DOI：10.13718/j.cnki.xdzk.2020.12.001

一种热点敏感的自适应跳跃表

文　韬，　吉　锋，　刘丽霞

中兴通讯股份有限公司 移动网络和移动多媒体技术国家重点实验室，南京 210012

摘要：研究了利用访问局部性原理提升跳跃表查询效率的问题，同时研究了以上加速策略的自适应机制. 首先改进了跳跃表查询操作，令其额外返回在特定层次上的访问路径子集；其次利用蓄水池采样对跳跃表查询操作进行采样，然后根据采样结果对跳跃表特定层次的工作负载进行预测，根据工作负载选择热点区域，在热点区域设置加速点以提升查询效率；最后根据查询效率的提升程度和系统负载变化情况，利用 SARSA 算法和奖励塑形机制自动调节加速点层级和规模，以达到查询加速和管理成本之间的平衡. 实验证明：在访问倾斜的跳跃表应用场景下，该方法相比原生跳跃表查询算法有更低的延迟，在访问模式逐步变化的应用场景下，该方法能够随环境变化灵活调整加速策略.

关　键　词：热点敏感；跳跃表；强化学习；SARSA；奖励塑形

中图分类号：TP391　　**文献标志码**：A　　**文章编号**：1673－9868(2020)12－0001－14

随着硬件技术的发展，大数据管理系统对大容量内存的利用途径日益增多. 索引作为数据库/大数据管理系统的核心组件，从面向磁盘的 B-tree 家族到面向内存的 T-tree、CST 等，逐渐形成了规模庞大的家族体系. 跳跃表以维护简洁、执行高效的优势逐渐得到业界的关注，并广泛地被运用在数据库/大数据管理系统中(如 Redis/rocksdb/leveldb 等).

本文基于访问局部性原理，提出了一种热点敏感的自适应跳跃表. 其要点是：

① 利用跳跃表本身结构特点、设计了一种根据访问热点进行快速路由设置的加速机制；

② 设计了一种能够根据加速实际效果对以上加速机制进行调节的自适应机制，并考虑负载成本因素.

1　研究主要内容

传统的跳跃表[1]是典型的概率型数据结构，整个数据结构以多层有序链表形式呈现，其中上层结构的节点均会在下层结构的节点集中，可以认为上层节点由下层节点通过概率采样得到.

跳跃表设计目的在于对点查询和范围查询的良好支持，既可以根据查询条件中的查询起始点 Firstkey 进行点查询、又可以在定位到 Firstkey 后继续进行范围查询直到抵达终止点 Endkey；本文讨论的跳跃表查询操作(包含查询、插入、删除，以下均以 Lookup 查询操作指代)均指包含查询起始点 Firstkey 的操作.

具体查询执行过程如图 1 所示：

原生跳跃表在 Lookup 查询时，从数据结构的左上角(以下称为哨兵 Pivot)出发，向右或向下逐步跳跃直至定位到查询目标或者无功而返. 设以采样概率 p 构造的跳跃表节点规模为 N，则其查询时间复杂

收稿日期：2020－10－18

基金项目：国家自然科学基金项目(71672074).

作者简介：文　韬(1982－)，男，工程师，硕士，主要从事大数据/数据库系统与人工智能结合方面的研究.

度为 $O(\mathrm{Log}_{1/p}(N))$[1]；如果 $p=0.5$，则跳跃表查询时间复杂度与红黑树相当(本文后续都假设该概率值 $p=0.5$)为 $O(\mathrm{Log}_2(N))$.

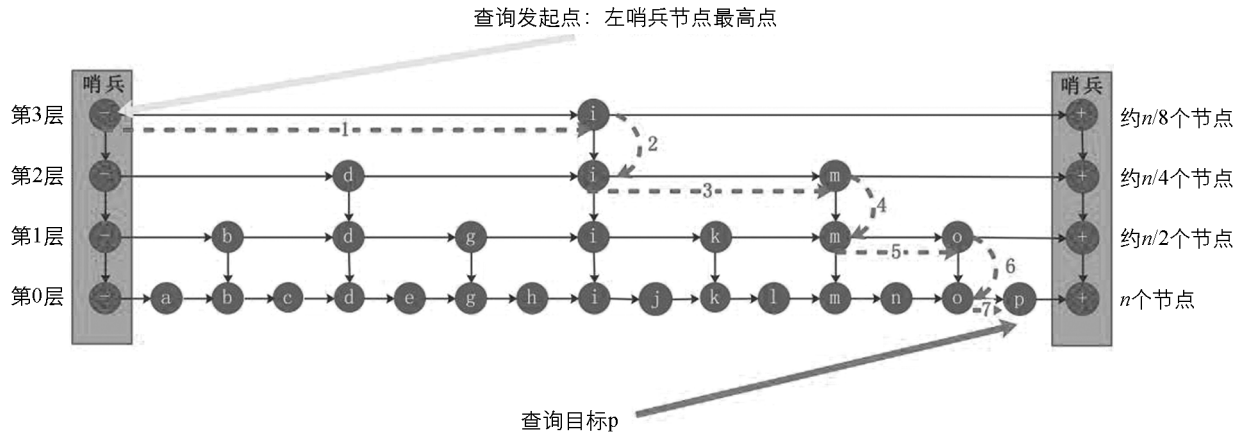

图1 原生跳跃表 Lookup 操作示意图

众所周知，大数据库管理系统对性能的渴求远未抵达终点. 对于以跳跃表为基础的内存索引来说，其加速的主要途径有两点：

其一是避免查询效率因随机因素退化. 确定性跳跃表[2]利用确定性技术替换随机概率生成方式，在更新跳跃表时同时调整目标节点附近的节点高度以维持事先定义的分布规则，单次更新影响的节点较多，不利于高并发场景的扩展；最新研究成果如文献[3]将传统跳跃表改造为上下两层分层结构：下层由多个固定的半有序区段(称为 GaurdEntity，区段间节点键值保证有序、区段内节点键值不保证有序)组成，用指向 GaurdEntity 的快速搜索结构(如 FAST[4])代替原生跳跃表上层链表，避免随机因素影响搜索效率.

其二是利用局部性原理. 假设索引访问可能存在局部性，在访问热点设置快速路由，查询从传统左上角哨兵节点转而从热点附近的快速路由入口发起，旁路不必要的跳跃过程实现访问加速. 文献[5]针对路由器管理的特定场景，将页面前缀和跳跃表节点地址分别做为哈希表的 Key 和 Value，查询时先提取前缀，通过哈希表定位到对应前缀所在跳跃表节点附近再开始查找，提升查询性能.

本文基于局部性原理设计加速机制，能够根据工作负载情况选择热点进行路由，并根据加速情况和系统负载调整路由策略(层次、规模).

本文后续章节中，第二节从宏观层面上概述了热点敏感自适应跳跃表的整体架构和工作流程，阐述了热点加速和自适应调整的相互关系；第三节具体阐述跳跃表的热点加速机制，包括与热点路由结合的跳跃表查询算法、跳跃表插入和删除算法的对应调整；第四节阐述跳跃表加速策略的自适应调整机制和跳跃表热点路由的构建算法；第五节给出相应实验和结论；第六节总结了全文工作以及提出后续工作思路.

2 自适应跳跃表宏观架构

2.1 跳跃表加速概述

对于原生跳跃表来说，每个包括查询起始点 Firstkey 条件的 Lookup 查询请求 q 抵达时，需要从左上角哨兵节点最高层往下、往右跳跃去进行查找目标.

