择最佳方案。众所周知，主流关系型数据库管理系统都采用基于成本的优化器（CBO），根据 SQL 查找查询执行计划（ qep ）。表统计信息用于估算每个候选计划的成本，CBO 则选择成本最低的计划。例如，在 KaiwuDB 中，查询执行计划可能会采用分布式哈希连接（ Dist-HJ）；其他连接备选方案（即 嵌套循环连接、排序归并连接）的成本可能高于 GraHJ。结合上一小节的见解，考虑到 GraHJ 是分布式哈希连接的一系列备选实现之一，关于 分布式哈希连接算子的成本估算不再仅依据 GraHJ 的成本，而是取三种备选方案（即 PRPD、PnR 和 GraHJ）中的最低成本。基于这一思路，我们提出将分布式哈希连接的不同实现视为子算子，每个子算子都独立参与分布式哈希连接的成本估算。显然，查询执行计划将采用成本最低的子算子。为了估算每个子算子的成本，我们需要以下信息：

- 对于 R 和 S，表读取器的分布情况表明了参与连接的节点，以及每个节点访问的数据量即 R^i， S^i。
- 连接列的频率统计信息，即倾斜值 x。

我们将分布式哈希连接的执行概括为三个阶段，即重新分配、连接和合并。每个阶段都对应一项开销，我们的基于成本的调度器会比较这三项开销，以决定选择哪种具体算法。接下来，我们将详细说明各个阶段：

重新分配。 在哈希连接中，较小的表 S 用于构建哈希表，而较大的表 R 则相应地进行探测。只有当 S 的所有元组都完成重新分配后，探测阶段才能开始。当R 的最后一个批次完成探测时，探测阶段结束。重新分配会影响连接阶段的开始和结束时间，并且在分布式哈希连接的执行时间中占据最大的份额。不同的重分布计划会导致通过网络传输的数据量不同。值得注意的是，在所有方案（即PRPD、PnR 和 GraHJ）中，非倾斜元组的重新分配方式相同；通过网络交换的倾斜元组数量则差异很大。因此，我们使用通过网络交换的倾斜元组量来衡量每个子算子在分布式哈希连接中重新分发阶段的成本关于[2]。

连接。 当元组完成重分布后，每个节点都会单独执行哈希连接。我们将此阶段的成本分为三部分：构建、探测和物化。扫描较小的表以构建哈希表，再对较大的表进行探测以查找匹配结果。完成这两个阶段后，连接产生的结果会被物化，以供后续处理器执行。物化成本取决于连接结果的规模。最后，将这三部分的成本相加。请注意，哈希连接会在所有 N 节点上并行执行，因此我们取所有节点中的最大成本作为此阶段的成本。

合并。 合并是指每个节点上的本地连接结果节点随后会在集群中传输，这取决于连接结果的最终分布以及后续执行计划。由于先前的重新分配以及各节点上不同的连接选择率，各节点上的连接结果数量各不相同。无论优化器选择哪种算法，连接结果的总数量都是固定的。

[2]由于此数据交换的耗时会因网络设备而大幅变化，我们选择仅比较数据量，因为在网络设备固定时，数据量能直接反映耗时。

根据查询执行计划中的后续算子，我们将其分为两类。第一类中，连接结果由一个网关节点汇集，作为最终结果。为此，各节点上的本地连接结果会被物化并发送到网关节点。请注意，网关节点上的结果无需通过网络传输。第二类中，连接结果保留在本地，并作为后续算子的中间结果进行处理。例如，考虑以下查询，

选择 计数(*) 来自 R , S 条件 R . a =S . b ;

连接结果经过聚合，因此无需全部发送到网关节点。请注意，后续算子不属于分布式哈希连接，因此计算合并步骤的开销时不应将其开销计入。开销按节点计算，然后求和。具体而言，对于第一类后续算子，合并的开销取决于非网关节点上的连接结果总数；对于第二类后续算子，连接结果保留在本地，因此合并的开销为 0。因此，合并的开销取决于查询。

事实上，对于非倾斜元组，这三个子算子的执行方式相同，因此我们只需研究这些倾斜值所产生的开销差异。

4.3 子算子的代价模型

根据上述研究，连接步骤和合并步骤的开销取决于重分布计划。因此，在深入探讨子算子的代价模型之前，我们先研究每种重分布策略的特性。

通常，在所有子算子中，元组只能通过四种方式重新分布，

i) 哈希重分布；ii) 保留在本地；iii) 随机重分布；iv) 广播。

一旦确定某个表中连接列值为 entry x 的元组的重分布策略，根据我们在第 3 节中的研究，另一个表中相关元组（其连接列值为 entry x）的策略也就随之确定。

给定一个值 x，使用第一种方法时，当一个表中值为 x 的元组 t 进行哈希重分布后，另一个表中的相关元组也会进行哈希重分布。例如，假设 $t \in R$ 和 $t.a = x$，一旦 t 被哈希重分布到某个节点 i，那么这些元组 $t\prime \in S$ 和 $t\prime.b = x$ 也必须重新分布到同一个节点 i，以确保结果的正确性。

使用第二种方法时，一个表中值为 x 的元组保留在本地，相应地，另一个表中的元组则进行广播。例如，假设 $t \in R$ 和 $t.a = x$，一旦 t 保留在本地，这些元组 $t\prime \in S$ 和 $t\prime.b = x$ 就可以进行广播。

