计 算 机 研 究 与 发 展
Journal of Computer Research and Development

https://crad.ict.ac.cn
63(5): 1139−1157, 2026

自适应优先搜索树的函数式建模、验证及访问优化

左正康[1,2,3] 周 超[1] 柯雨含[1] 游 珍[1] 王昌晶[1,2,3]

[1]（江西师范大学人工智能学院 南昌 330022）

[2]（高性能计算江西省重点实验室（江西师范大学） 南昌 330022）

[3]（语言智能研究中心（江西师范大学） 南昌 330022）

（zuo803@jxnu.edu.cn）

Functional Modeling, Verification and Access Optimization of Adaptive Priority Search Tree

Zuo Zhengkang[1,2,3], Zhou Chao[1], Ke Yuhan[1], You Zhen[1], and Wang Changjing[1,2,3]

[1]（*School of Artificial Intelligence*, *Jiangxi Normal University*, *Nanchang* 330022）

[2]（*Jiangxi Provincial Key Laboratory for High Performance Computing* (*Jiangxi Normal University*), *Nanchang* 330022）

[3]（*Language Intelligence Research Center* (*Jiangxi Normal University*), *Nanchang* 330022）

Abstract The priority search tree (PST) is a data structure that combines the characteristics of search trees and priority queues, enabling efficient retrieval of high-priority elements while maintaining worst-case search efficiency comparable to that of search trees. Nipkow et al. further modeled and mechanized the verification of PST with a static priority mechanism, laying an important foundation for its formal analysis and subsequent theoretical exploration. However, static priority mechanisms can lead to degraded performance under highly skewed access distributions, limiting their applicability in practice. To overcome this limitation, a functional adaptive priority search tree (FAPST) is proposed. FAPST adopts dynamic priority adjustment based on access frequency and is formally modeled and verified at the formal level to ensure functional correctness and reliability. This structure preserves balance and worst-case efficiency while adaptively responding to access patterns, thereby significantly improving the retrieval of frequently accessed elements. To further address the shifting nature of data hotspots in practical scenarios, a hotspot migration mechanism is introduced. This mechanism incorporates a priority decay strategy to accelerate responses to emerging hotspots and mitigate the stagnation of outdated ones. Experimental results demonstrate that, in both access-sensitive and hotspot-shifting workloads, the formally verified FAPST achieves notable reductions in the number of key function calls of 64.8% and 46.3% respectively, compared with conventional PSTs.

Key words functional adaptive priority search tree; dynamic priority; functional modeling; mechanical verification; hotspot migration

摘 要 优先搜索树（priority search tree, PST）是一种融合搜索树与优先队列特性的数据结构，能够在保持搜索树最坏查找效率的基础上，根据数据的优先级判断其重要性，从而实现高优先级数据的快速检索。

收稿日期：2025-12-15；**修回日期**：2026-03-05

基金项目：国家自然科学基金项目 (62462036,62462037)；江西省自然科学基金重点项目 (20242BAB26017)；江西省主要学科学术与技术带头人培养项目 (20232BCJ22013)

This work was supported by the National Natural Science Foundation of China (62462036, 62462037), the Key Project of Jiangxi Provincial Natural Science Foundation (20242BAB26017) and Cultivation Project for Academic and Technical Leader in Major Disciplines in Jiangxi Province of China (20232BCJ22013).

通信作者：王昌晶（wcj@jxnu.edu.cn）

Nipkow 等人对采用静态优先级机制的 PST 进行了形式化建模与机械化验证，为其理论分析奠定了重要基础。然而，在访问分布高度偏斜的场景下，静态机制易导致检索效率下降。为此，提出一种基于动态优先级思想的函数式自适应优先搜索树（functional adaptive priority search tree, FAPST），与 PST 相比，FAPST 在通过函数式建模与机械化验证确保正确性的同时，能够根据访问分布特征动态调整数据优先级，在维持树结构平衡性与可控最坏查找效率的前提下，有效优化访问敏感数据的查找性能。进一步，针对实际应用中热点数据随时间变化的特点，引入热点转移机制，通过优先级衰减策略提升新热点响应速度，缓解旧热点固化问题。实验评估表明，在访问敏感场景和热点转移场景下，经过形式化验证的 FAPST 通过动态优先级和热点转移机制有效优化了查询效率，较 PST 关键函数调用次数分别缩短 64.8%、46.3%。

关键词 函数式自适应优先搜索树；动态优先级；函数式建模；机械化验证；热点转移

中图法分类号 TP311

DOI: 10.7544/issn1000-1239.202550901 **CSTR:** 32373.14.issn1000-1239.202550901

在应用系统数据规模快速扩张的当下，数据密集型应用对高效检索提出了迫切需求[1]。基于层次化索引的搜索树为应对大规模数据检索问题提供了基础性的解决方案。然而，传统搜索树的设计范式主要依赖于键值的有序排列，在面对数据访问频率存在显著差异的场景时，往往难以实现有针对性的快速检索。而优先搜索树（priority search tree, PST）是一种结合了搜索树和优先队列的新型数据结构，特点是能够根据数据的优先级判断其重要性，从而实现高优先级数据的快速查找。Nipkow 等人[2] 基于静态优先级进行函数式建模与验证的 PST 是一种二叉树数据结构，其在保证接近搜索树最坏查找效率的同时，根据数据优先级调整数据存储位置，使高优先级数据更靠近根节点，从而实现高优先级数据的快速访问。然而，在许多高频访问模式动态变化场景中（如新闻资讯热点刷新、电商平台促销商品更迭），数据的优先级往往是变化的，传统的静态 PST 难以满足实时更新和查询的需求。当数据优先级不可知或低优先级数据被频繁访问时，PST 的查找效率将显著下降，限制了其在动态环境中的应用[3]。

为了解决静态 PST 的局限性，本文提出了一种基于动态优先级思想的函数式自适应优先搜索树（functional adaptive priority search tree, FAPST）。FAPST 在结构设计和操作实现中采用了函数式建模，其核心操作通过 Isabelle/HOL[4-5] 基于结构和元素不变性进行形式化验证。该方法有效避免了计算过程中的副作用，确保了结构在各种应用场景下的正确性和可靠性[6]，为 FAPST 的广泛应用提供了扎实的理论支撑。在确保结构和操作正确性的基础上，FAPST 融合了搜索树和优先级队列的特性[7-9]，引入了一种动态可调优先级机制，能够根据访问频率实时更新节点优先级，这种机制通过优化数据布局，提高了对频繁访问数据的检索效率，可适配访问敏感场景。

然而，FAPST 在热点频繁转移场景中仍存短板：历史热点优先级未及时衰减，会长期霸占高优先级路径，阻塞新热点的检索通道。新热点优先级提升后，也难以快速抢占高效路径，导致检索延迟，制约了对动态访问模式的适应性。为此，本文在 FAPST 基础上进一步引入热点转移机制。该机制通过历史热点优先级逐步衰减、新热点快速抢占高优先级路径的策略，实现热点数据平滑转移，缓解路径阻塞。新增机制与原有动态优先级机制协同工作，有效增强了 FAPST 在访问模式动态变化场景（如突发访问高峰、周期性热点转移）中的表现，使其更适用于网页搜索、智能推荐系统、大数据处理等数据密集型应用[10]。

最终实验结果表明，在访问敏感和热点转移场景下，FAPST 通过动态优先级调整与热点转移机制有效优化了算法查询效率，表现出比 PST 更优的性能和适应性。本文的组织结构如图 1 所示。根据上述内容，本文的主要贡献总结为 3 个方面：

1）基于动态优先级思想提出 FAPST 结构，并对其进行函数式建模与机械化验证。在确保结构与操作正确性的同时，可解决 PST 优先级固定的局限，优化不同动态场景下的数据访问效率。

2）在访问敏感场景下，针对 PST 难以适配访问频率动态变化、导致高频数据检索效率受限的问题，FAPST 通过其动态优先级调整机制有效提升了算法查询效率，表现出比 PST 更优的查询效率与动态适应性。

3）在热点转移场景下，针对历史热点优先级固化问题，将热点转移机制融入 FAPST，在原有结构上

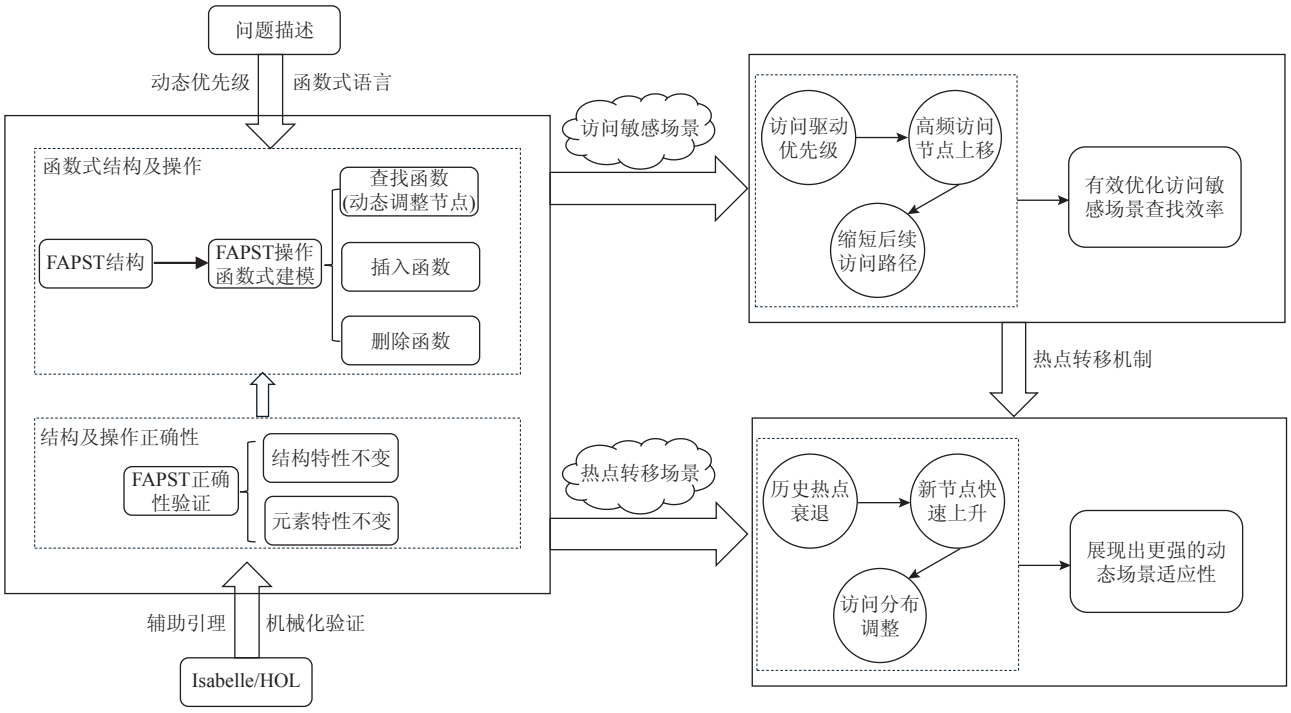


Fig. 1 Organizational chart

图 1 组织结构图

进行扩展并完成热点转移逻辑的建模与验证，有效缓解历史热点滞留导致的新热点上升延迟问题，进一步增强 FAPST 在动态访问场景中的适应性。

1 相关工作

本文主要聚焦 FAPST 的函数式建模与验证，并与相关搜索树结构进行不同场景的实验对比，本节主要介绍与 FAPST 相关的研究，包括：搜索树的结构及其变体、搜索树结构的形式化验证。

1）搜索树的结构及其变体。现有关于搜索树的研究多基于命令式编程语言。Bronson 等人[11]提出并发树，通过协调函数动态调整树状态以保障多线程环境下的操作效率。文献 [12] 设计无锁二叉搜索树算法，采用标记边的方式实现对树更新的细粒度控制。Afek 等人[13]提出的 CB 树 (counting-based tree) 依据节点访问次数设定权重，支持基于访问频率的自适应优化操作。Ellen 等人[14]针对异步共享内存系统，设计并实现了一种基于比较与交换操作的非阻塞二叉搜索树，该方案可支持线性化的二叉搜索树，且无需大字长硬件支持。Wang 等人[15]提出基于优先队列和自适应行为的动态博弈论模型，用于解决动态系统的排队与资源分配问题。文献 [16] 扩展经典二叉搜索树至任意树拓扑，引入 Steiner 闭合树概念并通过两级分解策略优化操作，实现 $O(\log\log n)$ 竞争比。文献 [17] 提出最大间隔区间搜索树，通过最小化间隔判别目标函数学习，并设计对数线性时间最优解计算的动态编程算法。Sleator 和 Tarjan[18] 提出的 Splay 树通过自调整操作实现 $O(\log n)$ 摊销性能，但最坏查询复杂度为 $O(n)$。文献 [3] 提出的动态访问搜索树，可根据访问频率动态调整节点优先级，在保持 $O(\log n)$ 最坏查询复杂度的同时降低偏斜访问下的平均查询成本。