然而对于实际工作场景而言，Lookup 访问模式往往存在局部性可供利用，即在一段时间内，大规模访问操作将会持续集中在某些区域，这就给加速提供了可能性. 在热点区域附近设置快速路由入口，对热点区域的查询访问操作从快速路由入口发起而不是左上角哨兵节点发起，以此减少往下跳跃和向右跳跃的次数，从而实现查询加速；对非热点区域的查询访问操作仍然从左上角哨兵节点发起. 以上算法如图2所示：

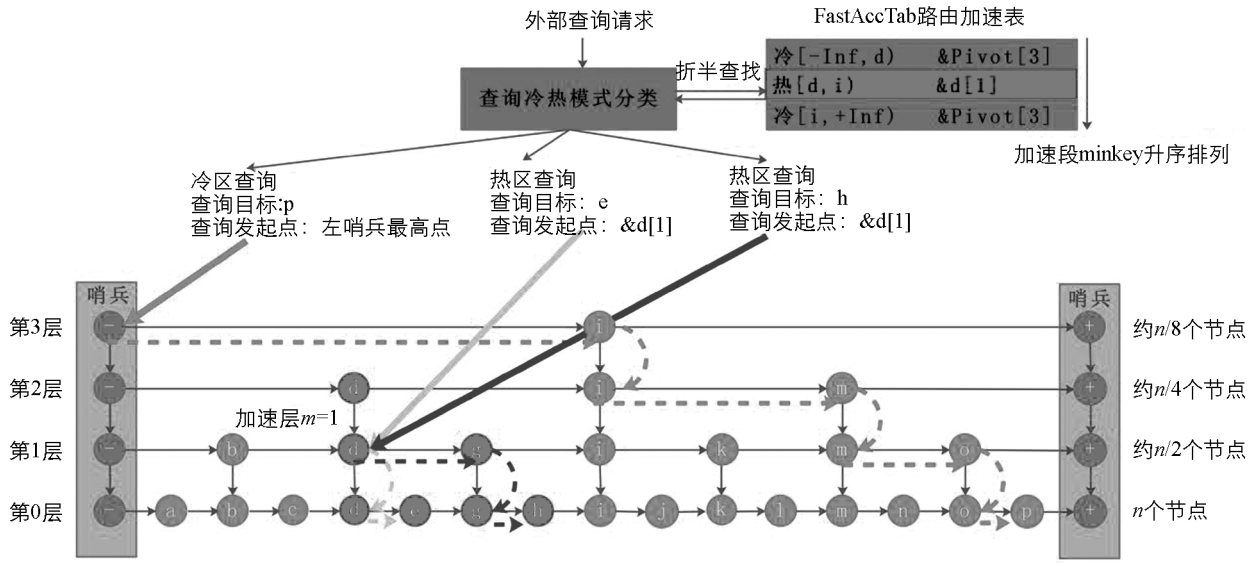

图 2　热点敏感跳跃表查询加速示意图

快速路由能够实现跳跃表查询加速，但有额外成本. 每当新查询 q 抵达跳跃表时，都必须判断其是否落在某个热点加速段内并获取对应路由入口地址. 这个路由表查询操作是范围查询，当热段和冷段的数目和是 n 时，每次路由表折半查询的时间复杂度是 $O(\mathrm{Log}_2(n))$，考虑到 n 远远小于跳跃表节点规模 N，跳跃表的时间复杂度并未变化，但路由表查询耗时并不能忽视，路由表规模不能无限扩展.

2.2　自适应跳跃表工作流程

对于原生跳跃表来说，每个包括查询起始点 Firstkey 条件的 Lookup 查询请求 q 抵达时，需要从左上角哨兵节点最高层往下、往右跳跃去进行查找目标，上述跳跃表加速技术需要数据分析的支撑，系统对跳跃表查询的执行过程进行记录，并定时分析以上查询执行信息以找到访问模式中的热点区域，刷新路由表. 其中，热点区域的设置策略并非一成不变，需要根据加速效果和系统负载情况在线学习和演化.

系统通过消息队列接收内外部消息，包括来自外部的跳跃表访问请求和来自内部定时触发的数据分析请求. 热点敏感的自适应跳跃表工作流程图如图 3 所示：

1）对于来源于外部的跳跃表访问请求

如果查询条件中包含范围起始点 Firstkey，本文将其定义为 Lookup 查询请求，后文将围绕 Lookup 查询请求进行讨论. 系统将首先利用蓄水池采样[6]技术对 Lookup 查询请求进行分流，再根据分流结果执行热点敏感的跳跃表查询算法 HotRegionLookup，该算法中包含 3 种不同的 Lookup 查询操作策略（在 3.1 节中进一步阐述以上 3 种策略）. 此外，对于跳跃表访问请求中的删除操作，HotRegionLookup 进行延迟删除（只设置删除标记、物理删除在数据分析周期内执行）.

2）对于内部定时触发的数据分析/路由表刷新请求

本文基于强化学习思想，将问题视为智能体与环境之间的互动，将环境视为外部查询访问操作的工作负载，智能体进行路由加速策略的选择. 奖励则由实际加速效果、系统负载变化、加速机制对跳跃表结构影响综合而成.

每个数据分析周期都获取最近时间段的执行过程信息、评估当前系统的状态、核算即时和额外奖励、选择下一时段的加速策略并根据此策略进行加速路由表的刷新. 在数据分析周期完成时，对不是路由点的待删除节点进行物理删除，若加速效果持续下降到达阈值，则关闭加速机制回到预热状态继续学习.

值得注意的是，由于强化学习是基于试探—奖惩的循环操作，在开始阶段往往不容易把握探索—利用之间的平衡，在本问题中将会造成不必要的性能颠簸以致影响关键系统的部署，因此本文针对强化学习系统的利用程度进行了控制.

① 初期系统工作在“预热(preheat)”模式. 此时被蓄水池采样算法选中的查询请求执行 HotRegionLookup 策略 1 或者策略 2，绝大多数其它查询请求执行 HotRegionLookup 策略 3，从哨兵节点发起查询，且不记录执行过程和耗时，对整体系统的性能影响可控，此时根据策略 1 和策略 2 的执行情况进行强化学习，学习成熟后切换为“加速”模式.

② 系统工作在“加速(fullacc)”模式. 此时被蓄水池采样算法选中的查询请求执行 HotRegionLookup 策略 1 或者策略 2，绝大多数其它查询请求执行 HotRegionLookup 策略 3，从快速路由表 FastAccTab 发起查询且不记录执行过程和耗时，充分利用强化学习的策略学习成果，此时根据策略 1 和策略 2 的执行情况继续强化学习，如果系统后续工作负载发生变化导致加速效果持续下降，则重新切换为“预热”模式.