在第三种方法中，值为 x 的元组会随机重分布到所有节点，与之对应，另一张表中的元组则会被广播。例如，假设 $t \in R$ 和 $t.a = x$，一旦 t 被随机重分布，那么这些元组 $t\prime \in S$ 和 $t\prime.b = x$ 就应当被广播。

为便于讨论，我们将对表 R 中连接列条目为 x 的元组执行的重新分配操作 i)、ii) 和 iii) 分别记为 $M_h(R, x)$、$M_l(R, x)$ 和 $M_r(R, x)$。对于

例如，M_l R x $t \in$ $t|t \in$ R $t.a = x$ (，) 表示采用某种方法使元组 {，} 保留在本地；根据 ii)，S 中相应的元组则会被广播。值得注意的是，对于涉及 R 和 S 的连接任务，元组可以采用上述任一种重新分配策略，而该连接任务的整体重新分配计划可以描述为一组重新分配方法，每种方法适用于元组的一个子集。例如，GraHJ 的重新分配计划可以描述为 $\forall x \in R.a$，$M_h(R, x)$ 和 $\forall x \in S.b$，$M_h(S, x)$。

为方便起见，我们用 $Q(R, x)$和 $Q(S, x)$分别表示 $R.a$和 $S.b$中值为 x的元组数量。$Q(R^i, x)$和$Q(S^i, x)$分别表示第 i个节点上 R和 S中值为 x的元组数量。我们将分别从 Dist-HJ 的三个阶段，即重新分配、连接和合并，研究每个子算子的开销。

重新分配。如上所述，重新分配策略有三种，即 M_h，M_l，M_r。

对于 M_h，假设元组 t（其 $t.a = x$）被分配到第 i个节点，即 $hash(x) \bmod N = i$。所有 Tablereader 中值为 $t.a = x$ 的元组 t 都会进行哈希重分布，但第 i个节点上的元组除外。值为 x 的重分布元组在网络上的数据量记为 $|M_h(x)| = Q(R, x) + Q(S, x) - Q(R, x) + Q(S, x)N$，它直接反映了 M_h的开销。

对于 M_l，当来自 R^i、值为 x 的元组保留在本地时，来自 S^i、值为 x 的元组会被广播。在这种情况下，体积为 $Q^i(S, x)(N-1)$ 的元组会通过网络发送。值为 x、通过网络发送的元组总数为 $|M_l(R, x)| = Q(S, x)(N-1)$；而当来自 S^i、值为 x 的元组保留在本地时，则为 $|M_l(S, x)| = Q(R, x)(N-1)$。

对于 M_r，假设来自 R^i、值为 x 的元组以 $\frac{1}{N}$ 的均匀概率随机分配到所有节点。因此，第 i 个节点将通过网络发送 $Q^i(R, x)(N-1)N$ 个元组；来自 S^i、值为 x 的元组会被广播，其体积可建模为 $Q^i(S, x)(N-1)$。总之，在 M_r 条件下，通过网络发送的元组体积为 $|M_r(R, x)| = Q(R, x)(N-1)N + Q(S, x)(N-1)$；而当来自 S^i、值为 x 的元组随机重分布时，则为 $|M_r(S, x)| = Q(S, x)(N-1)N + Q(R, x)(N-1)$。

请注意，对于单个连接值（例如，x），每种重分布策略的开销都以成对的形式建模，其中包含两张表各自的重分布元组数。这些策略以混合方式应用于每个子算子，即也就是 GraHJ、PRPD 和 PnR，具体如下。

A) GraHJ。在 GraHJ 中，倾斜元组会进行哈希重分布。GraHJ 的重分布成本可估算为

$$Re_{GraHJ} = \sum_{x \in \rho_a^p(R) \cup \rho_b^p(S)} |M_h(x)| \tag{1}$$

B) PRPD。在此处，倾斜元组始终保留在本地，因此重分布成本为

$$Re_{PRPD} = \sum_{x \in \rho_a^p(R)} |M_l(R, x)| + \sum_{x \in \rho_b^p(S)} |M_l(S, x)| \tag{2}$$

C) PnR。执行 PnR 后，得到最终的 R_{hash}^i、R_{loc}^i、R_{rand}^i 和 R_{repl}^i。R_{loc}^i 保留在本地，R_{rand}^i（分别为 S_{rand}^i）则随机重新分配。因此，

PnR 的重分布成本可估算为

$$Re_{PnR} = \sum_{x \in \mathcal{R}_{loc}} |M_l(R,x)| + \sum_{x \in \mathcal{R}_{rand}} |M_r(R,x)| + \sum_{x \in \mathcal{S}_{rand}} |M_r(S,x)| \tag{3}$$

连接。根据 [4], 提出的广为采用的公式，我们估算第 i 个节点上的连接成本，即 $Comp^i$，并

$$Comp^i = |\mathcal{R}^i|_{est} + |\mathcal{S}^i|_{est} + |\mathcal{R}^i \bowtie \mathcal{S}^i|_{est} \tag{4}$$

$|\mathrm{R}^i|_{est}$ 和 $|\mathrm{S}^i|_{est}$ 分别是重新分配到 $node_i$ 的 R 和 S 中倾斜元组数量的估计值，两者之和反映了节点 i 上构建阶段和探测阶段的成本。$|\mathrm{R}^i \bowtie \mathrm{S}^i|_{est}$ 是倾斜元组连接所产生结果数量的估计值，用于估算连接结果的物化成本。$Comp^i$ 因采用的重新分配计划不同而在各子算子之间有所差异。值得注意的是，无论采用哪种具体的子算子，都可以根据倾斜值的统计信息预测 $|\mathrm{R}^i|_{est}$ 和 $|\mathrm{S}^i|_{est}$。