然而，上述结构均采用命令式设计，需显式管理状态与副作用，不仅增加代码复杂性和出错概率，还提升了操作形式化验证的难度，给实际应用带来可靠性与可信度风险[19]。

2）搜索树结构的形式化验证。现有搜索树结构的形式化验证主要聚焦于不同类型搜索树上。文献 [20] 提出优化空间利用率的 Trie+结构，在 Isabelle 中完成函数式建模与验证，并给出支持中英文混合多模式匹配的通用验证规约。文献 [21] 在函数式编程语境下，完成根平衡搜索树摊销复杂度的形式化验证，提出避免显式对数操作的实现方案并开展竞争性分析。文献 [22] 给出 Splay 树的可执行结构，验证其基本属性与摊销性能。Hinze 研究 2-3 指针树[23]的结构与形式属性，证实其支持常数时间摊销访问列表两端。文献 [24] 确定原始权重平衡搜索树插入/删

除操作的旋转有效参数范围,并通过反例构造算法建立完整性。文献 [25] 以递归函数定义平衡二叉树布朗树的结构,在 Isabelle 中验证树操作正确性及必要不变性。文献 [26] 提出固定访问频率下最优二叉搜索树的递归算法并完成形式化验证。文献 [27] 对随机二叉树进行严格复杂度分析,建模概率分布、期望路径长度等属性并建立数学证明,确保不同插入顺序下关键统计属性的保持。

尽管上述结构的正确性已通过形式化验证,但设计中未纳入数据优先级维度,难以支持基于优先级的高效检索,限制了其在优先级敏感场景的应用。Nipkow 等人[2] 提出的函数式静态优先级搜索树既明确包含优先级维度,又通过函数式建模与机械化验证保障可靠性,显著提升高优先级数据搜索效率。但该结构存在关键局限:数据优先级固定,在访问频率未知时,频繁访问低优先级数据会导致搜索效率下降。

本文提出了一种基于动态优先级的 FAPST,能够根据访问频率实时更新节点优先级,提高了对频繁访问数据的检索效率,并进一步引入热点转移机制优化了数据存储结构,有效解决历史热点滞留与新热点路径阻塞问题,增强 FAPST 对热点转移场景的适应性。其核心操作在 Isabelle 中进行了形式化建模和验证,确保了结构正确性与功能正确性。实验结果表明,FAPST 在高频访问场景下优于 PST,在热点频繁转移场景中更显优势,突显了其在实际应用中的适配性。为明确 FAPST 的创新价值,本文选取节点优先级机制、形式化验证、最坏情况时间复杂度及动态访问场景适配能力等关键维度,与代表性相关工作进行系统性对比,详见表 1。

Table 1 Comparison of FAPST and Related Work

表 1 FAPST 与相关工作的对比

对比维度	FAPST(本文)	PST[2]	AAPST[3]	Splay[22]
节点优先级	动态	静态	动态	不支持
形式化验证	支持	支持	不支持	支持
最坏时间复杂度	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$
访问敏感场景	支持	不支持	支持	不支持
热点转移场景	支持	不支持	不支持	不支持

2 FAPST 结构的函数式建模

FAPST 是基于函数式建模与机械化验证的动态优先级红黑树结构,在维持结构平衡及可控最坏检索复杂度的基础上,引入自适应优先级机制优化访问敏感数据的检索性能。数据项通过元组分配优先级,可在插入时显式定义,也能依据访问频率自适应调整。本节介绍其函数式建模内容,涵盖数据结构定义及查找、插入、删除 3 种主要基本操作。

2.1 FAPST 结构的函数式定义

本文借助 Isabelle 的归纳类型 datatype 自定义 FAPST 结构,具体定义为:

```
datatype tcolor = Red | Black
type_synonym 'k psrbt = "(('k × nat)×
(tcolor × ('k × nat)))tree"。
```

FAPST 结构通过类型别名 type_synonym 将'k tree 重定义为'k psrbt 类型,其核心组成包括 3 个属性:

1)主节点对。每个节点存储一个主键-优先级对,类型为 ('k × nat),其中'k 是键的数据类型,nat 表示其优先级。例如,在图 2 中,根节点存储 (4, 1),其中 4 是键,1 是其初始优先级。所有节点的这些键的集合构成了树的主键集。

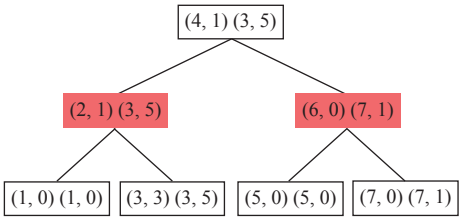


Fig. 2 Structural diagram of FAPST

图 2 FAPST 结构图

2)辅助节点对。同样采用 ('k × nat) 类型,用于存储当前子树中的最大优先级信息。例如图 2 中根节点的辅助对 (3, 5) 表示该子树中存在键值为 3、优先级为 5 的节点,且 5 为当前子树的最高优先级。

3)颜色属性(tcolor)。含 Red、Black 两种取值,用于强制执行红黑树不变性,通过约束规则(规则 3-5)保障树的平衡性。

FAPST 内部节点需满足五大核心规则,涵盖搜索树、优先级搜索树及红黑树变体的核心性质:

规则 1(搜索树性质)FAPST 主节点对数据的中序遍历产生一个线性升序的列表。

规则 2(优先级性质)辅助节点优先级为当前节点及其子树辅助节点优先级的最大值。

规则 3(红黑约束)任意内部节点路径中无连续红色节点。

规则 4(黑高一致性)任意内部节点到各叶子节点的路径中,黑色节点数量一致。

规则 5(根节点为黑)根节点的颜色为黑色。

2.2 FAPST 基本操作的函数式建模

FAPST 的设计确保查找、插入、删除 3 个基本操作严格遵循 FAPST 的 3 项核心规则：查找操作定位目标数据时，会在遍历中动态提升其优先级；插入操作整合新数据及优先级，兼顾搜索树主节点有序性与辅助节点最大优先级属性；删除操作移除目标数据后，会执行重平衡操作，维持红黑树着色约束、子树优先级一致性、黑色高度一致性等不变性。

2.2.1 查找操作

在 FAPST 中，查找操作采用“自顶向下查找，自底向上调整”的双阶段处理模式，如图 3 所示。自顶向下查找阶段：从根节点递归比较键值，匹配则返回节点，否则按二叉搜索树有序性遍历左/右子树，时间复杂度为 $O(\log n)$，与标准二叉搜索树一致。自底向上调整阶段：找到目标节点后，通过路径回溯逐层重构搜索路径节点，依据子树状态和优先级信息调整节点颜色与辅助节点对信息，确保满足 5 项核心不变性，同时实现优先级动态更新。该双阶段机制既保证高效定位数据、维持树结构平衡，又优化后续访问效率。

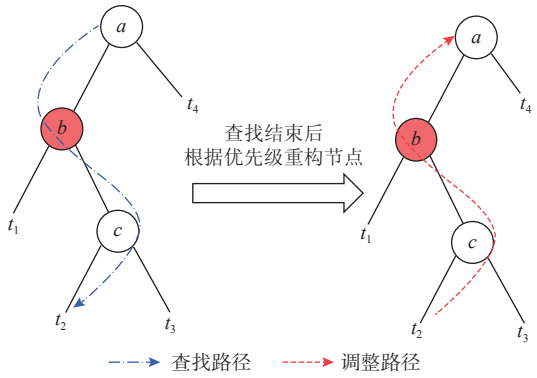


Fig. 3 Processing pattern of search operation

图 3 查找操作处理模式

查找操作在 Isabelle 中形式化为 search 函数，参数为'k psrbt 类型的 FAPST 结构和'k(可排序)类型数据，返回 ('k psrbt * bool) 二元组(处理后的树结构，查找是否成功)，建模如下：

```
//FAPST 的查找函数
fun search :: " 'k psrbt ⇒ 'k::linorder ⇒ 'k psrbt * bool "
where
  " search Leaf x = (Leaf, False) "|
  " search (Node l((a, b), c, (mk, mp)) r) x = (if cmp mk
x = EQ
  then (Node l((a, b), c, (mk, Suc mp)) r, True)
  else (case cmp x a of
    EQ ⇒ (mkNode c (mk, mp) l (a, Suc b) r, True)|
    LT ⇒ (mkNode c (mk, mp) (fst(search l x)) (a, b) r,
(snd(search l x)))|
    GT ⇒ (mkNode c (mk, mp) l (a, b) (fst(search r x)),
(snd(search r x))))) "
```

上述建模中 *mkNode* 函数用于重构新 FAPST 结构，*search* 函数对 FAPST 结构归纳：1)匹配空树时返回 (Leaf, False)。 2)匹配非空树时，先比对根节点辅助节点的整树最高优先级数据 *mk* 与 *x*，若匹配则提升 *mk* 优先级并返回 (新树, True)；否则继续比对根节点主节点数据 *a*，若 *a* 与 *x* 匹配则提升 *a* 优先级并返回 (新树, True)。若 *mk* 与 *a* 均未与 *x* 匹配则按 *x* 与 *a* 的大小递归遍历左右子树，返回子树查找结果构建的新树及查找状态。

图 4 对比了 FAPST 与 PST 对关键字“5”的查找过程：PST 需 3 次比对定位目标节点，且不修改树结构，再次查找仍需相同步骤。FAPST 先比对根节点辅助节点 (2, 1)，不匹配，按主键大小进入右子树再向左子树递归，最终匹配节点 (5, 1) 并将其优先级提升为

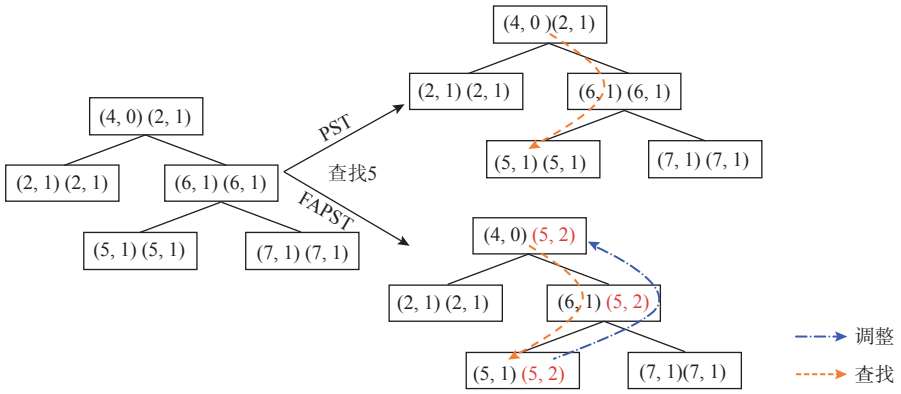


Fig. 4 Comparative example of search operations in FAPST and PST

图 4 FAPST 和 PST 查找操作实例对比

(5, 2)。随后系统检查路径是否满足规则 2(父节点辅助节点优先级不小于子树最大优先级),因当前节点优先级高于父节点辅助节点,遂向上传递更新父节点及根节点辅助节点为 (5, 2),有效缩短后续查找路径。

