

基于查询热度的密钥索引缓存

金伟^{1,2,3}, 李凤华^{1,2}, 周紫妍^{1,2}, 孙喜洋^{1,2}, 郭云川^{1,2}

(1. 中国科学院信息工程研究所, 北京 100093; 2. 中国科学院大学网络空间安全学院, 北京 100049;
3. 中国信息通信研究院, 北京 100191)

摘要: 在当前的分布式文件系统 (HDFS, Hadoop distributed file system) 密钥管理系统中, 加密区密钥在启动时全部加载至内存, 提供密钥服务。随着密钥资源的增加, 占据的内存空间也随之增长, 带来内存空间不足和密钥索引瓶颈, 如何组织缓存数据、高效处理未命中密钥的查询, 如何调整缓存中的密钥资源, 如何精准预测密钥的使用这三大要点是解决该瓶颈的关键所在。为了实现细粒度高效缓存, 提高密钥使用效率, 从密钥索引数据结构、密钥置换算法、密钥预取策略分析 3 个方面出发, 设计了密钥缓存置换的模块架构, 计算密钥热度, 设置密钥置换算法。具体地, 在密钥热度计算与缓存置换方面, 从密钥所绑定的文件系统和用户出发, 分析影响密钥缓存热度的潜在影响要素, 构建密钥使用热度的基本模型, 采用哈希表与小顶堆链表组合的方式, 维护在用密钥的热度, 基于热度识别设置淘汰算法, 由时间控制器调整密钥使用, 动态更新缓存中的密钥, 从而实现基于热度计算的密钥差异化置换。在密钥预取策略分析方面, 综合考虑业务流程和用户访问存在时间周期维度的规律, 通过日志挖掘获取密钥使用规律, 分析密钥预置策略。实验表明, 所提密钥置换算法可在降低内存占用的同时, 有效提升缓存命中率率和密钥查询效率, 降低密钥文件 I/O 交互对查询性能的影响。

关键词: 密钥管理; 缓存管理; 使用效率计算; 置换算法

中图分类号: TP393

文献标志码: A

DOI: 10.11959/j.issn.2096-109x.2023008

Cache of cryptographic key based on query popularity

JIN Wei^{1,2,3}, LI Fenghua^{1,2}, ZHOU Ziyang^{1,2}, SUN Xiyang^{1,2}, GUO Yunchuan^{1,2}

1. Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

2. School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

3. China Academy of Information and Communications Technology, Beijing 100191, China

Abstract: In the current HDFS (Hadoop Distributed File System) key management system, the encryption zone keys are all loaded into the memory during startup of key service. With the increase of the key resource, the occupied memory space also grows, bringing the bottleneck of memory space and key indexing. There are three

收稿日期: 2022-06-03; 修回日期: 2022-10-18

通信作者: 郭云川, guoyunchuan@iie.ac.cn

基金项目: 国家自然科学基金 (U1836203)

Foundation Item: The National Natural Science Foundation of China (U1836203)

引用格式: 金伟, 李凤华, 周紫妍, 等. 基于查询热度的密钥索引缓存[J]. 网络与信息安全学报, 2023, 9(1): 83-91.

Citation Format: JIN W, LI F H, ZHOU Z Y, et al. Cache of cryptographic key based on query popularity[J]. Chinese Journal of Network and Information Security, 2023, 9(1): 83-91.

challenges induced: how to organize cached data and efficiently handle queries with missed keys, how to adjust key resources in the cache, and how to accurately predict the use of keys. In order to achieve fine-grained and efficient caching and improve the efficiency of key use, key caching optimization was considered from three aspects: key index data structure, key replacement algorithm, and key prefetching strategy. An architecture of key cache replacement module was designed, and then a key replacement algorithm based on the query frequency was set. Specifically, from the perspective of heat computing and key replacement, the potential influencing factors affecting the popularity of key cache were analyzed which considered the file system and user of key management system. Besides, the basic model of key usage popularity was constructed. The hash table and minheap linked list was combined to maintain the heat of the key in use, and the elimination algorithm was set based on heat identification. The key in the cache was dynamically updated, and key usage was adjusted by the time controller, so as to realize key replacement according to the key heat. For key prefetching, key usage rules were obtained through log mining and periodical usage analyzing of key provisioning policies, which considered business processes and the time period dimension of user accessing. Experimental results show that the proposed key replacement algorithm can effectively improve the hit rate of cache queries, reduce memory usage, and ameliorate the impact of key file I/O interaction on query performance.