接下来，我们将研究重新分配后值为 x 的元组分布情况。

对于 M_h，值为 x 的元组会通过哈希重新分配到第 i 个节点，即 $hash(x) \bmod N = i$。第 i 个节点上值 x 的计算成本为 $|M_h(x)| = Q(R, x) + Q(S, x) + Q(R, x)Q(S, x)$。

对于 M_l，来自 R^i、值为 x 的元组保留在本地，来自 S^i、值为 x 的元组则进行广播。因此，来自 x R i $|M_l^i\ R\ x\ | = Q^i\ R\ x\ +$、值为该 $Q\ S\ x\ + Q^i\ R\ x\ Q\ S\ x$ $|M_l^i\ S\ x\ | = Q\ R\ x\ + Q^i\ S\ x$ 值的元组在第 - 个节点上的计算成本为 (,) (,) (,) (,) (,) 1。类似地，(,) (,) (,) $Q\ R\ x\ Q^i\ S\ x$ (,) (,)。

对于 M_r，来自 R^i、值为 x 的元组以 $\frac{1}{N}$ 的均匀概率随机分配到各个节点。同时，来自 S^i、值为 x 的元组进行广播。显然，根据此方法，每个节点针对值为 x 的元组的成本都相同。因此，来自 R、值为 x 的元组在每个节点上的计算成本为 $|M_r(R,x)| = {}^{Q}(R,x)_{N} + Q(S,x) + \underline{\ {}^{Q}(R,x)\ }_{N}\ \underline{\ Q(S,x)\ }$。类似地，$|M_r(S,x)| = {}^{Q}\underline{\ (S,x)\ }_{N} + Q(R,x) + \underline{\ {}^{Q}(S,x)\ }_{N}\ \ Q(R,x)$。

接下来，我们总结并正式说明三个子算子的计算成本建模方式：

A) GraHJ。在 GraHJ 中，我们将 R和 S中的所有倾斜值（包括部分倾斜值和完全倾斜值）划分为 N组，并将 $R_i \bigcup S_i$定义为第 i组。值 $x \in R_i \bigcup S_i$按照 $hash(x) \bmod N = i$分配。因此，第 i个节点的计算成本为 $\sum_{x \in R_i \bigcup S_i} |M_h(x)|$。总之，GraHJ 的计算成本估算为

$$Comp_{GraHJ} = \max_i \sum_{x \in R_i \bigcup S_i} |M_h(x)| \tag{5}$$

B) PRPD。这里，$\rho_a^p(R)$和 $\rho_a^p(S)$始终保留在本地，因此 PRPD 的计算成本为

$$Comp_{PRPD} = \max_i \left(\sum_{x \in \rho_a^p(R)} |M_l(R^i,x)| + \sum_{x \in \rho_a^p(S)} |M_l(S^i,x)| \right) \tag{6}$$

C) PnR。根据 PnR，R^i_{loc} 保留在本地，R^i_{rand}（分别是，S^i_{rand}）随机重分布。因此，PnR 的连接成本可以估算为

$$Comp_{PnR} = \max_i \sum_{x \in \mathcal{R}_{loc}} |M_l^i(R,x)| + \sum_{x \in \mathcal{R}_{rand}} |M_r(R,x)| + \sum_{x \in \mathcal{S}_{rand}} |M_r(S,x)| \tag{7}$$

合并。根据第 4.2 节，合并成本可根据通过网络传输到网关节点的元组数量建模。对于第一类执行计划，每个节点上的本地连接结果都需要汇集到网关节点，第 i个节点的合并成本为 $Summ^i = |\mathrm{R}^i \bowtie \mathrm{S}^i|_{est}$。假设网关是第 j 个节点，则总合并成本可建模为 $Summ = \sum_{i,\ i \neq j}^{N} |\mathrm{R}^i \bowtie \mathrm{S}^i|_{est}$。对于第二类无需物化全局连接结果的后续算子，其合并成本 $Summ$ 为 0。

每个子算子的总成本是三个阶段成本之和，即

$$C = Re + Comp + Summ \tag{8}$$

基于成本的调度器会计算每个子算子（即 GraHJ、PRPD 和 PnR）的成本，并选用成本最低的算子。只要能够获取各节点读取的元组统计信息以及倾斜值的统计信息，这一代价模型通常可用于主流无共享系统中的分布式哈希连接，例如 KaiwuDB。我们进一步研究了所提出的基于成本的调度器的时间复杂度。对于倾斜值 x，无论采用哪种重分布计划，成本估算本身都只需常数时间。因此，估算某个特定子算子的成本时，其复杂度与倾斜值的数量呈线性关系，假设倾斜值数量为 k，则总时间复杂度为 $O(3k)$。事实上，在后续实验中，我们发现基于成本的调度器开销极小，在实践中可以忽略不计。