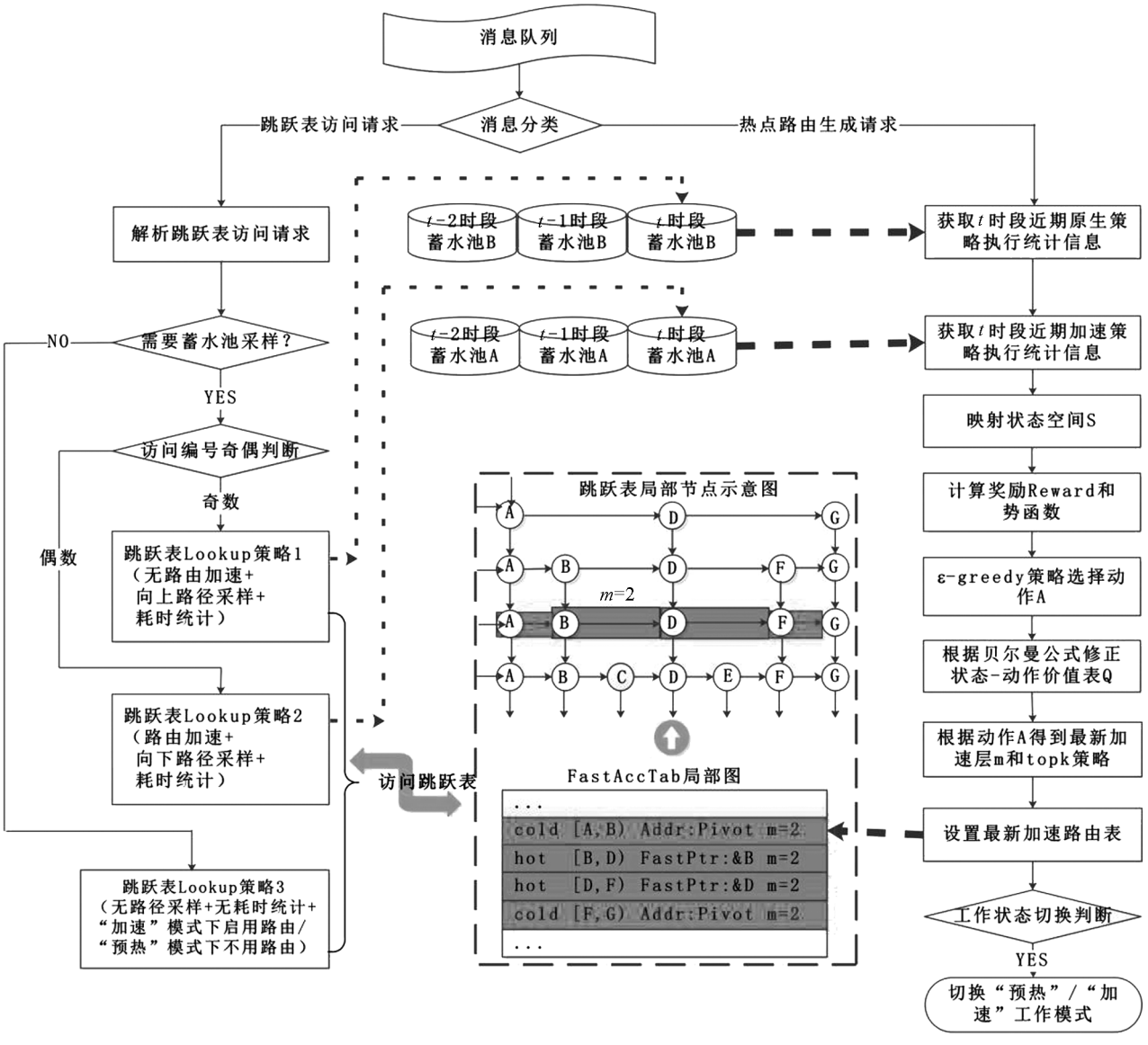

图 3 热点敏感的自适应跳跃表工作流程图

2.3 查询路径采样

由于跳跃表是大数据管理系统的关键索引结构，将某段时间内的访问操作全部记录进行分析将带来很大性能压力，因此需要利用采样技术提取访问操作记录的子集进行分析.

首先给出锚点定义：

定义 锚点 Anchor(q, k)是 Lookup 访问操作 q 途经跳跃表第 k 层的下行节点.

如果一个节点是访问操作 q 在第 k 层的锚点，设其键值为 key，其在 k 层的后继节点键值为 keynext，

根据跳跃表的访问逻辑，访问操作 q 的查询目标处于[key，keynext)区间内.

例如在图 1 中，以“p”为目标的 Lookup 查询 query 在第 3 层锚点 Anchor(query，3)的键是“i”，第 2 层锚点 Anchor(query，2)的键是“m”，第 1 层锚点 Anchor(query，1)的键是“o”. 随着查询逐渐进行，由锚点标示的搜索空间从[“i”，+Inf)到[“m”，+Inf)和[“o”，+Inf)逐步缩小.

如上节所述，本文使用蓄水池采样进行操作采样，其能够保证每个操作被等概率选中，因此热点访问区域将会在蓄水池中高频出现. 这里使用了双蓄水池 A/B 分别存储不同 HotRegionLookup 策略下的执行信息作为数据分析基础.

在本文所提出的加速方案中，加速路由点附着在第 m 层的某些符合要求的实际物理节点上，令当前加速层为 m，不同的查询执行策略中，需要保存不同的执行信息.

① 路径高采样：在 HotRegionLookup 策略 1 中，记录查询 q 的耗时，该查询路径上处于 $m+3$、$m+2$、$m+1$ 层的锚点信息，数据放入 BasePool(蓄水池 B)；

② 路径低采样：在 HotRegionLookup 策略 2 中，记录查询 q 的耗时、该查询路径上处于 m、$m-1$、$m-2$ 层的锚点信息，数据放入 AccPool(蓄水池 A).

3　热点敏感加速查询机制

3.1　跳跃表的区域和路由段

① 首先给出区域的定义：

区域 region(m，firstkey)：第 m 层区域 region(m，firstkey)是跳跃表处于第 m 层某些连续节点的子集，令子集首节点为该子集中键值最小的节点，其键值为 firstkey；特别地该子集有且仅有首节点的最大层高大于 m，该子集最后节点的后继节点是其它区域首节点.

区域之间的交集均为空；令 AllSearchSet 为以跳跃表各个第 0 层节点为目标的全部原生查询操作合集，则全部区域的并集等于 AllSearchSet 中全部查询在第 m 层的锚点集合.

令 AllRegionSet 为跳跃表第 m 层全部区域的集合，nodes(r)为区域 r 的节点集合，则：

$$\begin{cases}\bigcap\limits_{r\in \text{AllRegionSet}} \text{nodes}(r)=\varnothing \\ \bigcup\limits_{r\in \text{AllRegionSet}} \text{nodes}(r)=\sum\limits_{q\in \text{AllSearchSet}} \text{Anchor}(q,m)\end{cases}$$

② 其次给出路由段的定义：

路由段 segment(m，RegionList)：第 m 层路由段 segment(m，RegionList)是跳跃表处于第 m 层某些连续区域的集合，RegionList 中包含了该段中各个区域首节点的 firstkey. 路由段将由 1 个或多个完整区域 region 组成(热段仅包含 1 个完整 region，冷段可能包含 1 到多个完整 region).

令 AllSegmentSet 为跳跃表第 m 层全部路由段的集合，regions(s)为路由段 s 的区域集合，则：

$$\begin{cases}\bigcap\limits_{s\in \text{AllSegmentSet}} \text{regions}(s)=\varnothing \\ \bigcup\limits_{s\in \text{AllSegmentSet}} \text{regions}(s)=\text{AllRegionSet}\end{cases}$$

令路由段 segment 的 minkey 是其区域集合中 firstkey 最小区域的首节点键值，令路由段 segment 的 maxkey 是其区域集合中 firstkey 最大区域最后节点的后继节点键值.

如图 4 所示、路由加速表 FastAccTab 是一个结构体升序数组，数组中每个结构体元素代表一个路由段，包含如下信息：{Flag，[minkey，maxkey)，FastAccPtr，targetlevel}. 其中 Flag 标示了该路由段的冷热性质，minkey 是当前路由段首节点键值(以此升序)，maxkey 是下个路由段首节点键值. 如果该路由段是热段，FastAccPtr 是对应路由加速点地址，否则为哨兵节点地址. targetlevel 是加速点所在层数 m，所有元素的该值均一致.