2.2.2 插入操作

插入操作将关键字 x 及其优先级 y 整合为新 FAPST,在 Isabelle 中形式化为 *update* 函数:参数为可排序类型'k、nat 类型优先级及'k psrbt 类型 FAPST,返回'k psrbt 类型新结构。其中 *upd* 负责递归插入与局部调整,*update* 在其结果基础上通过 *paint* Black 保证根节点为黑色。其函数式建模如下:

```
//FAPST 的插入函数
fun upd :: " 'k::linorder ⇒ nat ⇒ 'k psrbt ⇒ 'k psrbt " where
  " upd x y Leaf = mkR (x, y) Leaf (x, y) Leaf " |
  " upd x y (B _ l (a, b) r) = (case cmp x a of
     LT ⇒ baliL (upd x y l) (a, b) r |
     GT ⇒ baliR l (a, b) (upd x y r) |
     EQ ⇒ mkB (x, y) l (x, y) r) " |
  " upd x y (R _ l (a, b) r) = (case cmp x a of
     LT ⇒ mkR (a, b) (upd x y l) (a, b) r|
     GT ⇒ mkR (a, b) l (a, b) (upd x y r)|
     EQ ⇒ mkR (x, y) l (x, y) r) "
definition update :: " 'k::linorder ⇒ nat ⇒ 'k psrbt ⇒ 'k psrbt " where
  " update x y t = paint Black (upd x y t) "
```

插入采用自顶向下递归策略:1)待插入子树为空时,将 (x, y) 设为节点主/辅助节点信息,节点置为红色;2)子树为非空时,若 x 已存在则更新其优先级为 y,否则递归遍历至空子树执行 1)。过程可能破坏规则 3 和 4,需同步执行平衡与颜色调整,调整至根节点时,若根为红色则通过 *paint* 操作强制转为黑色,确保满足规则 5。

2.2.3 删除操作

删除操作移除指定键节点并生成新 FAPST,在 Isabelle 中形式化为 *delete* 函数:参数为可排序类型'k 的键及'k psrbt 类型树,返回'k psrbt 类型新树。其中 *del* 实现递归删除与局部调整,*delete* 在其结果基础上通过 *paint* Black 保证根节点为黑色。删除核心思路为:空树直接返回;非空树遍历定位目标节点,通过 *combine* 函数合并左右子树完成删除。具体函数式建模如下:

```
//FAPST 的删除函数
fun del :: " 'k::linorder ⇒ 'k psrbt ⇒ 'k psrbt " where
  " del x Leaf = Leaf " |
  " del x (Node l ((a, b), (c, _)) r) = (case cmp x a of
     LT ⇒ if l ≠ Leaf ∧ color l = Black then baldL (del x l) (a, b) r else mkR (a, b) (del x l) (a, b) r|
     GT ⇒ if r ≠ Leaf ∧ color r = Black then baldR l (a, b) (del x r) else mkR (a, b) l (a, b) (del x r)|
     EQ ⇒ combine l r) "
definition delete :: " 'k::linorder ⇒ 'k psrbt ⇒ 'k psrbt " where
  " delete x t = paint Black (del x t) "
```

删除通过自顶向下递归执行:1)空树直接返回;2)非空树中,若 x 存在则递归定位后调用 *combine* 合并子树替换目标节点,若不存在则递归至空子树执行 1)。操作后需自底向上平衡与颜色调整以满足规则 3 和 4;与插入类似,根节点可能为红色,需通过 *paint* 操作强制转黑确保规则 5 成立。

3 FAPST 的 Isabelle 机械化验证

FAPST 操作函数的验证包含终止性证明与正确性验证 2 个核心层面。对于终止性证明,本文第 2 节的函数式建模均基于 Isabelle 的 fun 与 definition 机制定义,其终止性已通过内置验证工具自动校验通过。工具采用自动化证明策略,核心是证明存在某一良基关系,使函数递归调用的参数沿该关系严格递减,直至抵达递归基例,确保所有函数在任意有效输入下均能在有限步内终止,而非无限递归。对于正确性验证,本节将依托 Isabelle 推理能力,通过验证结构特性不变性与元素特性不变性,确保查找、插入、删除操作的正确性。

3.1 FAPST 的结构特性不变和元素特性不变

为确保 FAPST 结构及其操作的正确性与稳定性,本文在 Isabelle 中完成 FAPST 关键不变性的机械化验证。不变性是指操作或状态转换中始终成立的核心约束,是数据结构动态变化的正确性保障。通过形式化定义并验证这些不变性,可确保 FAPST 操作过程中维持设计核心特征,其验证核心为结构特性不变与元素特性不变 2 类属性。

1)结构特性不变。结构特性不变确保所有操作(优先级调整、结构重构、路径平衡等)执行前后,树的核心结构规则(规则 2~5)始终成立,其形式化定义为结构不变式:

definition *inv_rb_apst* :: " 'k psrbt ⇒ bool " where

" *inv_rb_apst t* = (*invpst t* ∧ *invc t* ∧ *invh t* ∧ *color t* = Black) "

该不变式与规则 2~5 严格对应：*invpst* 对应规则 2(节点辅助信息为自身及子树优先级最大值，且子树递归保持此性质)；*invc* 对应规则 3(无连续红色节点，且在所有子树中递归成立)；*invh* 对应规则 4(所有路径黑色节点数量一致)；顶层属性 *color t* = Black 对应规则 5(根节点为黑色)。

2)元素特性不变。元素特性不变是操作功能正确性的核心判据，要求操作后未被直接操作的元素在顺序及取值上保持不变。验证策略如图 5 所示，将 FAPST 通过中序遍历转为等价升序列表，对列表执行对应操作，再将直接操作 FAPST 后中序遍历得到的列表与之比对，通过这种与规则 1(中序遍历升序性)相呼应的方式，严格验证元素特性不变性。

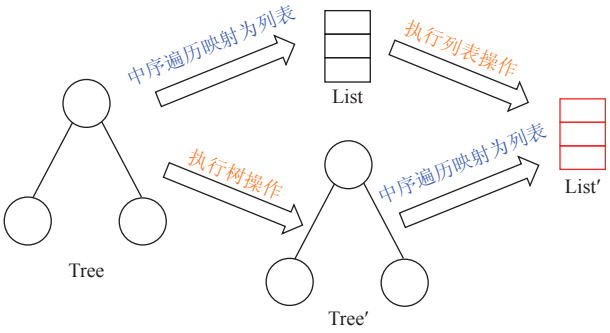


Fig. 5 Proof strategy for element property invariance

图 5 元素特性不变证明策略

上述结构特性不变(规则 2~5)与元素特性不变(规则 1)构成 FAPST 完整约束框架，Isabelle 机械化验证可确保所有操作执行前后这些核心性质均被严格维持，从而保障结构与操作的正确性和稳定性。

3.2 FAPST 操作函数的正确性验证

本节围绕 FAPST 操作过程中的结构特性不变与元素特性不变进行了系统的形式化定义与验证，旨在确保 FAPST 在不同操作场景下的正确性与一致性。具体而言，本文在 Isabelle/HOL 中对 FAPST 的结构属性进行了严格建模，随后针对查找、插入与删除 3 种基本操作函数开展结构特性不变的机械化验证，确认这些操作不会破坏树的有序性与层次约束。同时，本文对元素特性不变进行了形式化刻画，并逐一验证各操作在执行过程中是否保持数据元素的核心属性与语义稳定性。通过上述多层次的验证分析，为 FAPST 的函数式建模提供了全面形式化保障。

验证过程解决了三大关键挑战：首先，FAPST 操作存在多分支场景，需系统性拆解数据匹配情形以保障验证完备性；其次，查找操作的自适应优先级调整机制可能破坏规则 2，引发树结构连锁调整与递归重构，显著提升证明复杂度；最后，为支撑整体验证框架，构建了多层级递进式关键引理体系，需通过逐步搭建辅助引理化解难点，确保验证严谨性。

3.2.1 查找函数的正确性验证

当 FAPST 满足规则 1(中序有序)与规则 2(优先级约束)时，查找操作仅存在成功与失败 2 种情形，分别由定理 1、定理 2 描述：

定理 1. 查找成功。theorem search_rb_priority: "*invpst t* ⇒ *sorted*1 (*inorder t*) ⇒ *snd* (*search t x*) ⇒ *get_option_value* (*get_kp* (*fst* (*search t x*)) *x*) = *Suc* (*get_option_value* (*get_kp t x*)) "。

定理 2. 查找失败。theorem search_rb_priority_no: "*invpst t* ⇒ *sorted*1 (*inorder t*) ⇒ ¬*snd* (*search t x*) ⇒ *get_option_value* (*get_kp* (*fst* (*search t x*)) *x*) = *get_option_value* (*get_kp t x*) "。

定理 1 描述验证查找成功场景下的优先级调整特性，当搜索树满足规则 1 与规则 2 的前提下且查找操作命中关键字 *x* 时，目标数据元素优先级将上升 1。定理 2 则验证查找失败场景下的优先级稳定性，若查找操作未匹配到目标元素，则数据元素的优先级保持不变。其中，函数 *get_option_value* 用于从 option 类型单子结构中提取具体值，*get_kp* 用于获取指定键对应的优先级。

图 6 展示了定理 1 的证明过程，该定理的证明通过函数归纳展开，平凡分支经 auto 方法自动化简化后，再拆分左右子树情形。验证依赖分层构造的辅助引理，这些辅助引理为核心推理提供支持以确保证明严谨性，引理 1~4 的具体描述如下。

引理 1. lemma mkNode_Node3: "*invpst l* ⇒ *invpst r* ⇒ ∀*y* ∈ set (*tree2 l*) ∨ set (*tree2 r*). *snd y* ≤ *mp* ⇒ *mk* = *a* ⇒ *get_option_value* (*get_kp* (*mkNode c* (*mk*, *mp*) *l* (*a*, *b*) *r*) *a*) = *b* "。

引理 1 表明，在使用重构函数进行重构后的树中，左右子树均满足规则 2，且根节点的辅助节点不高于左右子树中任一节点的优先级，若辅助节点键值等于当前主键 *a*，则 *get_kp* 函数可直接返回主键 *a* 对应的优先级 *b*。

引理 2. lemma invpst_search: "*invpst t* ⇒ *invpst* (*fst* (*search t x*))"。

引理 2 表明，在执行查找操作前后，树中辅助节

```
theorem search_rb_priority: "invpst t ⟹ sorted1(inorder t) ⟹ snd(search t x) ⟹
                  get_option_value(get_kp (fst(search t x)) x) = Suc(get_option_value(get_kp t x))"
 apply(induction t x rule:get_kp.induct)  函数归纳
 apply (auto split:if_splits prod.split )
 apply(auto simp add: is_max2_def mkNode_Node3)  引理 1
                                                          引理 3      引理 4
 apply (metis (no_types, lifting) sorted_wrt_append Un_iff invpst_search get_kp_lt get_kp_ltk pst_max_Leaf)
 by (metis (no_types, lifting) Un_iff get_kp_rt get_kp_rtk invpst_search pst_max_Leaf sorted_cons sorted_mid_iff)
                                                  引理 2
```

Fig. 6 Proof process of theorem 1

图 6 定理 1 的证明过程

点优先级等于该节点及其子树中所有数据的最大优先级这一性质保持不变，即规则 2 始终成立。

引理 3. lemma get_kp_lt: "*invpst l* ⇒ *invpst r* ⇒ ∀*y* ∈ set (*tree2 l*) ∨ set (*tree2 r*). *snd y* ≤ *mp* ⇒ *mk* = *x* ⇒ *x* < *a* ⇒ *get_kp* (*mkNode c* (*mk*, *mp*) *l* (*a*, *b*) *r*) *x* = *get_kp l x*"。

引理 4. lemma get_kp_rt: "*invpst l* ⇒ *invpst r* ⇒ ∀*y* ∈ set (*tree2 l*). *snd y* ≤ *mp* ⇒ *l* = Leaf ⇒ *mk* = *x* ⇒ *x* < *a* ⇒ *get_kp* (*mkNode c* (*mk*, *mp*) *l* (*a*, *b*) *r*) *x* = *get_kp l x* "。

引理 3 与引理 4 均表明，若经重构后的树满足以下条件，其左、右子树均符合规则 2，查询值 *x* 小于当前主键 *a*，右子树中不存在小于 *x* 的值，且辅助节点不包含 *x*，则 *x* 在整棵树中的优先级与在左子树中的优先级相等。两者的差异在于，引理 4 的前提条件更为严格，对左子树设定优先级约束，且要求左子树为空节点。

查找操作的结构特性不变：要求操作前后树的核心结构性质保持不变，由定理 3 形式化描述，其证明依赖引理 2、5、6、7 细化推理过程。

定理 3. 查找的结构特性不变。 theorem rb_apst_search:"*inv_rb_apst t* ⇒ *inv_rb_apst* (*fst* (*search t x*))"。