算法 2 子算子调度器

输入： 构建表 S, S^i 的每个分区；探测表的每个分区 R, R^i； R, ρ_a^p 的倾斜值集合（R）； S, ρ_b^p 的倾斜值集合（S）；

输出： 最佳子算子， $selectedSubOperator$

```
selectedSubOperator ← NULL
minCost ← ∞
对于 each SubOperator ∈ {PRPD, PnR, GraHJ} 执行
    SubOperator.Init(S^i, R^i, ρ_a^p(R) ρ_a^p(S))
    Re ← SubOperator.Redistribution()
    Comp ← SubOperator.Computation()
    Summ ← 0
    若需要聚合结果则
        Summ ← SubOperator.Summarization()
    结束条件判断
    Cost ← Re + Comp + Summ
    若 Cost < minCost 则
        selectedSubOperator ← SubOperator
        minCost ← Cost
    结束条件判断
结束循环
返回 selectedSubOperator
```

算法 2 展示了基于成本的调度器的伪代码。第 3 至 11 行分别计算三个子算子的成本。第 12 至 15 行找出成本最低的特定子算子。

4.4 调度器的实现

下面介绍 PRPD 和 PnR 的实现方式，以及子算子成本模型在 KaiwuDB 中的应用方式。

算法实现。在 KaiwuDB 中，各算子通过流相互连接。流始于连接发送算子输出端的路由器，终于连接接收算子输入端的同步器。将一个 Tablereader 与每个节点上的所有 N 哈希连接器相连，路由器便用于分发 Tablereader 中的元组。KaiwuDB 默认在 Tablereader 与哈希连接器之间建立的路由器仅支持哈希重分布，即 M_h。为全面支持 PRPD 和 PnR，我们实现了一个调度器，以增强默认路由器，使其支持全部四种重分布策略：哈希、广播、保留本地和随机。调度器接收重分布策略作为参数，并根据所选方案分发元组。

代价模型。代价根据倾斜值的统计信息和 Tablereader 的统计信息计算。倾斜值的统计信息可以通过关系型数据库管理系统收集的表统计信息获取。至于第二部分，Tablereader 的统计信息可以从物理计划中推断，而物理计划是在物理规划阶段生成的。物理计划包含所有 Tablereader 的统计信息，由两部分组成：

(1) 每个 Tablereader 所驻留节点的 ID。(2) 每个 Tablereader 读取的记录集合。

参与的节点数量 N 以及每个 R^i 和 S^i 的局部性可以通过 (1) 推断。使用 (2) 可以估算 Tablereader 访问的数据量。如上一节所述，范围是按表的主键连续划分的数据块。由于范围局部性和副本选择，范围会分布到各个 Tablereader，并由其访问。每个 Tablereader 读取的数据量，即 $|R^i|$ 和 $|S^i|$，可以通过累加单个 Tablereader 读取的范围的数据量来计算。至此，代价模型所需的所有参数均已获得。

图 7 概述了在自适应调度器的支持下，KaiwuDB 如何处理查询执行计划中包含分布式哈希连接的 SQL 流水线，并在运行时为该查询选择最合适的子算子（例如，PRPD）。

5 评估

在本节中，我们进行实证研究，以测试 PnR 和自适应的基于成本的调度器。与 [21], 一致，我们采用相同的数据集设置，通过生成遵循齐夫分布的合成数据集，来测试存在倾斜的分布式哈希连接。齐夫分布 [33] 可用于模拟现实生活中许多变量的分布，例如人口收入和城市规模。遵循齐夫分布的数据集也常用于评估数据库系统的性能，尤其是数据倾斜问题 [8]。在评估用于解决数据倾斜问题的分布式哈希连接算法 [21] [14] [24] [22]时，齐夫分布总是被用于比较和验证所提出算法的性能。形式上，给定按频率排序的 n 个元素若其频率遵循偏斜因子为 z的齐夫分布，则排名第 1 的最高频元素占所有值的 $x = \frac{1}{H_{(n,z)}}$，其中 $H(n, z) = \sum_{i=0}^{n} \frac{1}{i^z}$ 是第 n 个广义调和数。排名为 r的元素出现 $\frac{x}{r^z}$ 次。此外，我们可以通过改变 z轻松调整数据集的倾斜程度。

5.1 实验设置

实验中，所有测试均在配备 Intel Xeon Gold 6240 ×2 的平台上进行，该平台共 36 个核心、内存容量为 64GB，持久化存储采用固态硬盘，网卡为 Mellanox MT27710 25Gbps。我们的自适应基于成本的调度器所实现的数据库版本为 KaiwuDB ver. 1.0（ZNBase v2.0），同时也测试了 crdb v 20.1.173。默认情况下，集群中使用 12 个数据库节点。对于数据库配置，我们尽可能使表在集群各节点间均衡分布，因此将范围大小设为最小值 65535 字节。为确保在所有测试的倾斜场景中对不同算法进行公平比较，Tablereader 的副本选择设为随机模块，每个范围均从 3 个副本（节点）中均匀随机读取 1 个副本；每种算法的性能取 10 次运行的平均值。

我们按照第 4.4 小节所述的细节，在 KaiwuDB 中实现了以下全部三个子算子以及自适应调度器。

- PnR：我们在第 3 节提出的分区与重分布。
- PRPD：部分重分布与部分复制，[29] 结合了 [21] 中提出的 SFR。
- GraHJ：[15] 中提出的并行 GRACE 哈希连接。
- 调度器：我们在第 4 节提出的自适应调度器。