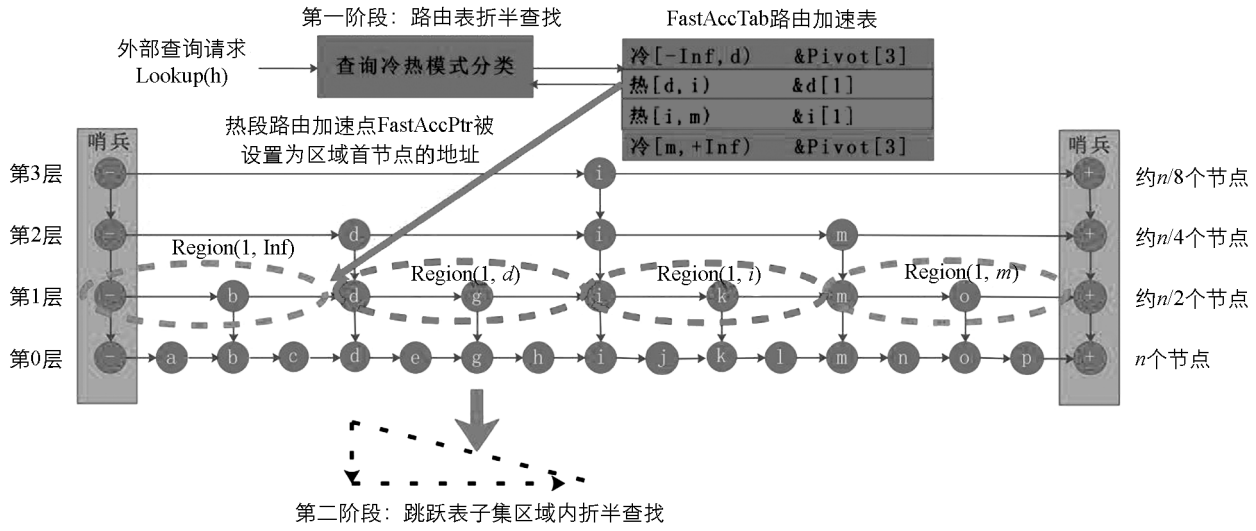

图4 跳跃表区域和加速段示意图

3.2 热点敏感的查询算法

热点敏感的查询算法的算法逻辑如下：

1）对于进入蓄水池且在蓄水池中编号为奇数的查询请求，执行 HotRegionLookup 策略1，查询从跳跃表最左哨兵节点发起并记录该查询的执行过程信息和耗时，将以上信息放入对应的缓存结构 BasePool(简称蓄水池 B)；

2）对于进入蓄水池且在蓄水池中编号为偶数的查询请求，执行 HotRegionLookup 策略2：查询从快速路由表 FastAccTab 发起(包含 FastAccTab 检索热点路由和后续查询执行)，向右向下搜索目标节点，并记录该查询的执行过程信息和耗时，将以上信息放入对应的缓存结构 AccPool(简称蓄水池 A)；

3）对于没有进入蓄水池的大部分查询请求，执行 HotRegionLookup 策略3，查询从最左哨兵节点(当系统处于“预热”模式)或者快速路由表 FastAccTab(当系统处于“加速”模式)发起，向右向下搜索目标节点，不记录查询执行过程信息和耗时，以降低对性能的影响.

特别地，当查询从快速路由表 FastAccTab 发起时，首先折半查询 FastAccTab 中路由段数组 segments，找到满足(s0.minkey$\leq q <$s0.maxkey)条件的对应路由段 s0；其次进行查询入口点设置：如果 s0 被标记为“冷”，则本次查询的入口点是哨兵节点，如果 s0 被标记为“热”，则本次查询的入口点是 s0 的对应 FastAccPtr. FastAccPtr 是该查询在第 m 层的锚点所在 region 的首节点地址，此处路由段的分配、冷热设置和对应 FastAccPtr 在路由刷新阶段会被提前写入到路由表中，具体在第4.2节的热区加速点分配算法 HotRouterAllocate 中有阐述.

综上所述，HotRegionLookup 算法主要修改搜索入口地址，其向右向下搜索目标节点的过程与原生跳跃表算法中的 Lookup 执行过程[1]保持一致，这里不再冗述. 此外，HotRegionLookup 需要对插入和删除操作做针对性调整.

3.3 对删除和插入操作的针对性处理

由于引入了路由加速机制，跳跃表的删除和插入操作需要进行针对性处理和影响分析.

1）删除操作分为逻辑标注和物理删除两个阶段. 标准的 Lookup 访问操作只会执行逻辑标注，物理删除在每个数据分析周期结束后统一进行，物理删除操作将从最左上角哨兵节点发起，无视路由加速机制.

对于 Lookup 访问操作来说，如果判断某个节点的 mask 为逻辑删除，则继续利用其指针进行跳跃，只有在查询确定当前节点就是目标节点且 mask 标记为逻辑删除时，才根据查询/插入/更新/删除的不同操作目的，依靠可见性进行处理.

如果某个被逻辑删除的节点被设置成为路由点，则需要等待其失去路由点身份后才能进行物理删除

操作，考虑到跳跃表能够维护的路由点规模有限，我们将其归结为额外的内存空间管理开销，并不视为内存泄露.

2）插入操作和原生跳跃表插入操作存在差异：对于冷区数据的操作来说，与原生算法一致；对于基于路由表进行加速访问的热区数据操作来说，插入节点的高度将不能高于加速层 m. 因为在路由加速机制下，系统无法低成本获取目标节点 p 在 m 层以上各层的前驱节点指针，本文将这种操作称为高度截断操作 insert_level_cut，当 m 被压低时，该操作会破坏跳跃表的概率结构，需要在自适应算法中引入规避机制. 因此这里插入截断操作的次数将被记录，支撑后续的自适应算法.

3.4　时间复杂度分析

如前文所述，第 0 层节点规模为 N 的跳跃表以 $p=0.5$ 概率逐层采样构成子链表. 从原生跳跃表的 Lookup 查询执行过程可知，其搜索算法本质上是概率化的折半查询，因此原生跳跃表搜索的时间复杂度为 $O(\mathrm{Log}_2 N)$.

由 4.2 节阐述的 HotRouterAllocate 分配算法，会在第 m 层设置 $topk$ 个路由加速热段，每段分配的路由加速点设置在热区首节点上. 由于冷热间隔的原因，冷段最大规模是 $1+topk$. 因此 FastAccTab 的最大规模是 $1+2\times topk$.

这里假设跳跃表逐层以 $p=0.5$ 采样概率构造子集，则第 $m+1$ 层($0<m<=$跳跃表最大层高 d)的节点规模为 $N/2^{m+1}$；由于第 m 层每个区域有且仅有 1 个最大高度超过 m 的首节点，因此第 m 层有 $N/2^{m+1}$ 个区域，以区域 Region 中节点为锚点的节点集合 S 平均规模是 $N/(N/2^{m+1})=2^{m+1}$. 由跳跃表的性质可知，S 其实是跳跃表的子集，从其在 m 层的首节点向右向下搜索，其时间复杂度是 $O(\mathrm{Log}_2 2^{m+1})=O(m+1)=O(m)$.

由此可知，假设一个 Lookup 查询以概率 p 落入热段中，其查询时间的期望则为

$$E(\mathrm{AccLookupCost}) \approx O(\log_2(1+2*topk))+p*O(m)+(1-p)*O(\mathrm{Log}_2 N)$$

采用 HotRegionLookup 算法基于原生跳跃表查询的吞吐量加速率(用 η_{ACC} 表示)为

$$\eta_{\mathrm{ACC}} \approx \frac{p*(O(\mathrm{Log}_2 N)-O(m))-O(\mathrm{Log}_2(1+2\times topk))}{O(\mathrm{Log}_2(1+2\times topk))+p\times O(m)+(1-p)\times O(\mathrm{Log}_2 N)} <= \frac{O(\mathrm{Log}_2 N)-O(m)-\mathrm{Log}_2 3}{O(m)+\mathrm{Log}_2 3}$$

由此可得出如下结论：

1）落入热段的概率 p 越大，加速率越大；

2）加速层高度 m 和加速段规模 $topk$ 越大，加速率则越小；

3）如果 $p\times(O(\mathrm{Log}_2 N)-O(m))<=O(\mathrm{Log}_2(1+2\times topk))$，即落入加速段的查询加速量不能覆盖 FastAccTab 折半搜索新增的查询成本，则无法实现加速效果；

4）在最理想的情况下($p=1$ 所有查询访问均落入热点加速区域，$topk=1$ 仅有 1 个热区)，

$$\eta_{\mathrm{ACC}} <= \frac{O(\mathrm{Log}_2 N)-O(m)-\mathrm{Log}_2 3}{O(m)+\mathrm{Log}_2 3}$$

此时能够取得非常高的吞吐量加速比.

但是，路由加速表的设置策略会影响热区命中率，因此概率 p 与 m 和 $topk$ 隐含相关. 不能一味地减少 m 和 $topk$，否则会降低预测命中率 p.