Keywords: key management, cache management, usage frequency calculation, key replacement algorithm

0 引言

近年来,我国互联网用户规模不断扩大,终端用户不断上升,与之相对应的密码资源和密钥管理需求,也随之增长。Hadoop 分布式文件系统(HDFS, Hadoop distributed file system)中,密钥数量随着用户规模、文件数量的增长,不断增加。截至2021年6月^[1],我国网民规模已达到10.11亿,5G手机终端连接数量达3.65亿,终端用户数量已达亿级。同时,网站、App等互联网资源应用数量众多,我国网站数量约有422万个。每个终端用户都会使用多款应用,即拥有多个应用账户,账户管理的密码资源需求,同样在亿级。

随着密钥资源的增加,其占据的存储空间也随之增大,有限增长的内存空间无法承载无限增长的密钥资源。HDFS的用户在访问加密区中的文件时,同时将使用加密区密钥,加密区密钥若全部放置在磁盘中,I/O开销不能满足海量用户访问时的响应速率,有限的内存空间不可能无限满足密钥数量增长和版本更新的要求。

同时,并非所有密码资源都会以相同的频率使用,不常用资源占用内存,将带来大量不必要的时间成本和空间消耗。密钥的使用取决于文件和文件夹的访问位置,密钥的使用频率与文件的访问频率紧密相关。根据网站访问数据的“二八定律”,即80%的访问集中在20%的数据中,也

就是说,反复使用的密钥不会是全部密钥。根据数据使用频率的“冷热”,给予“热数据”更快的访问资源,将“冷数据”置于更远的位置,降低访问的总体时间成本。

将常用的密钥缓存至内存中,其余密钥驻留在磁盘中,可以平衡密钥的总体访问开销。缓存技术可以缓解内存占用过高的问题,但缓存的查询命中率低时,会增加读取磁盘的I/O时延。具体还需解决如下两个问题。

1) 如何设置缓存、高效处理未命中密钥的查询。缓存的作用在于放置部分使用数据,面临如何组织缓存中的密钥、如何与硬盘中的密钥资源进行调度的问题。缓存中无法放置所有的密钥,即使基于日志、历史记录等进行预测,在实际访问中,也会存在未使用密钥、没有预测预取的情况。如何确保未预取、未命中密钥,在查询中能够快速获取,保证查询响应性能。

2) 如何调整缓存中的密钥资源。密钥资源的查询与使用,有一次性的瞬时使用,也有持续性的长时间使用,这取决于用户访问文件的需求和形式。无论何种方式,如何识别密钥已使用完毕、如何及时将过期密钥迁出缓存、如何维持缓存内的在用内容,需要合理计算。在用密钥的迁出,将导致密钥命中率下降,反复I/O操作读取磁盘;不及时清理过期密钥,将导致缓存空间迅速占满;若长期使用的密钥量超过缓存空间,是否动态调整缓存大小。

本文从密钥使用的影响要素出发, 构建密钥缓存; 计算密钥的使用热度, 设计基于查询热度的置换算法, 动态更新密钥缓存, 降低存储在硬盘中密钥文件 I/O 交互概率, 提高密钥查询速率。主要贡献如下。

1) 针对密钥置换算法, 基于热度识别设置淘汰算法, 动态更新缓存中的密钥。在降低内存占用的前提下, 提升密钥查询效率, 降低密钥文件 I/O 交互对查询性能的影响。

2) 针对密钥使用的影响要素, 从所绑定文件系统和用户的角度出发, 综合考虑密钥的使用规律。

3) 实验表明, 所提密钥置换算法可以有效提升缓存查询的命中率。

1 相关工作

缓存是指通过部分加载至内存的方式, 在保证数据访问速率的同时, 减少内存空间的占用, 通过承上启下的缓存架构, 承接服务查询需求, 打通硬盘数据加载。缓存的三要素是数据结构、置换算法、预取算法。其中, 数据结构是对索引数据的组织; 置换算法是指换出算法, 用于整理内存中已加载的内容, 换出暂时不需要的数据; 预取算法是一种提高命中率的优化方式, 在非预取情况下, 缓存需依靠实时访问来换入, 无法避免第一次的 I/O 操作。