与 [21], 一致，我们使用元组吞吐量测试所有基线方法的性能，该指标表示集群每秒处理的元组数量。此指标不仅能反映不同算法的相对性能，也能反映实验数据集的规模。

5.2 PnR 的性能

为了全面考察 GraHJ、PRPD 和 PnR 的性能及其优缺点，接下来我们分别通过改变连接表的大小、倾斜度或集群中的节点数量来进行实验。

改变探测表／构建表的大小。在第一个场景中，节点数设为固定值，两张表的 Zipf 参数均设为 $z = 1.2$。我们固定构建表的大小，改变探测表的大小，以评估不同算法在连接表大小相对比例方面的性能。图 8 展示了节点数不同的三组实验的吞吐量。从实验结果来看，PnR 的性能最佳且最稳定。一方面，由于实验在高速网络环境下运行，PnR 的网络开销有限，可以忽略不计。另一方面，

[3]实验设置及相应结果与我们在 KaiwuDB 中得到的结果相同。由于篇幅有限，我们仅展示 KaiwuDB 中的结果，并在 https://github.com/lihuixidian/SkewHJs 提供了基于 crdb 的实现源代码。

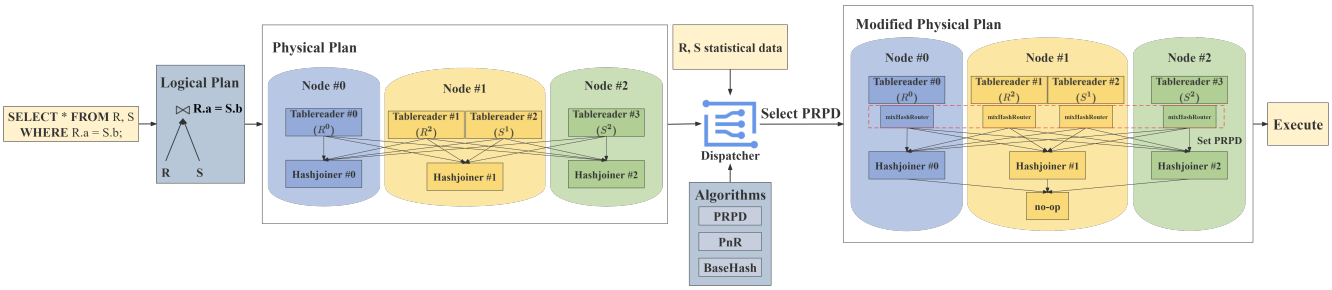
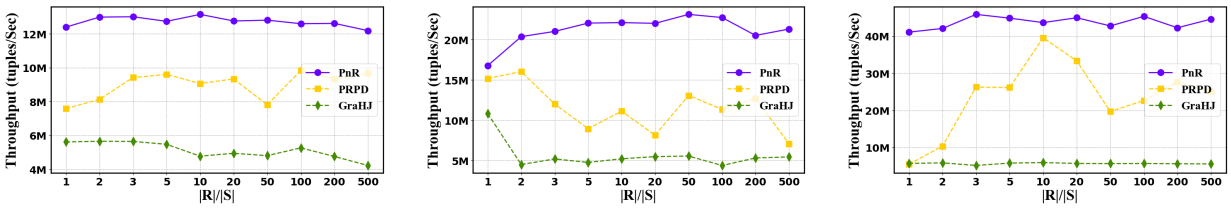
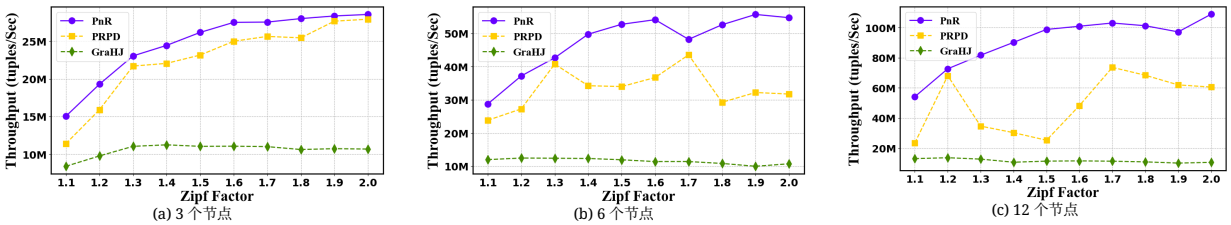


图 7：执行包含分布式哈希连接的 SQL 的整体流水线（示例展示自适应调度器找到 PRPD）



(a) 3 个节点　(b) 6 个节点　(c) 12 个节点

图 8：随着探测表与构建表数据量相对比例的增加，吞吐量的变化



(a) 3 个节点　(b) 6 个节点　(c) 12 个节点

图 9：随着探测表和构建表中的 Zipf 因子增大，吞吐量的变化 z （$|R|/|S| = 293K/10K$ *tuples*）

PnR 的随机重分布确保了其稳定性。PRPD 的性能不稳定，这是由于每次执行重新分配之前数据都存在不平衡。

倾斜度变化。图 9 比较了 Zipf 因子 z增大时三种算法的表现。随着 Zipf 因子的增大，数据集的偏斜度也随之增加。数据变得越倾斜，PnR 的性能就越好。在这部分实验中，PRPD 仍然不稳定，原因与上文讨论的相同。值得一提的是，PRPD 在 3 个节点上的性能明显比在 6 个和 12 个节点上更加稳定。为了揭示其背后的原因，我们研究了数据分布，发现集群中的节点数越少，倾斜数据的分布就越均衡。同时，实验结果也与我们在第 3 节中的讨论一致：PRPD 性能的理论上限接近 PnR，例如图 9(b) 中的 $z = 1.3$ 和图 9(c) 中的 $z = 1.2$。