4　自适应路由设置机制

从第三节分析可知，决定跳跃表的加速性能的主要参数是加速点所在层次 m 和路由加速表热段的规模 $topk$. 对于实际场景下的不同工作负载，超参数很难有一劳永逸的设置.

首先介绍跳跃表的加速策略自适应算法，其次给出基于特定加速层的热区加速点分配算法.

4.1 加速策略自适应算法

这里将问题抽象为马尔科夫决策过程，从而基于强化学习思想进行求解. 具体地将查询访问操作的工作负载视为环境，智能体进行路由加速策略选择，利用环境奖励的反馈机制评估不同状态下超参数设置策略的影响. 周而复始，逐步实现从与环境的交互中学习到特定跳跃表在特定工作负载下的超参数设置策略.

首先，前文分析过超参数 m 的设置将影响到跳跃表插入操作，即高度截断操作 insert_level_cut，该操作过于频繁有可能将跳跃表在 m 层以上的概率结构退化为链表，从而导致潜在性能问题. 基于加速效果的奖励机制难以直接反映 insert_level_cut 影响，因此本文将 insert_level_cut 操作的最近次数与奖励重塑机制[7]结合以提升强化学习效率.

其次，跳跃表作为大数据系统的关键数据结构，性能稳定性非常重要. 在强化学习初期往往需要加强探索，导致策略的效果波动较大，并不适合直接运用在大规模负载查询中，此时没有进入蓄水池 A/B 的绝大多数查询 q 执行原生算法，只有进入蓄水池 A 的查询利用路由加速表 FastAccTab，其加速性能与进入蓄水池 B 的原生查询进行对比，用来训练强化学习系统，称为“预热阶段”，当系统的稳定性和覆盖程度达到预设条件后切换到“加速阶段”.

进入“加速阶段”后，没有进入蓄水池 A/B 的绝大多数查询 q 利用路由加速表 FastAccTab 但并不进行计时和采样，通过蓄水池采样进入蓄水池 A/B 的查询继续训练强化学习系统；如果随着时间流逝，加速效果持续低于原生算法达到阈值次数，说明跳跃表所依赖的可预测局部性已经失效，需关闭加速机制，重新切换回“预热阶段”，降低系统性能风险.

为了达到以上设计要求，本文基于 SARSA 算法[8]思想进行算法设计. 将状态空间离散化，再通过监控状态—动作价值表 Q 的变化情况和加速效果，决定从“预热阶段”到“加速阶段”的切换时机.

给出相关定义如下：

1) 定义 t 时段直接奖励 reward：

$$\text{reward}(t)=\frac{\sum\limits_{b\in \text{BasePool}(t)}\text{duration}(b)-\sum\limits_{a\in \text{AccPool}(t)}\text{duration}(a)}{\sum\limits_{b\in \text{BasePool}(b)}\text{duration}(b)}-\tanh\left(\frac{\text{Load}(t)-\text{Load}(t-1)}{\text{Load}(t-1)}\right) \tag{1}$$

这里 Load(t)和 Load($t-1$)分别是 t 和 $t-1$ 时段系统 CPU 负载指标，在奖励设计中综合考虑了加速效率和系统负载变化情况. duration(x)是操作 x 的执行耗时.

2) 定义 t 时段的势函数：

$$F(S_t)=-\tanh(S_t->\text{insert\_cut\_sum}) \tag{2}$$

其中 insert_cut_sum 是当前时刻 t 所在状态 S_t 最近 3 个时间周期内对跳跃表 insert 操作的高度截断(insert_level_cut 操作)次数之和，反映了加速机制对跳跃表结构的负面影响程度，应越低越好.

3) 定义 ε-greedy 动作选择策略：

令

$$A^*=\arg\max_a Q(s,a)$$

$$p(a^*\mid s)=\begin{cases}1-\varepsilon+\varepsilon/\mid \text{Action}(s)\mid & a^*=A^*\\ \varepsilon/\mid \text{Action}(s)\mid & a^*!=A^*\end{cases} \tag{3}$$

4) 定义每时间段的状态—动作价值表更新算法，这里加入了奖励重塑机制：

令 S_{t-1} 为 $t-1$ 时刻状态，A_{t-1} 为 $t-1$ 时刻动作；令 S_t 为 t 时刻状态，通过公式(3)选择出动作 a^* 作为 A_t：

$$Q[S_{t-1}, A_{t-1}] = Q[S_{t-1}, A_{t-1}] + \lambda * (\varphi * F(S_t) - F(S_{t-1}) + \text{reward}(t) + \varphi * Q[S_t, A_t] - Q[S_{t-1}, A_{t-1}]) \quad (4)$$

5）定义状态空间 S，共 $\|S1\| * \|S2\| * \|S3\| * \|S4\| * \|S5\| * \|S6\|$ 个离散状态.

表 1　状态空间定义表

状态属性	含　义	离散化映射示例
$S1$	令 JS$[m, T, T-1]$为 T 与 $T-1$ 时刻第 m 层工作负载采样分布之间的 Jensen-Shannon 距离(以下简称 JS 距离，用 S_{JS} 表示)[9]；$S_{JS}[m, T-1, T-2]$为 $T-1$ 与 $T-2$ 时刻第 m 层工作负载采样分布之间的 S_{JS} 距离；$S1=(S_{JS}[m, T, T-1]+S_{JS}[m, T-1, T-2])/2$	L1：[0，1/4) L2：[1/4，1/2) L3：[1/2，3/4) L4：[3/4，1]
$S2$	令 $S_{JS}[m-1, T, T-1]$为 T 与 $T-1$ 时刻第 $m-1$ 层工作负载采样分布之间的 JS 距离；$S_{JS}[m-1, T-1, T-2]$为 $T-1$ 与 $T-2$ 时刻第 $m-1$ 层工作负载采样分布之间的 JS 距离；$S2=(S_{JS}[m-1, T, T-1]+S_{JS}[m-1, T-1, T-2])/2$	L1：[0，1/4) L2：[1/4，1/2) L3：[1/2，3/4) L4：[3/4，1]
$S3$	令 NS 是 T 时段工作负载查询落在跳跃表加速层 m 上的锚点集合；$p(x)$是锚点 x 在 NS 中出现的频率 $S3=\frac{-\sum_{x\in NS} p(x)\times\log_2(p(x))}{\log_2\lvert\text{unique}(NS)\rvert}$	L1：[0，1/2) L2：[1/2，3/4) L3：[3/4，+∞)
$S4$	T 时段跳跃表加速层 m 的节点规模	L1：[0，5K) L2：[5K，50K) L3：[50K，+∞)
$S5$	令 C_{cpuinfo} 为 T 时刻 CPU Load Average 指标，令 C_{Cores} 为服务器 CPU 核心总数；$S5=C_{\text{cpuinfo}}/C_{\text{Cores}}$	L1：[0，1/2) L2：[1/2，3/4) L3：[3/4，+∞)
$S6$	当前状态属性组合$[S1, S2, S3, S4, S5]$维持不变的时段长度	L1：[0，2) L2：[2，5) L3：[5，+∞)

6）定义动作空间，共 6 个离散动作：

表 2　动作空间定义表

动作	层数控制策略	*topk* 控制策略
Action1	加速点所在层数下移 1 层	精简 Tidy
Action2	加速点所在层数下移 1 层	普通 Common
Action3	加速点所在层数保持不变	精简 Tidy
Action4	加速点所在层数保持不变	普通 Common
Action5	加速点所在层数上移 1 层	精简 Tidy
Action6	加速点所在层数上移 1 层	普通 Common

7）定义状态切换控制逻辑：

令 Qt 是 Q 表在 t 时刻的副本，$Qt[s, *]$表示 t 时刻状态 s 对应的状态—动作值向量 a_vs. 令 max_rankpos 函数返回该向量中最大值的位置 maxpos；

定义辅助矩阵 QMx，令 $QMx[t, s]=\text{max\_rankpos}(Qt[s, *])$；即 QMx 的元素 $QMx[t, s]$表示 t 时刻状态 s 下价值最高的动作编号. $QMx[t, *]$表示 t 时刻全部状态分别对应的最大价值动作的编号向量；

定义辅助矩阵 USx，令 $USx[s, a]$ 为 Q 表中对应元素的更新次数，$\min(USx[*, *])$ 表示 Q 表中全部元素被更新的最少次数；

定义辅助列表 RewardHistory，令 RewardHistory[t] 取值为强化学习系统在 t 时刻的即时奖励值；定义列表 LvlHistory，令 LvlHistory[t] 取值为 t 时刻的跳跃表加速层高度 m.