定理 3 表明，若查找操作执行前的 FAPST 满足结构不变式 *inv_rb_apst*，则操作后得到的 FAPST 仍保持该不变性。这一性质的保持确保了查找操作不会破坏 FAPST 的固有结构约束，从而维持了树结构的稳定性与一致性。图 7 展示了定理 3 的证明过程，首先展开 *inv_rb_apst* 函数定义，将结构不变式的验证目标转化为一组可形式化验证的子性质。通过引理 2、5、6 和 7 确立了查找操作过程中关于结构保持、颜色一致性和黑高约束的逻辑推理，最终实现了对该定理的形式化验证。

引理 5. lemma invc_search: "*invc t* ⇒ *invc* (*fst* (*search t x*))"。

引理 5 指出，在 FAPST 中，从任一内部节点到叶子节点的路径上不会出现连续的红色节点，并且该性质在查找操作之后仍然保持不变。

引理 6. lemma invh_bheight: "*invh t* ⇒ *invh* (*fst* (*search t x*))"。

引理 6 表明，在 FAPST 中，从任一内部节点到其叶子节点的路径上黑色节点的数量一致，并且该属性在查找操作之后依然成立。

引理 7. lemma color_search:"*color t* = Black ⇒ *color* (*fst* (*search t x*)) = Black"。

引理 7 表明，FAPST 的根节点在查找操作前为黑色，操作后仍保持黑色不变。

查找操作的元素特性不变：元素特性不变要求查找操作不修改节点主节点对的数据及相对顺序，由定理 4 提供核心依据。

定理 4. 查找的元素特性不变。 theorem inorder_search: "*sorted*1 (*inorder t*) ⇒ *map fst* (*inorder* (*fst* (*search t x*))) = *map fst* (*inorder t*)"。

定理 4 确保查找后，FAPST 中数据元素集合及中序遍历逻辑顺序保持不变，即便树因优先级调整发生内部变化，通过 *map fst*(*inorder t*) 提取的主键序列始终一致，从语义层面保障了数据的完整性与有序性，证明过程如图 8 所示。

```
theorem rb_apst_search:
  "inv_rb_apst t ⟹ inv_rb_apst (fst(search t x))"
 apply(simp add:inv_rb_apst_def)  结构不变式定义
 by(simp add:invpst_search invc_search invh_bheight color_search)
                引理 2      引理 5      引理 6      引理 7
```

Fig. 7 Proof process of theorem 3

图 7 定理 3 的证明过程

```
theorem inorder_searchs:
  "sorted1(inorder t) ⟹
   map fst(inorder(searchs t xs)) = map fst(inorder t)"
   apply(induction t xs rule: searchs.induct)
   apply(simp add:mkNode_def)+
   by (simp add: inorder_search)
```

Fig. 8 Proof process of theorem 4

图 8 定理 4 的证明过程

3.2.2 插入函数的正确性验证

插入操作的结构特性不变：插入操作执行前后，FAPST 的结构属性不会被破坏。如定理 5，用以描述插入操作对结构稳定性的保持。

定理 5. 插入的结构特性不变。 theorem rb_apst_update: "*inv_rb_apst t* ⇒ *inv_rb_apst* (*update x y t*)"。

定理 5 表明，若操作前 FAPST 满足结构不变式 *inv_rb_apst*(综合规则 2~5)，则操作后新树仍维持该不变性。这确保插入操作在引入新元素时，不会破坏 FAPST 的固有结构约束，保障数据扩展过程中树结构的一致性与稳定性。

插入操作的元素特性不变：该特性要求插入后原有数据的取值、数量及相对顺序不变，仅新增元素被正确纳入，由定理 6 提供保障。

定理 6. 插入的元素特性不变。 theorem inorder_update: "*sorted*1 (*inorder t*) ⇒ *inorder* (*update x y t*) = *upd_list x y* (*inorder t*)"。

定理 6 确保插入后，FAPST 的中序遍历序列与预期更新结果一致。验证策略如图 9 所示：采用元素特性不变的证明思路，将插入操作后的 FAPST 中序遍历列表，与经 Isabelle 引理库验证的 *upd_list* 函数模拟插入后得到的列表比对。若两者一致，则可确认插入操作既保持原有元素完整性，又维持逻辑顺序的正确性。

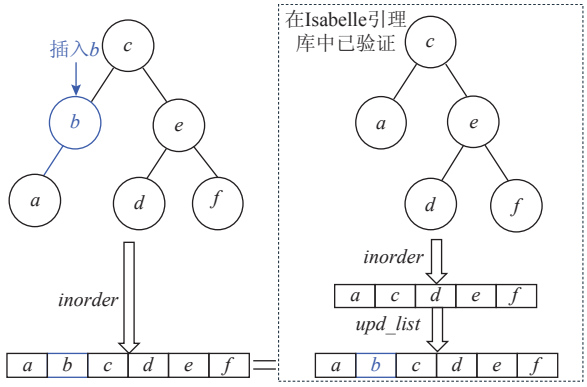


Fig. 9 Verification process of element property invariance in FAPST insertion operation

图 9 FAPST 插入操作的元素特性不变验证过程

3.2.3 删除函数的正确性验证

删除操作的结构特性不变：该特性要求删除操作不破坏树的固有结构约束，由定理 7 形式化验证。

定理 7. 删除的结构特性不变。 theorem rb_apst_delete: "*inv_rb_apst t* ⇒ *inv_rb_apst* (*delete k t*)"。

定理 7 明确，若操作前 FAPST 满足 *inv_rb_apst* 约束(规则 2~5)，则操作后新树仍维持该不变性。这保证删除操作在移除目标元素时，不违反 FAPST 的优先级特性、红黑节点约束、黑高一致性及根节点颜色规定，保障树结构动态修改后的稳定性。

删除操作的元素特性不变：该特性要求删除仅移除目标元素，未被操作元素的取值、数量及相对顺序保持不变，由定理 8 验证。

定理 8. 删除操作的元素特性不变。 theorem inorder_delete: "*sorted*1 (*inorder t*) ⇒ *inorder* (*delete x t*) = *del_list x* (*inorder t*)"。

定理 8 确保删除后 FAPST 剩余节点的中序遍历序列与预期结果一致。验证策略如图 10 所示，采用与插入操作同源的抽象层级比对方法：将 FAPST 删除操作后的中序遍历列表，与经 Isabelle 引理库验证的 *del_list* 函数模拟删除后得到的列表比对。若两者一致，则可确认删除操作仅精准移除目标元素，完整保留数据集合的语义一致性与逻辑顺序。

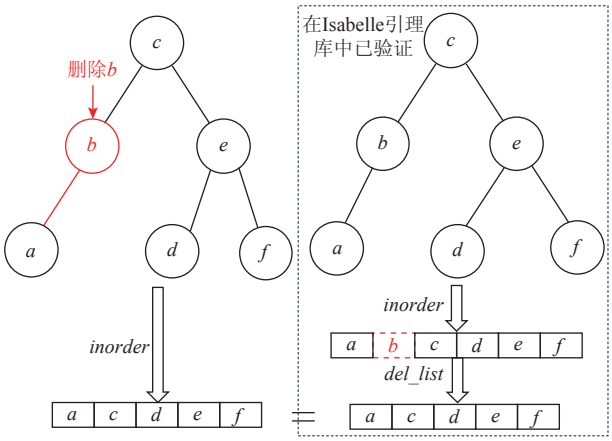


Fig. 10 Verification process of element property invariance in FAPST deletion operation

图 10 FAPST 删除操作的元素特性不变验证过程

4 FAPST 算法的访问优化

本文以云原生环境中的 Kubernetes(K8s)任务调度系统[28]与内容分发网络(content delivery network, CDN)缓存系统[29]为典型应用，分别对应访问敏感场景和热点转移场景。在 K8s 任务调度系统中，调度模块负责在集群节点间分配待执行任务，需持续访问任务队列并依据优先级进行排序[28]。在实际运行

中，少量高优先级任务在特定时间窗口内被频繁触发，体现显著的时序局部性特征，属于典型的访问敏感场景；在 CDN 缓存系统中，内容被分发并缓存至边缘节点，以降低源站压力并提升访问时延性能，系统通常通过缓存替换与命中策略维持高命中率[29]。由于内容热度随时间变化，访问重心在不同时间窗口内持续转移，旧热点衰减、新热点产生，形成阶段性更替，具备显著的访问分布动态演化特征，属于典型的热点转移场景。

上述系统日志通常呈现明显的访问集中特征，即少数对象在不同阶段承担主要访问负载。针对该类动态负载环境下静态搜索结构适应能力不足的问题，本节围绕访问敏感场景与热点转移场景 2 类典型应用，系统展开 FAPST 的函数式建模、复杂度分析与实验验证，评估其在不同访问偏斜程度下的结构自适应能力；同时，借助 Isabelle/HOL 的代码生成机制生成 Haskell 实现，为其工程部署提供实现支撑。

4.1 访问敏感场景

访问敏感场景是指数据访问呈现显著时序局部性特征，即少量对象在短时间内被频繁访问。在云原生环境下的 K8s 任务调度系统中，少量高优先级或周期性任务会在特定时间窗口内被频繁触发，而多数任务仅偶发执行，体现出典型的访问敏感负载特征。当任务优先级未知或随运行动态变化时，静态搜索结构难以适配访问频率演化，检索路径无法随高频任务累积访问而优化。FAPST 在查找过程中动态提升命中节点优先级，并沿路径自底向上调整结构，使高频任务逐步转移至树上层，从而缩短后续访问路径。围绕该类调度负载，将对访问敏感场景进行建模与复杂度分析，并通过实验评估 FAPST 的自适应优化效果。

4.1.1 访问敏感场景下的建模与验证

在 K8s 任务调度过程中，访问敏感负载通常表现为对部分高优先级任务的批量调度请求，调度效率高度依赖优先级的动态调整。为刻画该动态调整特征，引入动态优先级机制，并构建查找函数 *search* 的序列化扩展模型 *searchs*，用于描述批量调度过程中的结构调整机制，定义如下：

fun *searchs* :: " 'k psrbt ⇒ 'k::linorder list ⇒ 'k psrbt" where

"*searchs t* [] = *t*"|

"*searchs t* (*x*#*xs*) = *searchs* (*fst*(*search t x*)) *xs*"

searchs 输入 FAPST 结构与待查关键字序列，输出优先级调整后的 FAPST：空序列返回原树，非空序列先执行单次查找更新树，再递归处理剩余序列，实现高频关键字累积提权与位置调整。为保障其可靠性，基于单次 *search* 操作的正确性，验证其结构与元素特性不变性。

定理 9. 批量查找操作的结构特性不变。theorem *rb_apst_searchs*: "*inv_rb_apst t* ⇒ *inv_rb_apst*(*searchs t xs*)"。

定理 9 是 *searchs* 的结构稳定性核心保障，若初始 FAPST 结构 *t* 满足结构不变式 *inv_rb_apst*，则处理任意序列 *xs* 后仍满足。如图 11 所示，其证明依托前文 *search* 函数的结构不变性，通过访问序列长度归纳，推导批量迭代中结构约束始终成立。

定理 10. 批量查找操作的元素特性不变。theorem inorder_searchs:"*sorted1*(*inorder t*) ⇒ *map fst* (*inorder* (*searchs t xs*)) = *map fst*(*inorder t*)"。

定理 10 是 *searchs* 操作正确性的形式化保障，若初始 FAPST 的中序遍历序列有序，则经 *searchs* 处理任意关键字序列后，中序遍历取主节点键(*map fst*)的结果与初始完全一致。如图 11 所示，其证明依赖前文已验证的 *search* 函数及其相关辅助引理。