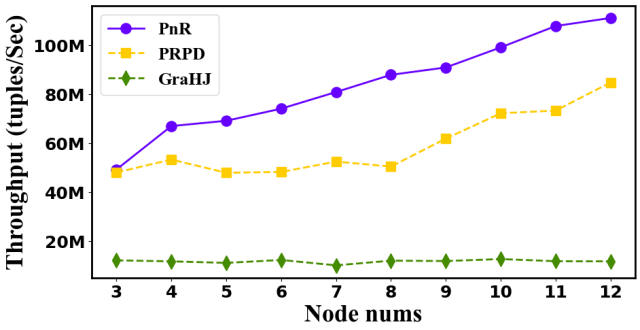


图10：增加 KaiwuDB 集群中的节点数量时的吞吐量（$z = 1.5$, $|R|/|S| = 391K/19K$ *tuples*）

改变集群规模。图10展示了集群中的节点数量增加时三种算法的可扩展性。由于所有具有相同值的倾斜元组都会分配到同一个节点，因此随着节点数量增加，GraHJ获得的加速比最小节点数量。与 GraHJ 相比，PRPD 的性能显著提升，速度可提高 4× 至 8×。PnR 的可扩展性最佳，并且在不同设置下更加稳定。

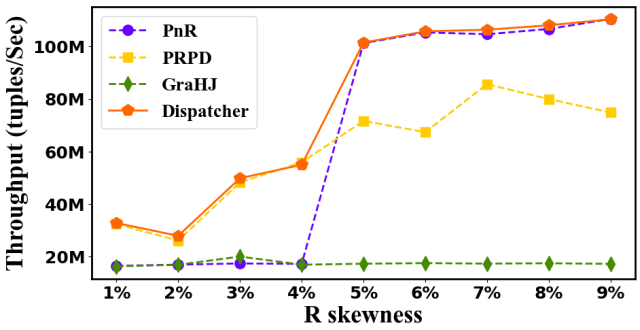


图11：随着探测表的倾斜度增加，基于成本的调度器会选择最优子算子（12 *nodes*，$|R|/|S| = 195K/147K$ *tuples*）

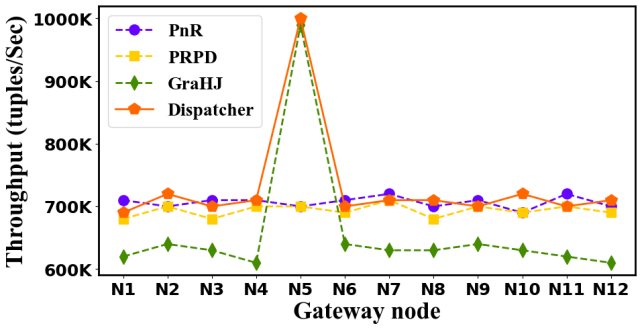


图12：不同网关节点的查询执行时间（12 *nodes*，$|R|/|S| = 195K/98K$ *tuples*）

5.3 基于成本的自适应调度器的性能

本部分通过一组实验测试我们提出的基于代价的自适应调度器的性能。具体而言，根据第 4 节的研究，PnR、PRPD 和 GraHJ 在不同场景下各有优势。除了不同场景的边界情况（此时单一算法的性能会突然变化）之外，我们的基于代价的调度器很容易找到最佳方案。为此，我们针对这些具有挑战性的任务开展测试，并将其归纳为两种不同的实验设置。在这些实验中，我们没有使用服从齐夫分布的数据集，而是采用单个倾斜值，以便更精确地控制倾斜值的频率，从而更容易观察当某个值从非倾斜变为倾斜时，不同 Dist-HJ 子算子的性能变化。

在第一组实验中，我们将频率阈值设为 $p = 5\%$，即 频率高于该阈值的值被视为倾斜值。构建表的倾斜固定为50%，探测表的倾斜则从1%逐渐增加到9%。如图11所示，当 R 的倾斜低于5%时，PnR和GraHJ的性能一致。在这种情况下，PnR退化为GraHJ，因为 R 中不存在倾斜值；而PRPD不受影响，因为它会在 S中处理倾斜值。当 R 的倾斜大于5%时，情况则相反，即PnR的性能优于PRPD。不过，无论是PnR优于PRPD，还是反之，调度器始终表现最佳，这表明它在所有情况下都能正确找出最佳子算子。

在第二种实验设置中，我们评估在不同网关节点上提交相同 SQL 时的查询耗时。在此场景中，结果大小为 $18M$，需要由网关节点汇总。如图12所示，当所有倾斜元组都重新分布到网关节点时，查询耗时比非网关节点快53%。基于成本的调度器能够识别成本最低的最优子算子，性能优于其他算法。

基于成本的自适应调度器在单一倾斜工作负载下表现良好。在齐夫分布等其他工作负载中，也观察到了类似结果。此外，倾斜阈值决定了可捕获的倾斜值数量，因此会影响调度器作出的决策。PnR 和 PRPD 同样依赖倾斜统计信息，因此自适应调度器倾向于选择 PnR 或 PRPD，其性能至少不逊于 GraHJ。