缓存不只是一种技术方案, 许多数据库管理系统中, 已经将缓存提升开发为内存数据库, 因此本节也扩展包含内存数据库、文件系统、元数据负载均衡等与缓存的相关内容。

1.1 经典置换算法

置换算法是指缓存内数据的换出算法, 用于清除缓存中不必要的数据, 减少内存占用。置换的经典算法包括先进先出(FIFO, first-in-first-out)置换算法、最近最少未使用(LRU, least-recently-unused)置换算法、最近最频繁未使用(LFU, least-frequently-unused)置换算法等。

FIFO 是指换出缓存中的最早查询。其优点是算法简单, 类似队列; 缺点是循环查询时的命中率低。为了提高高速缓存置换的速度, Haque 等^[2]针对 FIFO 置换算法提出了一种基于二叉树的高速缓存配置表示的搜索方法, 不仅加快了置换速

度, 还能够精确置换缓存。

LRU 是指淘汰缓存中最近时间最少次数使用; 其优点是短期反复查询的命中率高。为了提高服务的端到端性能, 必须保证键值缓存能够在高吞吐量的情况下实现高命中的问题, Inoue 等^[3]提出了一种多步 LRU 缓存置换算法。这种算法通过使用向量混洗指令对向量中的项目进行重新排序来记录访问历史, 增强了基于向量内 LRU 的朴素组关联缓存, 并且通过单指令多数据流(SIMD, single instruction multiple data)保持效率, 最终实现了更高的缓存命中率。为了分析高速缓存失效问题, Zheng 等^[4]提出了用 LRU 置换算法对 4 种高速缓存失效方案进行建模的方法, 从理论上确定一些指标的关键参数和关于参数设置的解释, 从而平衡缓存失效的性能, 使得该模型在预测方面更加准确、缓存命中率更高。

LFU 是指统计缓存中数据的使用频率, 淘汰使用频率最低的数据; 其缺点是容易淘汰刚刚换入的最新数据, 导致反复 I/O 读取。针对高速缓存回收问题, Matani 等^[5]提出了一种基于 LFU 的缓存回收算法, 这种算法通过维护一个哈希表和一个双向链表, 在缓存回收时时间复杂度为 $O(1)$, 性能比其他算法优越。针对 LFU 置换算法易产生大量存储冗余、网络缓存易被攻击等问题, Liu 等^[6]在研究网络缓存时提出了一个稳定的内网缓存替换策略, 这种策略权衡了每个节点的重要性和缓存内容的流行度, 以便网络访问有更高的缓存命中率和更少的平均跳数。为了实现快速访问数据仓库中的数据, Hassan 等^[7]同时考虑数据的大小和频率, 使用基于老化的 LFU 算法, 计算缓存中数据的年龄, 减少了分析查询响应时间和长期执行引起的数据输入干扰。

最优置换(OPT, OPTimal-replacement), 是指淘汰永不使用, 或未来最长时间中不再被访问。优点是最低的缺页率; 缺点是无法预测, 因此无法实现。随着其他预测算法的兴起, 决策要素的发现, 结合历史信息的预测可以用于优化置换算法。为了实现图的最优缓存置换, 解决缓存局部性差的问题, Balaji 等^[8]提出了一种 P-OPT 置换算法, 这个策略使用图的转置对每个顶点的未来重引用信息进行编码, 从而使置换变得高效、接

近 OPT 的性能。针对减少无线网络中的数据传输问题, Chen 等^[9]根据 OPT 的思想提出了一个基于更新的逐步优化置换算法, 由定量分析的分析模型支撑, 在缓存时优先缓存频繁访问但不频繁更新的数据, 有效提高了命中率并降低了通信开销。

最近经常使用 (MRU, most recently used) 置换算法, 是指首先移除最近最常使用的条目。为了提高非易失性磁随机存储器 (STT-RAM, spin-transfer torque magnetic random-access memory) 的寿命, Priya 等^[10]使用 MRU 置换算法, 基于滑动模式的 ECC 架构, 在高速缓存线上的所有数据位上均匀循环地写 ECC 位, 降低了 ECC 位的磨损程度, 不仅比 LRU 置换算法性能提升了 4 倍, 而且延长了 STT-RAM 的寿命。