4.1.2 访问敏感场景下的复杂度分析

本节从时间复杂度与空间复杂度 2 个方面，对

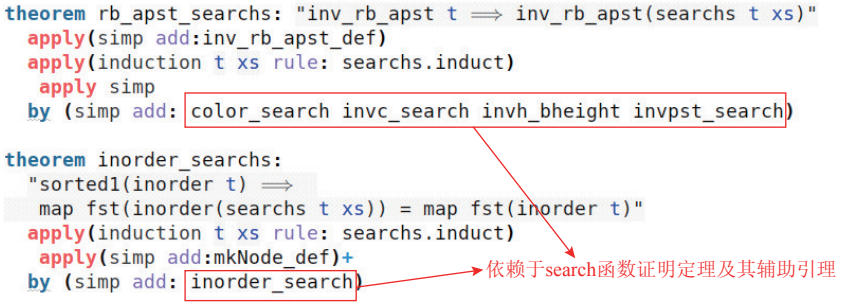


Fig. 11 Proofs process of theorem 9 and theorem 10

图 11 定理 9 和定理 10 的证明过程

FAPST 在访问敏感场景下的性能开销进行理论分析。

1)时间复杂度。访问敏感场景的核心特征是任务访问偏斜，少量高频任务占据绝大部分访问请求。在 K8s 调度场景中，设调度队列中任务规模为 n，其中高频触发任务集合为 H，规模为 $k(|H|=k$，且 $k \ll n)$。由 FAPST 满足红黑性质可知，总树高 $h_{\text{树高}} < 2\log(n+1)$，任意查找操作的最坏情况时间复杂度为 $O(\log n)$。下面分 3 个阶段进行分析。

①初始阶段。高频任务未转移，存储位置无优势，查找路径长度受总树高约束，此时，高频任务的查找路径长度等于其在树中的实际深度 d，且 d 受总树高约束，满足

$$d \leqslant h_{\text{树高}} < 2\log(n+1) = O(\log n)。$$

由于 FAPST 的查找时间成本与路径长度成正比，设单次查找的常数成本系数为 c，则高频任务子集 H 的单次查找时间成本可表示为

$$cost_{\text{初始}} = c \times d。$$

因此，初始阶段单次查找复杂度为 $O(\log n)$。

②过渡阶段。高频任务经多次访问优先级提升，逐步向树上层迁移形成高频子树 B (B 树高 $h<2\log(k+1)$)，查找路径长度递减，复杂度从 $O(\log n)$ 逐步向 $O(\log k)$ 收敛。设总访问次数为 C，前 m 次访问处于初始阶段($m \ll C$，即初始阶段持续的访问次数远少于总访问次数)，则累计查找时间复杂度可拆分为初始阶段与过渡阶段 2 部分：

$$\sum_{i=1}^{C} cost(i) = \sum_{i=1}^{m} O(\log n) + \sum_{i=m+1}^{C} O(\log k)。$$

由于 $m \ll C$，且 $O(\log k) < O(\log n)(k \ll n)$，过渡阶段的累计复杂度逐步由 $O(\log k)$ 主导，整体呈现从 $O(\log n)$ 向 $O(\log k)$ 收敛的趋势。

③稳定阶段。高频任务完全转移至子树 B，平均查找复杂度为

$$cost_{\text{稳定}} = P(H) \times O(\log k) + (1 - P(H)) \times O(\log n)。$$

在典型的访问敏感场景中 $P(H) \to 1$(高频访问概率较高)，复杂度最终趋近 $O(\log k)$，实现显著优化。

综上对比来看，FAPST 与 PST 虽同为红黑树变体且最坏复杂度均为 $O(\log n)$，但 PST 因优先级静态固定，高频任务无法转移，复杂度始终维持 $O(\log n)$，难以适配访问敏感场景的偏斜特性。而 FAPST 靠动态优先级驱动高频任务转移，查找复杂度呈“初始 $O(\log n)$→过渡收敛→稳定 $O(\log k)$”的阶段性特征，精准契合高频任务集中访问的实际需求，最终效率优于 PST，这正是其动态优先级机制的核心优化价值。

2)空间复杂度。FAPST 在访问敏感负载下的结构调整仅涉及节点优先级更新与局部旋转操作，未引入额外辅助存储结构，其空间复杂度与 PST 一致，故不再展开。

4.1.3 访问敏感场景的访问优化

为验证 FAPST 在访问敏感场景下的性能表现，围绕 K8s 任务调度系统的实际运行特征构建实验模型。在 K8s 调度过程中，调度器需要在待执行任务队列中定位目标任务并完成资源分配。实际运行日志表明，少量高优先级任务会在一定时间窗口内被频繁触发，而多数任务仅偶发调度，呈现明显的访问偏斜特征。为准确刻画这一现实分布特征，实验采用符合幂律分布的访问模型，并基于此自动生成具有不同偏斜程度的数据集：设定全体任务集合为 $T=\{t_1, t_2, \cdots, t_n\}$，其中，高频任务子集为 $H=\{h_1, h_2, \cdots, h_k\}$，全体任务集合 T 除去 H 后剩余的一般任务子集部分为 $S=\{s_1, s_2, \cdots, s_u\}$，对全体任务的总查找次数为 C，其中对高频任务子集的查找次数为 C_H，对一般任务子集的查找次数为 C_S，则查找次数之间满足：

$$C = C_H + C_S。$$

高频任务子集 H 的数据查找次数：

$$C_H = \sum_{i=1}^{k} C(h_i)。$$

一般任务子集 S 的数据查找次数：

$$C_S = \sum_{i=1}^{u} C(s_i)。$$

本文定义高频任务子集访问概率为：

$$P(H) = \frac{C_H}{C}。$$

为评估 FAPST 与 PST 在该调度负载下的性能差异，本文选取查找过程中的关键函数调用次数作为响应时间度量的指标[30]。函数调用是算法执行的基本单元，核心逻辑(键值比对、优先级判断等)均依托其实现，总次数可直接反映执行路径长度与计算量。在此基础上，通过控制 3 个核心变量模拟多场景性能，开展算法效率的全面对比。

1)高频任务访问概率 $P(H)$ 取 30%、60%、90%，对应一般、中等与显著偏斜调度负载。

2)调度队列规模 n 取 30 与 100，覆盖不同规模的任务池。

3)总任务查找次数 C 从 1 000 次递增至 10 000 次，观察长期运行下的结构演化趋势。

通过上述参数组合，系统比较不同调度偏斜度、规模和运行周期下 FAPST 与对比结构的效率表现。

为确保实验结果的可靠性与稳定性，每组实验均独立重复 10 次，取关键函数调用次数的平均值作为统计结果，以降低随机误差对测量结果的影响。

图 12 和图 13 对比展示了在不同调度队列规模与访问偏斜度条件下各结构的性能差异。

1）PST 的检索性能受高频任务访问概率影响较小，关键函数调用次数随访问概率提升无显著变化，这与其静态优先级机制无法响应访问模式变化的特性一致，该机制无法响应访问模式的动态变化，导致查找流程难以适配访问偏斜度的调整，体现了静态结构在动态场景下的局限性。

2）Splay 树性能呈现明显的偏斜依赖特征：当偏斜度 $P(H)$=30% 时效率最低，显著劣于 PST 和 FAPST。随着偏斜度增加，其性能逐步改善，在 $P(H)$=90% 时优于 PST 但仍次于 FAPST。这表明 Splay 树对访问局部性具有一定自适应能力，但结构调整过程波动较大，在中低偏斜调度场景下稳定性不足。

3）FAPST 展现出最优的综合性能，当 $P(H)$=30% 时已优于对比算法，随着 $P(H)$ 提升至 60% 和 90%，FAPST 的优势逐步扩大，且调度次数越多，效率提升越显著。这一现象源于 FAPST 的访问驱动优先级调整机制：高频任务被反复调度后逐步转移至树上层，显著缩短后续定位路径。

表 2 进一步给出了 10 000 次调度请求条件下的性能对比结果。在各访问偏斜度场景中，FAPST 均显著优于 PST，随着 $P(H)$ 从 30% 逐步提升至 90%，其性能提升幅度从 20% 左右持续扩大至 64.8%，充分印证了其对动态访问模式的较强适配性；相较于 Splay 树，FAPST 在低偏斜度场景（$P(H)$≤60%）下优势尤为突出，性能提升超 45%，即便在 Splay 树更擅长的高偏斜度场景（$P(H)$=90%）中，仍保持约 30% 的性能领先；且当数据规模从 n=30 扩大至 n=100 时，FAPST 的性能优势进一步凸显，各场景下提升幅度平均增加约 10 个百分点，展现出在大规模数据场景中的良好适配潜力。

综上，实验结果验证了 FAPST 在 K8s 调度负载下的结构自适应能力：其通过优先级动态调整机制优化搜索路径，在访问偏斜增强与系统规模扩大时性能优势持续提升。相比之下，PST 受限于静态优先级机制难以适应访问模式变化，而 Splay 树虽具备一定自适应性，但在典型偏斜场景下仍逊于 FAPST。因此，FAPST 为高偏斜动态检索负载提供了更优的

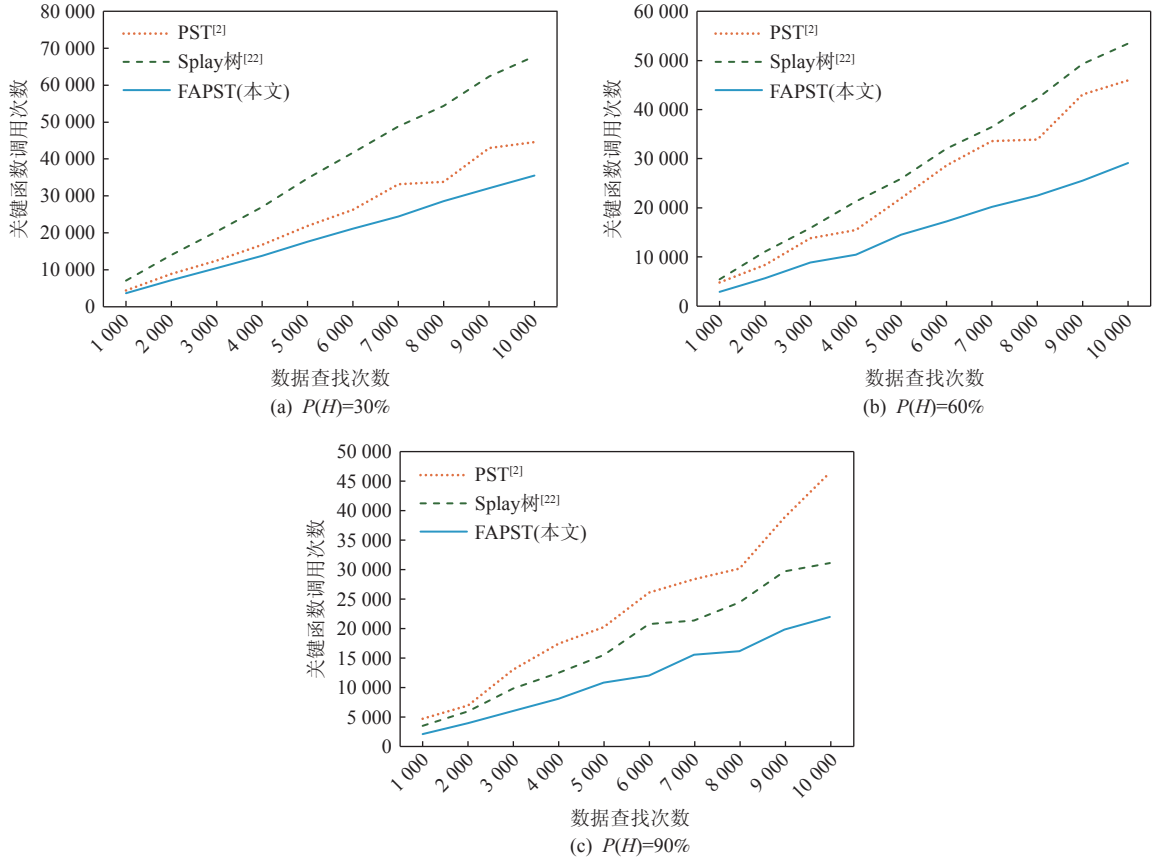


Fig. 12 Key function call performance of different structures under the K8s task scheduling system (n=30)

图 12 不同结构在 K8s 任务调度系统下的关键函数调用表现 (n=30)