如果满足公式(5)中的阈值 $C1/C2/C3$ 和条件，即：t 时刻 Q 表中全部元素都被充分更新覆盖，最近 Q 表各个状态对应的最大价值动作没有剧烈变化，系统通过蓄水池采样测量的加速效果稳定保持一定周期，系统将从“预热”切换到“加速”阶段：

令

$$F(x)=\begin{cases}1, x==\text{true}\\ 0, x==\text{false}\end{cases}\begin{cases}\sum_{i=0}^{C1}F(QMx[t-i, *]==QMx[t-i-1, *])>=C1*C2\\ \&\quad \min(USx[*, *])>=C3\\ \&\quad \sum_{i=0}^{C1}F(\text{RewardHistory}[t-i]>=0)==C1\end{cases}\tag{5}$$

如果满足公式(6)中的阈值 $C1$ 和条件，即 t 时刻加速层在较高区间徘徊且系统通过蓄水池采样测量的加速效果不佳延续一定周期，系统从“加速”切换回到“预热”阶段.

令

$$F(x)=\begin{cases}1, x==\text{true}\\ 0, x==\text{false}\end{cases}\begin{cases}\sum_{i=0}^{C1}F(\text{LvlHistory}[t-i]>=\frac{2}{3}*(\text{skiplist}\rightarrow\text{maxLevel}))==C1\\ \&\quad \sum_{i=0}^{C1}F(\text{RewardHistory}[t-i]<0)==C1\end{cases}\tag{6}$$

综上所述，跳跃表自适应加速算法的逻辑流程如下(Q 表在跳跃表建立时候被随机初始化)：

1) 获取当前时间段 t 内在蓄水池 A/蓄水池 B 中的路径采样信息，根据表 1 进行状态映射得到 S_t；

2) 获取当前时间段 t 内在蓄水池 A/蓄水池 B 中的耗时统计信息以及系统负载信息，根据公式(1)计算出时间段 t 的即时奖励 reward(t)；

3) 获取当前状态 S_t 和之前状态 S_{t-1} 的 insert_cut_sum 统计值，利用公式(2)计算奖励重塑的修正值；

4) 根据公式(3)中的 ε-greedy 动作选择策略，选择当前状态 S_t 下的动作 A_t；

5) 根据公式(4)调整 $t-1$ 时段下状态 S_{t-1} 和行为 A_{t-1} 在状态－动作价值表 $Q[S_{t-1}, A_{t-1}]$ 的取值；

6) 执行当前选定的离散动作 A_t：调整跳跃表的加速层 m 和热区规模 $topk$ 选择策略；

7) 执行热区加速点分配算法 HotRouterAllocate(在 4.2 节进一步阐述)，在跳跃表第 m 层上选择 $topk$ 个热区，刷新路由表 FastAccTab；

8) 进行系统工作状态切换判断：

① 若当前系统状态为“预热”且满足公式(5)中切换条件，将系统状态切换为“加速”；

② 若当前系统状态为“加速”且满足公式(6)中切换条件，将系统状态切换为“预热”，此时辅助统计量 RewardHistory/LvlHistory/USx 均重置，但 Q 表不会被重置.

4.2 热区加速点分配算法 HotRouterAllocate

从本质上来说，热区加速点设置是一个组合优化问题，目的在于基于工作负载的预测情况找到合适加速点取得合理加速效果. 以上优化问题的搜索空间不能过大，如果加速点的可选范围是跳跃表全部节点，加速点规模设为 $topk$，令跳跃表节点规模为 S，则以上优化问题的解空间规模是 C_S^{topk}，当跳跃表的规模在百万级以上时，求解空间将过于庞大以至于无法在可以接受的时间范围内完成搜索. 本文对问题进行简化：将加速点的可选范围从跳跃表全体节点缩小到某一层 m(m 取值由 4.1 节强化学习所选择动作决定)，以减轻运算压力.

下面对热区加速点分配算法中的要素进行定义.

1) 定义区域捷径值 shortcut(m, firstkey)：从跳跃表哨兵节点最高层到第 m 层区域 region(m,

firstkey)首节点的跳跃次数. 如果加速点设置在某个 m 层区域的首节点 firstkey，则所有经过该节点的访问操作都节约 shortcut(m，firstkey)次跳跃. 如图 5 所示.

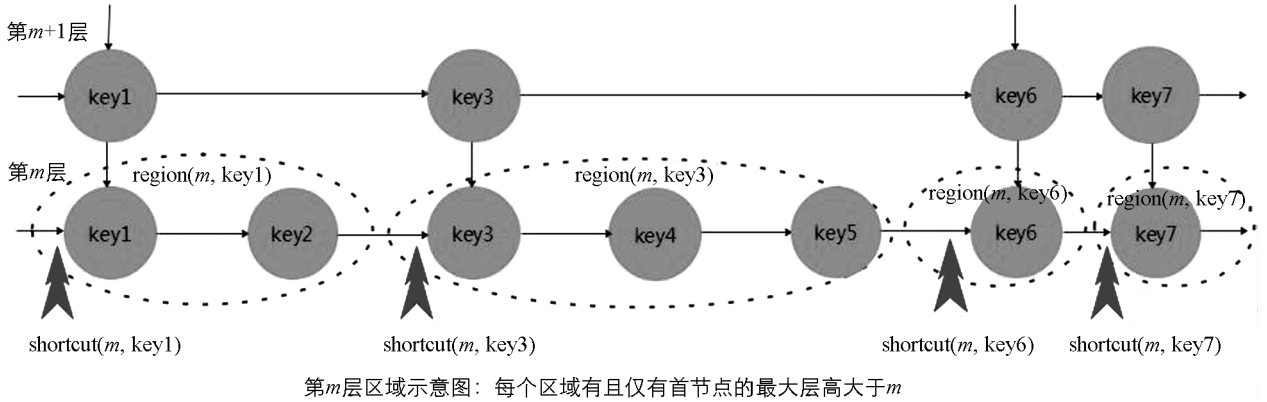

图 5　区域 region 的捷径值 shortcut 示意图

定义哈希表 ShortcutHash 为区域捷径的存储结构，通过区域首节点 Firstkey 访问.

2) 定义节点工作负载 Workload(m，t，key)：在 t 时间段内，经过跳跃表第 m 层键值为 key 的节点下行访问计数值. 从以上定义可知：节点工作负载 Workload 是该时间段内访问操作落在其与后继节点间的频数，也就是该节点作为第 m 层锚点的查询路径出现频数.

这里的节点工作负载需要通过现有负载对未来进行预测，例如 BP 神经网络[10]、LSTM[11]、SVR[12]；考虑到跳跃表节点规模可能非常大，百万规模跳跃表中间层包含的节点数目就高达数万，为每个节点维护独立的时间序列预测模型将会带来很大运算压力，本文给加速层 m 中每个节点采用二阶指数平滑算法[13]做负载预测，以追求预测精度与时效性之间平衡.

定义哈希表 HitHash(形如 HitHash<key，Workload>)为节点工作负载的存储结构，通过 key 访问.

3) 定义区域价值 RegionValue(m，t，firstkey)：当加速点设置在 firstkey 上，在 t 时间段内，跳跃表第 m 层以 firstkey 为首节点键值区域 r 减少的跳跃次数.