6 结论与未来工作 K

鉴于倾斜数据分布给分布式哈希连接任务中的负载均衡带来了巨大挑战，我们针对不同倾斜设置，对一系列现有分布式哈希连接策略进行了大量实证测试与比较。我们提出了一种新的分布式哈希连接策略，即 PnR，适用于一组具有代表性的场景，在这些场景中，GraHJ 和 PRPD 的表现都不理想。全面的研究表明，没有一种策略能够在所有设置下都胜出。因此，我们提出了一个基于代价的子算子分发器，并将其嵌入代价模型和查询优化器中，使其能够在运行时自适应地找到最合适的策略。我们在 crdb 和 KaiwuDB 中都实现了这一自适应调度器，实证结果表明，我们的方案在所有测试设置下均表现最佳。值得注意的是，第 4 节末尾介绍的当前自适应调度器方案及其相应代价模型依赖于对探测表和构建表收集的统计信息；这些统计信息可以来自存储的原始关系，也可以来自经原子谓词筛选后的关系。执行过程中生成的中间表和临时表，即，由前序连接任务产生的临时结果，目前尚不受我们的方案支持。为解决这一问题，需要开发运行时采样器和估算器，以便为自适应调度器及代价模型的输入提供估算；这也是我们在 KaiwuDB 中未来工作的重要组成部分。

致谢

感谢Gene Fu博士在本项目中给予的宝贵建议。本工作部分得到XD-Inspur数据库创新实验室基金和国家自然科学基金委员会（编号：61972309、62272369）的资助。