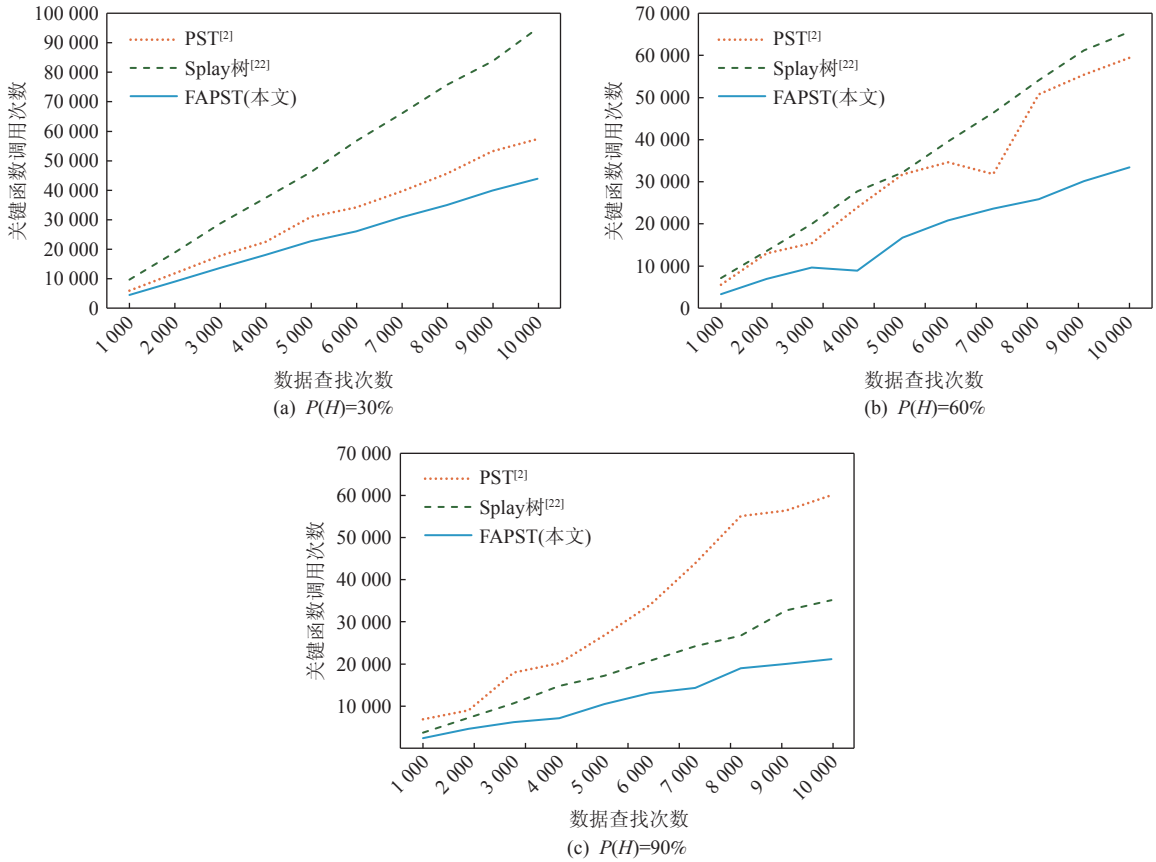


Fig. 13 Key function call performance of different structures under the K8s task scheduling system (n=100)

图 13 不同结构在 K8s 任务调度系统下的关键函数调用表现 (n=100)

Table 2 Performance Comparison of FAPST, PST and Splay over 10 000 Task Searches

表 2 在 10 000 次任务查找下 FAPST 与 PST、Splay 的性能对比

规模 n	$P(H)$/%	PST 关键函数调用次数	Splay 关键函数调用次数	FAPST 关键函数调用次数	FAPST 较 PST 调用次数缩短/%	FAPST 较 Splay 调用次数缩短/%
30	30	44 600	67 800	35 500	20.4	47.6
30	60	46 000	53 400	29 100	36.7	45.5
30	90	46 400	31 100	22 000	52.6	29.3
100	30	57 300	94 900	43 900	23.5	53.7
100	60	59 400	65 600	33 500	43.6	48.9
100	90	60 000	35 200	21 100	64.8	40.1

解决方案。

4.2 热点转移场景

热点转移场景是指访问重心随时间窗口持续转移的动态过程，即原有热点逐步降温、新热点不断生成并占据主要访问负载。在 CDN 缓存系统中，某一阶段的热门内容会集中产生大量请求，而随着关注度变化，访问流量迅速转移至新的内容对象，形成典型的热点转移模式。

FAPST 虽通过动态优先级机制缓解了 PST 的静态优先级适配问题，但其优先级调整主要依赖访问频率累积，缺乏历史热点主动衰减策略。在热点频繁更替环境下，已降温对象可能长期滞留于树结构上层路径，抑制新热点的快速上移。为此，引入热点转移机制，构建 HS-FAPST（hotspot migration FAPST），通过历史热点优先级衰退策略，使新热点能够及时占据高优先级路径，避免路径垄断。图 14 展示了从 PST 到 HS-FAPST 的演进过程，后续将围绕 CDN 缓存负载展开函数式建模、复杂度分析与性能验证。

4.2.1 热点转移场景下的建模与验证

在 CDN 缓存系统中，缓存替换与命中效率高度依赖内容访问分布的动态演化。为刻画该动态演化特征，需在 FAPST 动态优先级机制基础上引入历史

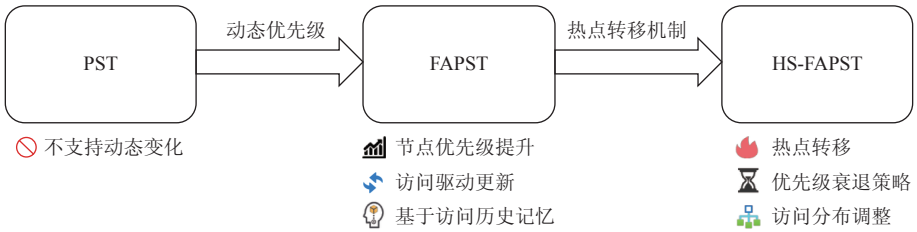


Fig. 14 Evolution process of FAPST structure

图 14 FAPST 结构的演化过程

热点衰减能力。为此，基于 Isabelle 对原结构进行扩展，构建融合全局访问感知的 HS-FAPST。在保持节点结构不变的前提下，引入周期性优先级衰减策略，使历史热点在访问间隔扩大后逐步降权，从而为新热点腾出上层路径空间。为实现上述机制，扩展数据类型如下：

type_synonym 'k apst = "'k psrbt × nat",

其中 'k psrbt 为原 FAPST 主体结构，nat 类型的全局计数器用于累计自上次衰减后的访问次数，从而实现对衰减时机的统一控制。针对历史热点优先级长期居高不下的问题，设计批量衰减函数 *tree_div*，对全树节点优先级进行同比缩减（递归将优先级更新为原值的一半），具体函数建模如下：

```
fun tree_div :: " 'k psrbt ⇒ 'k psrbt " where
  "tree_div Leaf = Leaf " |
  "tree_div (Node l ((a,b), c, (mk, mp)) r) =
(Node (tree_div l) ((a, b), c, (mk, mp div 2)) (tree_div r))"
```

衰减函数 *tree_div* 对所有节点优先级进行同比放缩后，不会改变 FAPST 的原有性质，即便遇到奇偶数并列的情况，也能满足其性质约束。例如，若根节点优先级 *mp* 为 7、子树某节点 *mp* 为 6，经衰减后 *mp* 均变为 3，依旧满足原本的优先级性质（规则 1），不需要对经过优先级衰减的 FAPST 调整节点位置。

在优先级衰减函数 *tree_div* 的基础上，定义触发衰减策略的逻辑操作，*search_new* 函数作为扩展接口，在保留 *search* 核心功能的基础上，新增全局计数器 count 与阈值 MAX 的比较逻辑，实现衰减时机的动态调控，具体函数建模如下：

```
fun search_new:: "'k apst ⇒ 'k::linorder ⇒ 'k apst" where
  "search_new (Leaf, count) x = (Leaf, Suc count)"|
  "search_new (t, count) x = (if cmp (Suc count)
MAX = GT
    then (fst(search (tree_div t) x), 0)
    else (fst(search t x), Suc count))"
```

融合热点转移机制的核心操作由 *tree_div* 与 *search_new* 协同完成，其优先级动态调控机制如图 15 所示。*tree_div* 函数承担历史优先级批量衰减的核心功能，其设计受到计算机网络 TCP 协议中“乘法减小”策略[31]的启发。该策略的主要思想是在网络拥塞时通过“阶跃式减半”快速减少数据发送量，平衡过载避免性能波动。值得注意的是，本文并非对该策略进行简单移植，而是在应用对象与约束条件方面针对 FAPST 的结构特性进行了定制化适配设计。

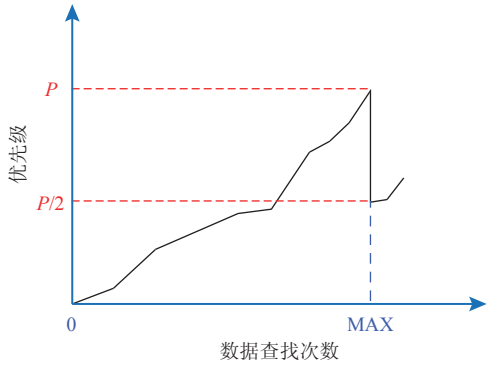


Fig. 15 Priority decay strategy

图 15 优先级衰退策略

在应用对象方面，TCP 协议作用于网络层的流量控制，而 HS-FAPST 的 *tree_div* 则面向数据结构内部的优先级调度。具体到本设计，*tree_div* 通过递归遍历 FAPST 节点，在访问次数触达预设阈值 MAX 时，将所有节点优先级从当前值 *P* 阶跃式衰减至 *P*/2。该操作的核心目标是为潜在的新热点释放优先级资源，从而实现热点数据的平滑转移。为适配不同应用场景，阈值 MAX 通常依据访问模式（如访问的局部性强度、高频项比例）预先调优，针对不同场景的访问特征，可通过离线分析历史访问数据统计热点数据生命周期，将 MAX 初始值设为该场景下热点生命周期内平均访问次数的 1~1.5 倍。

在约束条件方面，TCP 的窗口调整仅需遵循协议栈行为约束，而 *tree_div* 的操作必须严格遵循 FAPST 的结构不变式。具体包括：在优先级衰减过

程中始终保持树的结构稳定性，避免因节点值突变引发重构抖动；同时，衰减操作与 FAPST 的优先级性质（规则 2）深度协同，确保辅助节点中存储的子树最大优先级同步更新，以维持后续动态调整的准确性。这种在强结构约束下的批量衰减策略，既实现了历史热点的“软退出”，又通过阶跃式降权减少性能开销，适应热点持续更迭场景。

search_new 作为热点转移入口，通过全局计数器 count 与阈值 MAX 的动态比较调控衰减时机，该逻辑与访问强度深度耦合：访问密集时高频衰减加速历史热点弱化，稀疏时则降低衰减频率以维持结构稳定。具体而言，count 未达阈值时随访问递增以保持优先级和提升连续性；触达 MAX 时先调用 *tree_div* 执行整树优先级衰减，再重置计数器并通过 *search* 处理当前访问，流程契合 FAPST“查找—调整—重构”闭环。*tree_div* 与 *search_new* 操作协同实现网络流控到数据结构调度的适配，保障 FAPST 结构不变式，形成与访问强度耦合的“衰减—提升”动态逻辑。该逻辑可使历史热点在阈值处衰减（优先级降至 *P*/2）后，新热点依托后续访问积累优先级，既为新热点保留上升空间、避免历史热点资源垄断以实现平滑转移，又能平衡资源释放效率与结构稳定性。

HS-FAPST 在原有的查找操作基础上新增了优先级衰减及其对应的衰减阈值触发 2 个关键逻辑。本节基于第 4 节建立的机械化验证框架，仅需证明新增逻辑与 FAPST 原有不变性约束的兼容性，即可确保算法在维持正确性的同时，又实现了热点转移特性。

1）结构特性不变。第 4 节已验证 FAPST 结构稳定性及基础查找函数对结构不变式（含规则 2~5）的保持性，新增逻辑需证明不破坏该不变式，即函数 *tree_div* 与 *search_new* 的执行不违反原有结构不变式，对其具体描述如定理 11~12。

定理 11. 衰减操作的结构特性不变。 theorem rb_apst_div: "*inv_rb_apst t* ⇒ *inv_rb_apst* (*tree_div t*)"。

定理 12. 触发衰减操作的结构特性不变。 theorem rb_apst_search_new: "*inv_apst_new t* ⇒ *inv_apst_new* (*search_new t x*)"。