令 PtDis(m，key1，key2)为跳跃表第 m 层键值分别为 key1、key2 节点之间跳跃间隔.

$$\text{RegionValue}(m, t, \text{firstkey}) = \sum_{p \in \text{region}(m,\ \text{firstkey})} \text{workload}(m, t, p) * (\text{shortcut}(m, \text{firstkey}) - \text{PtDis}(m, \text{firstkey}, p)) \tag{7}$$

定义哈希表 RegionValHash(形如〈Firstkey，RegionValue〉)为区域价值的存储结构，通过区域首节点 Firstkey 访问.

下面对热区加速点分配算法 HotRouterAllocate 进行阐述，算法基于如下两点考虑：

一是任何被选中的热点区域，有且只有一个加速点被设置用来对本区域的节点访问进行加速，简而言之，热点区域将不会被分割. 以上策略保证热区加速点的宝贵资源能够充分利用，让尽量多的热点区域能够被选中进行加速，从实际效果来看，本文工作将该加速点设为区域首节点，具有更强的适应性.

二是跳跃表特定层的热点区域规模不应该过大，针对特定跳跃表的特定工作负载，在特定层热点区域规模也难以由人为预先指定，因此热点区域规模需要尽可能基于工作负载实际分布得出，此处逻辑由 4.1 节强化学习所选择动作的“Tidy”或者“Common”策略决定.

综上所述，HotRouterAllocate 算法逻辑流程如下：

1) 对跳跃表 m 层中每个区域，计算每个区域 shortcut，依次放入 ShortcutHash；

2) 对跳跃表 m 层中每个区域的各个节点 p，利用之前 AccPool/BasePool 蓄水池中采样数据，采用指数平滑算法预测该节点后续工作负载 workload(m，$t+1$，p)，并依次放入 HitHash；

3) 对跳跃表 m 层中每个区域，根据公式(7)结合 HitHash 和 ShortcutHash 计算每个区域的价值 RegionValue，并依次放入 RegionValHash；遍历哈希表 RegionValHash，将其中各个元素的区域价值插入数组 RValVec；令 $mvv = \text{median}(RValVec)$，$rvv = \text{std}(RValVec)$；

4) 确定热区规模 $topk$：利用聚类算法来提取一维工作负载数据分布的内在规律，考虑到效率因素，本文采用划分聚类 k-means[14] 来确定较大价值的热点区域规模，而非更高精度但耗时的聚类算法如 PAM 算法或者 dbscan 密度聚类.

① 若 $topk$ 设置策略为“Common”：令 Sub$Ds0$=Region$ValVec$ 中取值超过中位数 mvv 的全部元素集合，设置 $topk$=|Sub$Ds0$|；

② 若 $topk$ 设置策略为“Tidy”：首先抽取 Region$ValVec$ 中取值大于($mvv+rvv$)的元素到集合 SubDs1、其次对 Region$ValVec$ 中取值在[$mvv-rvv$，$mvv+rvv$]范围的元素进行二分 $k-$means 划分聚类，令中心值较高的聚簇为 Sub$Ds2$，设置 $topk$=|Sub$Ds1$|+|Sub$Ds2$|.

5) 刷新路由加速表 FastAccTab：选择哈希表 RegionValHash 中取值最大的 $topk$ 个元素所在 region 构成热段，每个热段的入口点 FastAccPtr 设置为对应 region 所在 firstkey 的地址；未被选中的区域被热段分割成为冷段，保持冷热段的 minkey 呈现升序排列插入 FastAccTab->segments，完成 FastAccTab 刷新.

5 实验结果

实验环境采用 1 台 DELL R730 服务器，256G 内存，2 颗 Intel Xeon E5-2680 v4 规格 CPU，每颗处理器有 14 个核心. 实验环境 CPU 总体负载指标保持在 10 左右，模拟系统遭遇中等程度服务压力. 其中强化学习的学习率 λ 设置为 0.1，折扣率 φ 设置为 0.5，探索率 ε 设置为 0.1，$C1$=10，$C2$=0.8，$C3$=6. 为了验证本文所提出的加速算法和自适应机制，给出如下实验结论.

5.1 跳跃表查询加速有效性实验

目的在于验证访问压力倾斜和波动幅度较低的场景下，本文方法的加速有效性. 本实验有 4 个规模不同的 24 层跳跃表，分别有 100,200,500,1 000 万规模，节点键值类型是 8 字节长整形.

1) 局部性弱的场景(50%随机、50%压力集中在 20%区域形成 512 个正态分布，各个热点区域中心每个周期随机左右移动 1/8 个热区范围标准差)测试：

此时访问负载较为分散，为了尽量捕捉负载热点就需要提升 $topk$. 这种情况下 m 无论过高(m=12，shortcut 利用率不高)或者过低(m=4，过多 $topk$ 增加了路由搜索消耗，存在加速失败的情况)都不能有效工作. 自适应算法在不同规模跳跃表停留在不同层次上(m=7/8/9/10)，能够达到相对稳定的加速效果.

2) 局部性强的场景(20%随机、80%压力集中在 20%区域形成 64 个正态分布，各个热点区域中心每个周期随机左右移动 1/8 个热区范围标准差，模拟访问压力重度倾斜)测试：

此时访问负载较为集中($topk$ 无需过大)且可预测性较好，总的来说局部性较强，采用路由加速算法都可以达到性能提升的效果，此时自适应算法尽量驻留在了较低层次(m=4)以提升 shortcut 利用率，能够达到稳定和高效的加速效果.

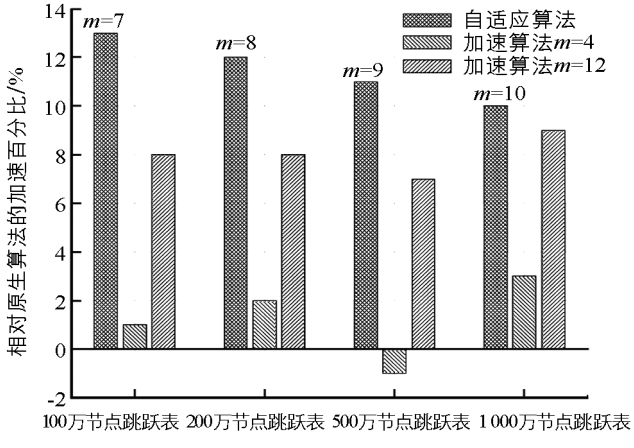

图 6 局部性弱的场景测试结果

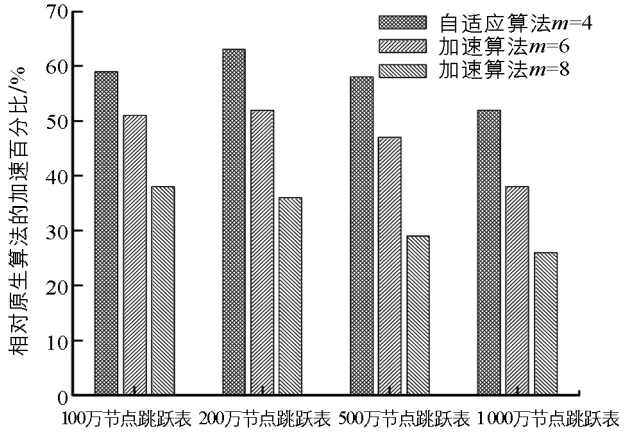

图 7 局部性强的场景测试结果

5.2　跳跃表查询加速策略自适应实验

目的在于验证加速策略针对访问热点波动的自适应能力，自适应机制每隔 1 个时间段(15 s)启动一次调整加速策略. 图 8 和图 9 是工作负载漂移场景下加速策略性能：

1) 工作负载中速漂移的场景测试结论. 24 层 200 万节点规模跳跃表，节点键值类型是 8 字节长整形，访问压力重度倾斜，64 个热点区域中心每个周期随机左右平移：1～10 时段 1/4 个热区范围标准差；11～20 时段 1/2 个热区范围标准差；21～30 时段 1/4 个热区范围标准差.