参考文献

[1] Khaled Alsabti和Sanjay Ranka。2001。关系连接的偏斜不敏感并行算法。《沙特国王大学计算机与信息科学杂志》13（2001），79–110。[2] R. Blumofe和C. E. Leiserson。1999。通过工作窃取调度多线程计算。《*ACM*学报》（*JACM*）（1999）。[3] Wei Cao、Feifei Li、Gui Huang、Jianghang Lou、Jianwei Zhao、Dengcheng He、Mengshi Sun、Yingqiang Zhang、Sheng Wang、Xueqiang Wu、Han Liao、Zilin Chen、Xiaojian Fang、Mo Chen、C henghui Liang、Yanxin Luo、Huanming Wang、Songlei Wang、Zhanfeng Ma、Xinjun Yang、Xiang Peng、Yubin Ruan、Yuhui Wang、Jie Zhou、Jianying Wang、Qingda Hu和Junbin Kang。2022。PolarDB-X：面向云原生应用的弹性分布式关系数据库。发表于第*38*届*IEEE*国际数据工程会议（*ICDE 2022*），马来西亚吉隆坡，*2022*年*5*月*9*日至*12*日。IEEE，2859–2872。[4] David J DeWitt、Jeffrey F Naughton、Donovan A Schneider和Srinivasan Se- shadri。1992。并行连接中的实用偏斜处理。技术报告。威斯康星大学麦迪逊分校计算机科学系。[5] Siying Dong、Mark Callaghan、Leonidas Galanis、Dhruba Borthakur、Tony Savor和Michael Strum。2017。优化 RocksDB 中的空间放大。发表于第八届两年一度的创新数据系统研究会议（*CIDR 2017*），美国加利福尼亚州查米纳德，*2017*年*1*月*8*日至*11*日，在线论文集。www.cidrdb.org。[6] Manuel Dossinger和Sebastian Michel。2021。优化多个多路流连接。发表于*2021*年*IEEE*第*37*届国际数据工程会议（*ICDE*）。IEEE，1985–1990。[7] Ronald L. Graham。1969。多处理时序异常的界。《*SIAM*应用数学杂志》17，2（1969），416–429。[8] Jim Gray、Prakash Sundaresan、Susanne Englert、Ken Baclawski和Peter J Weinberger。1994。快速生成十亿条记录的合成数据库。发表于*1994*年*ACM SIGMOD*数据管理国际会议论文集。243–252。[9] Lilian Harada和Masaru Kitsuregawa。1995。无共享数据库系统中哈希连接的动态连接结果倾斜处理。发表于高级应用数据库系统 *'95*：第四届高级应用数据库系统国际会议（*DASFAA*）论文集，新加坡，*1995*年*4*月*11*日至*13*日，5 246 255-卷。–。[10] Z. He、Z. Li、X. Peng和C. Weng。2021。DS 2：通过高速网络上的数据窃取处理数据倾斜。发表于*2021*年*IEEE*第*37*届国际数据工程会议（*ICDE*）。[11] Kien A Hua和Chiang Lee。1991。使用分区调优处理多处理器数据库计算机中的数据倾斜。发表于*VLDB*，第91卷。Citeseer，525–535。[12] Dongxu Huang、Qi Liu、Qiu Cui、Zhuhe Fang、Xiaoyu Ma、Fei Xu、Li Shen、Liu Tang、Yuxing Zhou、Menglong Huang、Wan Wei、Cong Liu、Jian Zhang、Jianjun Li、Xuelian Wu、Lingyu Song、Ruoxi Sun、Shuaipeng Yu、Lei Zhao、Nicholas Cameron、Liquan Pei和Xin Tang。2020。TiDB：基于Raft的HTAP数据库。《*VLDB*学报》13，12（2020），3072–3084。[13] Edward G. Coffman Jr.、M. R. Garey和David S. Johnson。1978。装箱问题在多处理器调度中的应用。《*SIAM*计算杂志》7，1（1978），1–17。[14] Masaru Kitsuregawa和Yasushi Ogawa。1990。桶扩展并行哈希：一种适用于超级数据库计算机（SDC）中数据倾斜的稳健新型并行哈希连接方法。发表于*VLDB*。210–221。[15] Masaru Kitsuregawa、Shin-ichiro Tsudaka和Miyuki Nakano。1992。全共享多处理器上的并行 GRACE 哈希连接：在 Symmetry S81 上的实现与性能评估。发表于*1992*年*2*月*3*日至*7*日，第八届国际数据工程会议论文集，美国亚利桑那州坦佩。IEEE计算机学会，256–264。[16] D. B. Larkins。2021。利用结构化原子操作优化工作窃取通信。发表于国际并行处理会议。[17] Zhenghua Lyu、Huan Hubert Zhang、Gang Xiong、Gang Guo、Haozhou Wang、Jinbao Chen、Asim Praveen、Yu Yang、Xiaoming Gao、Alexandra Wang等。2021。Greenplum：面向事务型与分析型工作负载的混合数据库。载于*2021*年国际数据管理大会论文集。2530– 2542。[18] Vasilis Mageirakos、Riccardo Mancini、Srinivas Karthik、Bikash Chandra 和 Anastasia Ailamaki。2022。面向复杂查询的高效 GPU 加速连接优化。载于*2022*年 *IEEE* 第*38*届国际数据工程会议（*ICDE*）。IEEE，3190–3193。[19] Magnus Müller、Daniel Flachs 和 Guido Moerkotte。2021。通过重数和交集大小进行内存高效的键/外键连接大小估算。载于*2021*年 *IEEE* 第*37*届国际数据工程会议（*ICDE*）。IEEE，984–995。[20] Diego Ongaro 和 John K. Ousterhout。2014。探索一种易于理解的共识算法。载于*2014*年 *USENIX* 年度技术大会，*USENIX ATC '14*，美国宾夕法尼亚州费城，*2014*年*6*月*19*日至*20*日，Garth Gibson 和 Nickolai Zeldovich（编）。305–319。[21] Wolf Rödiger 、 Sam Idicula 、 Alfons Kemper 和 Thomas Neumann。2016。Flow-Join：面向高速网络上分布式连接的自适应倾斜处理。载于第*32*届 *IEEE* 国际数据工程会议（*ICDE 2016*），芬兰赫尔辛基，*2016*年*5*月*16*日至*20*日。1194–1205。[22] Wolf Rödiger、Tobias Mühlbauer、Philipp Unterbrunner、Angelika Reiser、Alfons Kemper 和 Thomas Neumann。2014。面向并行主内存数据库集群的局部性感知算子。载于*2014*年 *IEEE* 第*30*届国际数据工程会议。IEEE，592–603。[23] Masoumeh Shafieinejad、Suraj Gupta、Jin Yang Liu、Koray Karabina 和 Florian Kerschbaum。2022。针对一系列查询的加密数据等值连接。载于*2022*年 *IEEE* 第*38*届国际数据工程会议（*ICDE*）。IEEE，1635–1648。[24] Ambuj Shatdal 和 Jeffrey F Naughton。1993。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。119–128。[25] James W. Stamos 和 Honesty C. Young。1993。一种用于分布式连接的对称分片与复制算法。*IEEE* 并行与分布式系统汇刊 4，12（1993），1345–1354。[26] Michael Stonebraker。1986。无共享架构的理由。*IEEE* 数据库工程公报 9，1（1986），4–9。[27] Rebecca Taft、Irfan Sharif、Andrei Matei、Nathan VanBenschoten、Jordan Lewis、Tobias Grieger、Kai Niemi、Andy Woods、Anne Birzin、Raphael Poss、Paul Bardea、Amruta Ranade、Ben Darnell、Bram Gruneir、Justin Jaffray、Lucy Zhang 和 Peter Mattis。2020。CockroachDB：弹性地理分布式 SQL 数据库。载于*2020*年国际数据管理大会论文集，*2020*年 *SIGMOD* 大会，在线会议 *[*美国俄勒冈州波特兰*], 2020*年*6*月*14*日至*19*日。1493– 1509。[28] Nikolaos Tziavelis、Wolfgang Gatterbauer 和 Mirek Riedewald。2022。迈向响应迅速的数据库管理系统：最优连接算法、枚举、因式分解、排序与动态规划。载于*2022*年 *IEEE* 第*38*届国际数据工程会议（*ICDE*）。IEEE，3205–3208。[29] Yu Xu、Pekka Kostamaa、Xin Zhou 和 Liang Chen。2008。处理无共享系统并行连接中的数据倾斜。载于*ACM SIGMOD*数据管理国际会议论文集，*2008*年 *SIGMOD* 大会，加拿大不列颠哥伦比亚省温哥华，*2008*年*6*月*10*日至*12*日。ACM，1043–1052。[30] Hao Zhang、Qiyan Li、Kangfei Zhao、Jeffrey Xu Yu 和 Yuanyuan Zhu。2022。学习如何助力复杂循环连接分解。载于*2022*年 *IEEE* 第*38*届国际数据工程会议（*ICDE*）。IEEE，3138–3141。[31] Hao Zhang、Miao Qiao、Jeffrey Xu Yu 和 Hong Cheng。2021。快速分布式复杂连接处理。载于*2021*年 *IEEE* 第*37*届国际数据工程会议（*ICDE*）。IEEE，2087–2092。[32] X. Zhou 和 M. E. Orlowska。1995。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*IEEE* 第一届并行处理算法与架构国际会议。[33] George Kingsley Zipf。2016。人类行为与最省力原则：人类生态学导论。Ravenio Books。