定理 11 验证 *tree_div* 仅调整优先级而不破坏树结构，定理 12 验证 *search_new* 与原有结构约束兼容，这 2 个定理共同保障 HS-FAPST 结构稳定。图 16 和图 17 分别给出定理 11 和定理 12 的证明过程。

2）元素特性不变。*tree_div* 与 *search_new* 仅修改节点优先级，不影响数据元素的取值、数量及相对顺序。这一性质分别由定理 13 和定理 14 验证。

```
theorem rb_apst_div:"inv_rb_apst t ⟹ inv_rb_apst (tree_div t)"
  apply(simp add:inv_rb_apst_def)
  by (simp add: invpst_div invc_div div_invh tree_div_color)
```

Fig. 16 Proof process of theorem 11

图 16 定理 11 的证明过程

```
theorem rb_apst_search_new:
  "inv_apst_new t ⟹ inv_apst_new (search_new t x)"
proof (cases t)
  case (Pair t_tree count_orig)
  assume inv_t: "inv_apst_new t"
  hence inv_props:
    "invc t_tree" "invh t_tree"
    "invpst t_tree" "color t_tree = Black"
    by (simp add: Pair)+
  show ?thesis
  proof (cases t_tree)
    case Leaf
    then show ?thesis
      by (simp add: Pair)
  next case (Node l a r)
    let ?count' = "Suc count_orig"
    show ?thesis
    proof (cases "cmp ?count' Max = GT") [13 lines]
    next case False [12 lines]
    qed
  qed
qed
```

Fig. 17 Proof process of theorem 12

图 17 定理 12 的证明过程

定理 13. 衰减操作的元素特性不变。 theorem inorder_div: "*sorted*1(*inorder t*) ⇒ *inorder* (*tree_div t*) = *inorder t*"。

定理 14. 触发衰减操作的元素特性不变。 theorem inorder_search_new:"*sorted*1(*inorder t*) ⇒ *map fst* (*inorder*(*fst* (*search_new* (*t*, count) *x*))) =*map fst* (*inorder t*)"。

定理 13 确保 *tree_div* 执行后中序遍历结果不变，保障数据完整性与顺序稳定。定理 14 确保 *search_new* 执行后主节点序列不变，验证其仅调整优先级而不改变元素主节点及排列，从而保障元素特性不变，证明过程如图 18 所示。

4.2.2 热点转移场景下的复杂度分析

本节从时间复杂度与空间复杂度 2 个方面，对 HS-FAPST 在访问敏感场景下的性能开销进行理论分析。

1）时间复杂度。在 CDN 缓存系统中，内容热度随时间演化，访问重心阶段性迁移。为刻画该动态特征，在访问敏感场景分析基础上，进一步纳入高频数据随时间转移的影响，对 HS-FAPST 在热点迁移条件下的时间复杂度进行分析。设总任务规模为 n，其中高频访问任务集合为 H，规模为 k（$|H| = k$，且 $k \ll n$）。热点转移周期为 Th，优先级衰减阈值为 MAX，其中 Th 与 MAX 近似相等，以符合 FAPST 的衰减触发特性。

```
theorem inorder_div:
   "sorted1(inorder t) ⟹ inorder (tree_div t) = inorder t"
proof (induction t)
  case Leaf show ?case by simp
next
  case (Node l x r)
  obtain a b c mk mp where x_def: "x = ((a, b), c, (mk, mp))"
    by (cases x) auto
  from Node.prems have
    sorted_t: "sorted (map fst (inorder l @ (a, b) # inorder r))"
    by (simp add: x_def )
  have sorted_l: "sorted1 (inorder l)"
    using sorted_t sorted_wrt_append by auto
  have sorted_r: "sorted1 (inorder r)"
    using sorted_mid_iff sorted_t by force
  have IH_l: "inorder (tree_div l) = inorder l"
    using Node.IH(1) sorted_l .
  have IH_r: "inorder (tree_div r) = inorder r"
    using Node.IH(2) sorted_r .
  show ?case by (simp add: x_def IH_l IH_r)
qed
theorem inorder_search_new:
  "sorted1(inorder t) ⟹
   map fst(inorder(fst(search_new (t, count) x))) = map fst (inorder t)"
  apply(induction t x rule: search.induct)
   apply(simp add:mkNode_def)+
  by (simp add: inorder_div inorder_search sorted_wrt_append)
```

Fig. 18 Proofs process of theorem 13 and theorem 14

图 18 定理 13 和定理 14 的证明过程

①快速上升期($0 < t \leqslant Th/3$)

快速上升期与访问敏感场景的初始阶段类似,高频数据的查找路径长度等于其在树中的实际深度 d,设单次查找的常数成本系数为 c,则高频数据子集 H 的单次查找时间成本可表示为:

$$cost_{\text{上升期}} = c \times d。$$

因此,快速上升期的单次查找时间复杂度受总树高主导,保持为 $O(\log n)$。

②优化收敛期($Th/3 < t \leqslant 2Th/3$)

此时新热点数据已进入树结构的中上层,查找路径显著缩短。此阶段性能由热点数据规模主导,逐步收敛至最优水平,与访问敏感场景类似,查找成本可表示为:

$$cost_{\text{收敛期}} = P(H) \times O(\log k) + (1 - P(H)) \times O(\log n)。$$

收敛速率受访问概率 $P(H)$ 影响,当 $P(H) \to 1$ 时,查找时间复杂度逼近 $O(\log k)$。

③稳定高效期($2Th/3 < t \leqslant Th$)

新热点数据完全占据优势路径,树结构达到当前周期内的最优布局,性能稳定在最佳水平,查找成本可表示为:

$$cost_{\text{高效期}} = O(\log k)。$$

综合 3 个阶段的表现,HS-FAPST 在完整热点转移周期内的平均查找时间复杂度为:

$$cost_{\text{HS}} = 1/3 \times O(\log n) + 2/3 \times O(\log k)。$$

稳定高效期结束后,全局访问计数触达阈值 MAX,系统自动触发整树优先级减半操作,解除历史热点路径垄断,为新一轮热点转移创造条件,形成“快速上升—优化收敛—稳定高效”的自适应闭环循环。

理论分析表明,HS-FAPST 通过优先级衰减策略实现性能优化,复杂度从 $O(\log n)$ 降至 $O((1/3)\log n + (2/3)\log k)$。相较于文献 [3] 中复杂度为 $O(\log n)$ 的非适配方案,在 $k \ll n$ 时优势显著,为 HS-FAPST 适配热点转移场景提供理论依据。

2)空间复杂度。在热点转移场景下,HS-FAPST 在 PST 模型之外引入常数规模的全局状态变量,用于控制优先级衰减的触发时机。该扩展未增加额外的存储层级,因此不会改变节点数量及树规模的渐进增长特性。由此,算法在保持与 PST 相同渐进空间复杂度的前提下,实现了对动态访问负载的适应性增强,未引入额外的空间开销。

4.2.3 热点转移场景的访问优化

为评估 HS-FAPST 在热点转移负载下的适应能力,本实验以 CDN 缓存系统的周期性访问特征为背景构建动态访问模型。CDN 场景中内容热度呈阶段性变化,访问重心随时间窗口持续转移,因此实验按固定时间窗口重划分高频子集 H。每完成 500 次查找后,根据最新访问统计更新高频集合,以对应缓存周期内的热点更替过程。HS-FAPST 中由 *search_new* 触发的优先级衰减频率约为 1/500,远低于常规访问操作,其额外开销在整体执行过程中占比极小,对实验结果无显著影响。实验固定总访问次数 C 与高频访问概率 $P(H)$,比较 PST、FAPST 与 HS-FAPST 在动态热点负载下的执行效率,并以关键函数调用次数作为查找时间的量化指标。

图 19 展示了 3 种结构在热点动态转移负载下关键函数调用次数随总查找次数变化的趋势,可得以下结论:

1)PST 采用静态优先级,无法响应 CDN 缓存周期内的热点更替。访问频繁的元素可能因初始优先级较低而长期处于结构低层,导致查找路径难以优化。实验表明 PST 查找执行效率为 3 种结构中效率最低的一个,凸显了其在面对动态访问场景时的结构性局限。

2)FAPST 虽支持动态优先级调整,但在热点转移场景中,历史热点的高优先级会持续锁定优势位置,导致新热点上升路径缓慢、响应延迟。例如当 $P(H)$=90% 时,历史热点对路径的垄断性更强,FAPST 的执行效率显著低于 HS-FAPST,暴露出难以应对热点快速迭代的缺陷。

3)HS-FAPST 通过引入周期性优先级衰减策略,模拟 CDN 中内容热度随时间自然降温的过程,使历史热点逐步退居次级路径,为新热点释放结构空间。实验结果表明,随查找次数增加(如 $P(H)$=90% 场景),

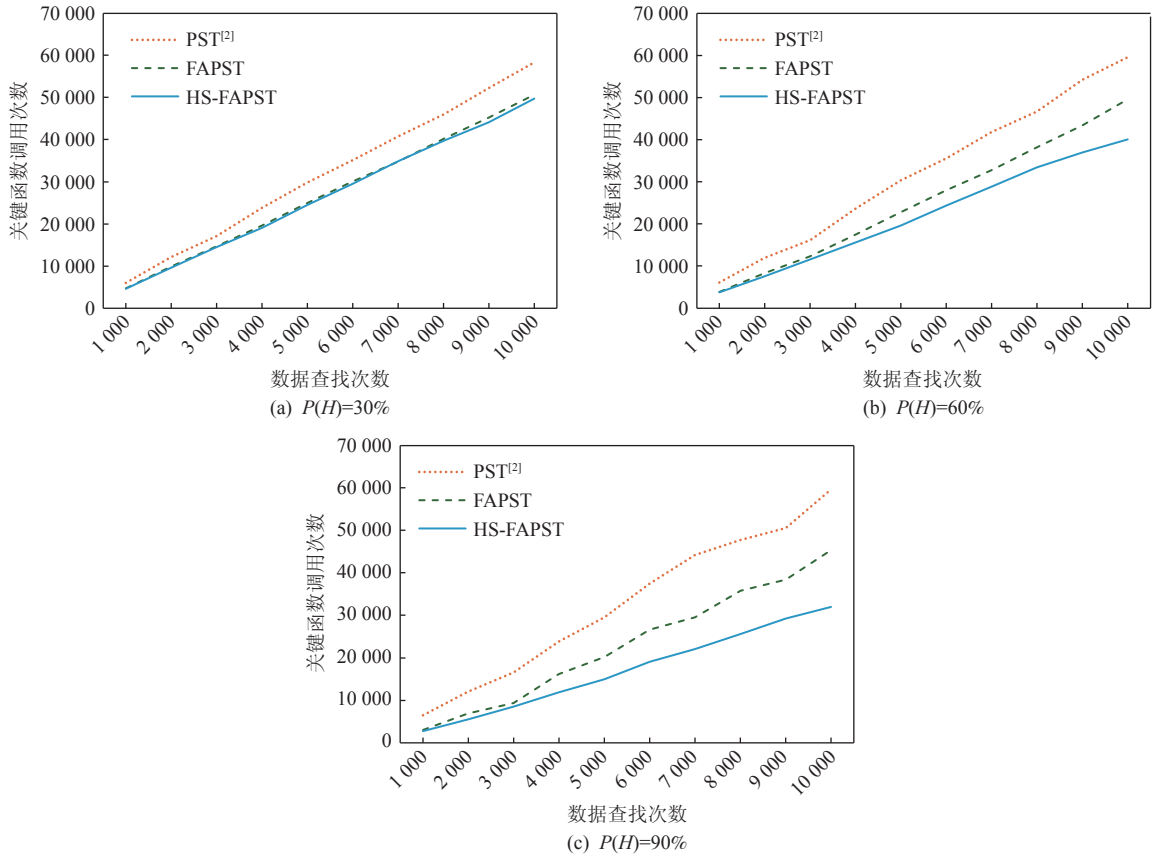


Fig. 19 Key function call performance of different structures under the CDN caching system

图 19 不同结构在 CDN 缓存系统下的关键函数调用表现

与 PST、FAPST 的差距持续扩大，说明针对性引入热点转移机制，可在一定程度上优化热点转移场景访问效率。

表 3 从量化维度对 3 种结构的性能进行了精准对比，在固定 10 000 次数据查找、数据集规模 n=100 的实验条件下，对比了 3 种结构在不同访问偏斜度下的查找操作关键函数调用次数。结果显示，随着高频子集访问概率 $P(H)$ 的上升，HS-FAPST 相较于 PST 和 FAPST 展现出更显著的效率优势。当 $P(H)$=90% 时，HS-FAPST 的关键函数调用次数较 FAPST 缩短约 29.4%，较 PST 缩短约 46.3%。