该结果说明：当工作负载变化存在可预测性时，自适应算法的灵活性优于原生算法和加速层固定的算法. 在 1～10 阶段负载集中且波动率较低，此时 $m=4$ 停留在较低层提升 shortcut 利用率；在 11～20 阶段负载波动率增强，m 呈现上涨趋势以提升预测命中率；在 21～30 阶段负载波动又下降，m 也随之下降以提升 shortcut 利用率，说明在这里利用强化学习可以从环境中充分学习，找到适合工作负载的超参数设置策略.

2) 工作负载高速漂移的场景测试结论. 24 层 200 万节点规模跳跃表，节点键值类型是 8 字节长整形. 访问压力重度倾斜，64 个热点区域中心每个周期随机左右平移：1～10 时段 1/4 个热区范围标准差；11～20 时段 1/2 个热区范围标准差；21～30 时段 1 个热区范围标准差.

该结果说明：当工作负载的波动逐渐增强时，对热点的可预测性逐渐降低，此时基于局部性原理的算法均存在性能下降的问题，但自适应跳跃表能够充分感知环境变化(m 值在持续上涨以尽量捕捉热点)，及时切换加速机制(根据公式(6)，200 万规模的 24 层跳跃表在 $m>=16$ 且加速失败情况持续一段时间后，将达到切换条件)以稳定整体性能、节约运算资源.

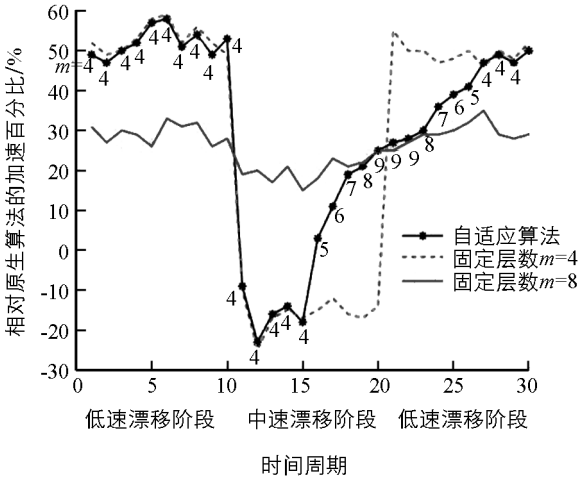

图 8　工作负载中速漂移场景测试结果

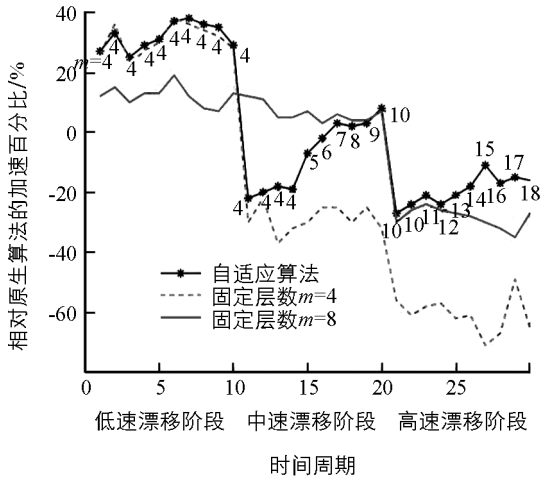

图 9　工作负载高速漂移场景测试结果

6　结束语

本文针对跳跃表性能优化的自适应机制进行了初步的探索，具体研究了利用局部性原理提升跳跃表查询效率的问题. 首先利用跳跃表本身的结构特点，结合工作负载预测技术设计了基于热点的路由加速机制；其次根据加速的实际效果和管理资源的消耗情况，设计了基于强化学习的自适应机制来动态调整加速点的所在层次. 后续工作目标是去掉加速点保持在同一层的约束，不同热点区域的路由加速点由对应热点区域的变化幅度和变化频率自主决定. 一是需要研究更有效的针对性搜索算法以减少搜索的计算压力，目前基于同一层的加速点选择算法可以作为进一步工作的基础；二是既然加速机制由集中控制变为分散控制，自管理模式也需要随之变化，需要考虑多智能体强化学习与以上加速机制的结合方式.

参考文献：

[1] PUGH W. Skip Lists: A Probabilistic Alternative to Balanced Trees [J]. Communications of the ACM, 1990, 33(6): 668-676.

[2] MUNRO J, PAPADAKIS T, SEDGEWICK R. Deterministic Skip Lists [C] //In Proceedings of the 3rd Annual ACM-

SIAM Symposium on Discrete Algorithms, Florida, 1992: 367-375.
[3] ZHANG J T, WU S, TAN Z Y, et al. S3: A Scalable in-Memory skip-list Index for key-Value Store [C]. Proceedings of the VLDB Endouiment, 2019, 12(12): 2183-2194.
[4] KIM C, CHHUGANI J, SATISH N, et al. FAST: Fast Architecture Sensitive tree Search on Modern Cpus and Gpus [C] //SIGMOD, 2010: 339-350.
[5] 潘 恬, 黄 涛, 张雪贝. 基于局部性原理跳表的内容路由器缓存快速查找机制 [J]. 计算机学报, 2018, 41(9): 2029-2043.
[6] VITTER J S. Random Sampling with a Reservoir [J]. ACM Transactions on Mathematical Software, 1985, 11(1): 37-57.
[7] NG A Y, HARADA D, RUSSELL S J. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping [C] //In: Proc. Of the 16th Int'l Conf. On Machine Learning. New York: Morgan Kaufmann Publishers, 1999: 278-287.
[8] SUTTON R S, BARTO A G. 强化学习 [M]. 2版. 俞 凯, 译. 北京: 电子工业出版社, 2019: 127-129.
[9] MAJTEY A P, LAMBERTIP W, PRATO D P. Jensen-Shannon Divergence as a Measure of Distinguishability Between Mixed Quantum States [J]. Physical Review, 2005, 72(5): 762-776.
[10] REN C X, WANG C B, YIN C C, et al. The Prediction of Short-Term Traffic Flow Based on the Niche Genetic Algorithm and BP Neural Network [M]. Berlin: Springer, 2012: 775-781.
[11] HOCHREITER S, SCHMIDHUBER J. Long Short-Term Memory. Neural Computation [J], 1997, 9(8): 1735-1780.
[12] CHANG C, LIN C. LIBSVM: A Library for Support Vector Machines [J]. ACM Transactions on Intelligent Systems and Technology, 2011, 2(3): 27-65.
[13] HOLT C. Forceasting Seasonals and Trends by Exponentially Weighted Moving Averages [J]. International Journal of Forecasting, 2004, 20(1): 5-10.
[14] HAN J W, KAMBER M. 数据挖掘: 概念与技术 [M]. 3版. 范 明, 孟小峰, 译. 北京: 机械工业出版社, 2012.

A Hot-Spot-Sensitive Adaptive Skiplist

WEN Tao, JI Feng, LIU Li-xia

ZTE Co. Ltd. State Key Laboratory of Mobile Network and Mobile Multimedia Technology, Nanjing 210012, China

Abstract: In this study, the locality principle of program access is used to improve the skiplist query efficiency, and the adaptive mechanism of the above acceleration strategy is investigated. Firstly, the skiplist query operation is improved, so that it can additionally return a subset of the access paths at a specific level. Next, the skiplist query operation is sampled by using reservoir sampling. Then, the workload of the specific level of the skiplist is predicted based on the sampling results, hot-spot areas are selected according to workload, and acceleration points are setin the hot-spot areas to improve query efficiency. Finally, according to the improvement of query efficiency and the changes insystem load, SARSA algorithm and reward shaping mechanism are usedto automatically adjust the acceleration point level and scale so as to achieve atrade-off between query acceleration and management costs. Experiments show that in the application scenario of accessing skewed skiplist, the proposed method has lower latency than the original skiplist query algorithm; and in the application scenario where the access mode changes gradually, the proposed method can flexibly adjust the acceleration strategy as the environment changes.

Key words: hot-spot-sensitive; skiplist; reinforcement learning; SARSA; reward shaping

责任编辑 汤振金