综上，HS-FAPST 在动态热点迁移场景中表现出了较强的适应能力，特别是在周期性热点更替的过程中。通过周期性优先级衰减策略，HS-FAPST 能够有效缓解历史热点滞留问题，使得新热点能够迅速占据高优先级路径，避免了历史热点对路径的长期占用。实验结果进一步验证了该机制在 CDN 缓存等动态访问负载场景中的适用性，表明其在高频动态负载下能有效提升查询效率。

4.3 不同访问场景优化算法的 Haskell 代码生成

在完成 FAPST 结构的理论分析与形式化验证后，实现形式化模型向工业级可部署代码的转化成为核心环节。本文基于 Isabelle/HOL 验证框架，生成面向不同访问场景的 Haskell 实现代码。选取 Haskell 作为实现语言具备显著的工业化优势，其类型安全以及纯函数式特性与形式化验证模型高度契合，可确保实现代码与验证模型一致；同时，Haskell 已在金融、通信等高可靠系统中广泛应用，成熟的工程实践为

Table 3 Performance Comparison of PST, FAPST, HS-FAPST over 10 000 Data Searches

表 3 在 10 000 次数据查找下 PST、FAPST 和 HS-FAPST 的性能对比

规模 n	$P(H)$/%	PST 关键函数调用次数	FAPST 关键函数调用次数	HS-FAPST 关键函数调用次数	HS-FAPST 较 PST 调用次数缩短/%	HS-FAPST 较 FAPST 调用次数缩短/%
100	30	58 300	50 700	49 700	14.8	2.0
100	60	59 600	49 600	40 100	32.7	19.2
100	90	59 600	45 300	32 000	46.3	29.4

FAPST 在实际部署中提供了稳定基础。本文可通过 Code Generation 方法中的 export_code 命令将访问敏感与热点转移场景下的核心算法自动生成为 Haskell 可执行程序,为进一步应用和工业化打下基础。

5 适用范围

FAPST 的设计目标是解决 PST 在动态访问场景中的局限,当访问分布高度偏斜或热点数据随时间变化时,PST 因优先级固定易导致检索效率下降,而 FAPST 通过动态优先级调整、热点转移机制实现自适应优化,同时依托 Isabelle 完成函数式建模与机械化验证,确保结构正确性及基础操作的最坏时间复杂度为 $O(\log n)$。基于这些核心设计与特性,FAPST 的适用范围可主要划分为 2 类。

1)访问敏感场景。访问分布高度偏斜(少量高频数据占主导),FAPST 通过动态提升高频数据优先级缩短检索路径,确保调整后结构与数据完整性不变。具体适配场景包括:医疗患者信息高频检索、金融热门标的查询、电商爆款检索等。

2)热点转移场景。热点随时间动态转移,FAPST 通过“历史热点衰减+新热点抢占路径”适配需求。经 Isabelle/HOL 机械化验证,该机制执行后树的结构稳定性、数据完整性及逻辑顺序均保持不变,确保检索逻辑无偏差。适配场景包括:短视频热门更新、监控系统异常追踪、新闻热点迭代等。

同样,FAPST 在以下几类场景中仍有优化空间:静态访问模式下,数据访问频率稳定,其动态机制增益有限且存在维护开销,此时 PST 更具简洁性优势。轻量级检索任务(数据规模/访问频次极低),其函数式结构显复杂,数组、哈希表等轻量结构更高效。此外,访问模式极端抖动时,频繁优先级波动可能影响结构稳定性与调度效率。未来将探索自适应阈值与抖动抑制策略,增强其动态环境鲁棒性。

6 总结与展望

本文结合 PST 的研究现状,针对其优先级固定所导致的高频访问项查找性能局限,提出了一种基于动态优先级思想改进的 FAPST 结构。FAPST 在通过函数式建模及机械化验证确保结构及操作正确性与可靠性的同时,通过引入访问频率驱动的优先级演化机制,使节点优先级能够根据访问行为动态更新,从而提升在访问敏感场景下的查找效率。然而,在访问热点频繁变化的应用中,FAPST 仍面临历史热点优先级固化导致阻碍新热点快速响应的问题。为进一步提升结构对热点转移的适应能力,本文在原有 FAPST 基础上引入了热点转移机制,通过历史热点优先级逐步衰减的策略,解决历史热点滞留与新热点路径延迟的矛盾,且该机制的核心操作经形式化验证,确保与原有结构特性的兼容性。

实验结果表明,在访问敏感场景下,FAPST 相较于 PST 在查找效率上展现出一定的改进,引入热点转移机制后,在热点频繁变化的访问负载下,其性能表现进一步优化,初步验证了该结构在复杂动态环境中的适应潜力与应用前景。未来研究可探索将该结构推广至更广泛的应用场景,如多维数据索引、高维检索任务及动态网络拓扑下的数据结构优化问题。

作者贡献声明:左正康和周超共同提出算法思路和实验方案;周超完成实验并撰写论文;柯雨含整理实验数据并完善实验方案;游珍和王昌晶提出指导意见并修改论文。

参 考 文 献

[1] Paulson L C. Computational logic: Its origins and applications[J]. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2018, 474(2210): 20170872

[2] Lammich P, Nipkow T. Priority search trees[EB/OL]. [2025-09-10]. https://www.isa-afp.org/browser_info/current/AFP/Priority_Search_Trees/document.pdf

[3] Massa H, Uhlmann J. Access-adaptive priority search tree[J]. Innovations in Systems and Software Engineering, 2025, 21(4): 1323−1330

[4] Nipkow T, Klein G. Concrete Semantics: With Isabelle/HOL[M]. Berlin: Springer, 2021

[5] Shi Haihe, Lan Sunwen, Liu Riming, et al. Unified formal construction and Isabelle verification of the dynamic programming algorithms for biological sequence alignment[J]. Journal of Computer Research and Development, 2025, 62(1): 119−131(in Chinese)
(石海鹤, 蓝孙文, 刘日明, 等. 生物序列比对动态规划算法的统一形式化构造与 Isabelle 验证[J]. 计算机研究与发展, 2025, 62(1): 119−131)

[6] Matichuk D, Murray T, Wenzel M. Eisbach: A proof method language for Isabelle[J]. Journal of Automated Reasoning, 2016, 56(3): 261−282

[7] McCreight E M. Priority search trees[J]. SIAM Journal on Computing, 1985, 14(2): 257−276

[8] Alistarh D, Kopinsky J, Li J, et al. The spraylist: A scalable relaxed priority queue[C] //Proc of the 20th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2015: 11−20

[9] Seidel R, Aragon C R. Randomized search trees[J]. Algorithmica,

1996, 16(4): 464−497

[10] Icking C, Klein R, Ottmann T. Priority search trees in secondary memory[C] // Proc of Int Workshop on Graph-Theoretic Concepts in Computer Science. Berlin, Springer, 1987: 84−93

[11] Bronson N G, Casper J, Chafi H, et al. A practical concurrent binary search tree[J]. ACM Sigplan Notices, 2010, 45(5): 257−268

[12] Natarajan A, Mittal N. Fast concurrent lock-free binary search trees[C] //Proc of the 19th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2014: 317−328

[13] Afek Y, Kaplan H, Korenfeld B, et al. The CB tree: A practical concurrent self-adjusting search tree[J]. Distributed Computing, 2014, 27(6): 393−417

[14] Ellen F, Fatourou P, Ruppert E, et al. Non-blocking binary search trees[C] //Proc of the 29th ACM SIGACT-SIGOPS Symp on Principles of Distributed Computing. New York: ACM, 2010: 131−140

[15] Wang Zhongbin, Yang Luyi, Cui Shiliang, et al. In-queue priority purchase: A dynamic game approach[J]. Queueing Systems, 2021, 97(3): 343−381

[16] Bose P, Cardinal J, Iacono J, et al. Competitive online search trees on trees[J]. ACM Transactions on Algorithms, 2023, 19(3): 1−19

[17] Drouin A, Hocking T D, Laviolette F. Maximum margin interval trees [C] //Proc of Advances in Neural Information Processing Systems 30. Red Hook: Curran Associates, Inc., 2017: 4947−4956

[18] Sleator D D, Tarjan R E. Self-adjusting binary search trees[J]. Journal of the ACM, 1985, 32(3): 652−686

[19] Wang Ji, Zhan Naijun, Feng Xinyu, et al. Overview of formal methods[J]. Journal of Software, 2019, 30(1): 33−61(in Chinese)
(王戟, 詹乃军, 冯新宇, 等. 形式化方法概貌[J]. 软件学报, 2019, 30(1): 33−61)

[20] Zuo Zhengkang, Ke Yuhan, Huang Qing, et al. Trie+ structural functional modeling, mechanized merification and application[J]. Journal of Software, 2024, 35(9): 4242−4264 (in Chinese)
(左正康, 柯雨含, 黄箐, 等. Trie+结构函数式建模、机械化验证及其应用[J]. 软件学报, 2024, 35(9): 4242−4264)

[21] Nipkow T. Verified root-balanced trees [C]//Proc of Asian Symp on Programming Languages and Systems. Cham: Springer, 2017: 255−272

[22] Nipkow T, Brinkop H. Amortized complexity verified[J]. Journal of Automated Reasoning, 2019, 62(3): 367−391

[23] Hinze R, Paterson R. Finger trees: A simple general-purpose data structure[J]. Journal of Functional Programming, 2006, 16(2): 197−217

[24] Hirai Y, Yamamoto K. Balancing weight-balanced trees[J]. Journal of Functional Programming, 2011, 21(3): 287−307

[25] Nipkow T, Sewell T. Proof pearl: Braun trees[C] //Proc of the 9th ACM SIGPLAN Int Conf on Certified Programs and Proofs. New York: ACM, 2020: 18−31

[26] Nipkow T, Somogyi D. Optimal Binary Search Trees[EB/OL]. [2025-9-10]. https://www.isa-afp.org/browser_info/current/AFP/Optimal_BST/document.pdf

[27] Eberl M, Haslbeck M W, Nipkow T. Verified analysis of random binary tree structures[J]. Journal of Automated Reasoning, 2020, 64(5): 879−910

[28] Burns B, Grant B, Oppenheimer D, et al. Borg, omega, and kubernetes[J]. Communications of the ACM, 2016, 59(5): 50−57

[29] Nygren E, Sitaraman R K, Sun J. The akamai network: A platform for high-performance Internet applications[J]. ACM SIGOPS Operating Systems Review, 2010, 44(3): 2−19

[30] Knuth D E. The Art of Computer Programming, Volume 3: Sorting and Searching[M]. New York: Addison-Wesley, 1999: 426−458

[31] Jacobson V. Congestion avoidance and control[J]. ACM SIGCOMM Computer Communication Review, 1988, 18(4): 314−329



Zuo Zhengkang, born in 1980. PhD, professor, PhD supervisor. Distinguished member of CCF. His main research interest includes high-confidence and intelligent software.

左正康, 1980 年生。博士, 教授, 博士生导师。CCF 杰出会员。主要研究方向为高可信与智能化软件。



Zhou Chao, born in 2001. Master candidate. Student member of CCF. His main research interest includes formal method.

周　超, 2001 年生。硕士研究生。CCF 学生会员。主要研究方向为形式化方法。



Ke Yuhan, born in 1998. Master candidate. Student member of CCF. His main research interest includes formal method.

柯雨含, 1998 年生。硕士研究生。CCF 学生会员。主要研究方向为形式化方法。



You Zhen, born in 1982. PhD, associate professor, Master supervisor. Senior member of CCF. Her main research interests include formal methods, distributed virtual reality, and concurrent distributed computing.

游　珍, 1982 年生。博士, 副教授, 硕士生导师。CCF 高级会员。主要研究方向为形式化方法、分布式虚拟现实、并发分布式计算。



Wang Changjing, born in 1977. PhD, professor, PhD supervisor. Distinguished member of CCF. His main research interests include high-confidence software and intelligent software.

王昌晶, 1977 年生。博士, 教授, 博士生导师。CCF 杰出会员。主要研究方向为高可信软件、智能化软件。