自适应缓存 (ARC, adaptive replacement cache) 置换算法由 IBM Almaden 研究中心开发^[11], 核心思想为同时记录 LFU 和 LRU, 并采用 ghost 区域记录短期内被替换的 LFU (2 次以上访问的记录) 或 LRU (只有 1 次访问的记录), 即 ARC 缓存中共有 4 个区域 (ghost MRU、MRU、MFU、ghost MFU)。ghost 区域存放从 MRU 和 MFU 区域淘汰下的内容; 当 ghost 区域被命中时, 调整 LFU 和 LRU 两部分的存储划分比例, 以适应当前系统中的使用模式。此方法可以兼容 LRU 和 LFU 两类算法的优点, 消化两者的缺点。Meint^[12]等在研究多媒体 Web 流量特性时, 使用自适应置换缓存算法, 这种算法巧妙地结合了两个不同大小的 LRU 列表和最近被删除的历史做出移除决定, 弥补了 LRU 替换策略性能低的问题, 不断地对处理的请求变化做出反应, 无须手动设置参数。

1.2 置换算法优化及应用

Butt 等^[13]对现有的 Linux 置换算法、预取算法进行综述和评测, 包括滑动窗口等方式。Osman 等^[14]针对音视频服务中需降低下载次数、减少网络堵塞的需求, 在不同缓存大小的环境中, 实验对比了常用的视频缓存置换算法 (OPT、CC、QC、LRU-2、LRU、LFU、FIFO) 的性能, 结果显示 CC 置换算法可以在最小缓存容量的情况下获得最大命中率和最好性能, 而 FIFO 的命中率最低。

在缓存策略配置方面, 现有替换策略主要基于权重、价值、热度等要素。刘恒等^[15]针对 Spark

采用 LRU 缓存替换算法, 将本质上高重用但最近未使用的块换出缓存的缺点, 提出了基于弹性分布式数据集 (RDD, resilient distributed dataset) 分区的权重缓存替换算法, 考虑 RDD 的计算代价、使用次数、分区的大小生命周期、任务执行时 Locality Level 等因素对权重的影响并进行量化计算, 更准确计算 RDD 分区权重值, 提高内存资源利用率和作业执行效率。

Levandoski 等^[16]针对数据库中的冷热数据识别问题, 分析日志中每个时间段对 Hash 表中每条记录的访问频率, 采用时间序列预测法的平滑指数算法, 通过前向算法计算完整日志记录中的访问频率, 通过后向算法计算最新日志中的访问频率, 以此避免扫描全部日志。同时, 针对并行计算方案, 通过日志片段分割、时间片分割等方式进行前向算法的并行计算, 随后归一化聚集, 通过并行计算初始值、阈值等参数进行后向算法的并发执行。该研究可以有效提高记录访问的命中率。

陈建等^[17]为提高基于密文策略属性基加密 (CP-ABE, ciphertext policy attribute based encryption) 系统的缓存性能, 提出基于最小属性价值的缓存替换算法。结合 CP-ABE 文件的访问策略, 该算法统计高频属性值的个数, 通过高频属性值统计表和余弦相似度方法来计算属性相似度; 同时, 结合属性相似度、文件大小, 计算缓存文件的属性价值, 从缓存中替换属性价值最小的文件。

葛微等^[18]提出了分层式非主键索引方案, 针对 Hadoop HBase 查询效率低、仅支持主键索引的问题, 构建全局索引模型, 设计非主键持久化索引层, 为索引机制提供可扩展性和可靠性; 内存中, 索引热点数据的缓存层, 通过热度累积的缓存替换策略, 提供非主键查询的实时、准确的热点数据缓存; 采用一致性哈希的分布式缓存数据结构, 提高 HiBase 缓存层的可扩展性和容错性。

王文楚^[19]设计并实现了 MQ-LRU 算法, 页面迁移策略通过位置置换而非页面淘汰完成, 提高页面迁移时的内存命中率, 避免淘汰回写至外存的效率损耗, 完成基于脉冲编码调制 (PCM, pulse code modulation) 平行架构的混合内存页面置换。

现有分布式内存文件系统的缓存多设于服务端而非客户端, 导致缓存共享算法在服务端不能

兼顾多租户共享缓存空间的公平与效率,李崇杰^[20]提出了一种细粒度客户端缓存模型,该方法使用子模优化算法识别热数据,并提供同步/异步的缓存替换策略;设计多租户缓存共享算法,提升全局命中率;构建了兼顾服务端和客户端的缓存框架,提供客户端缓存,并支持可插拔的缓存迁移策略。

2 基于热度计算的密钥差异化置换

缓存通过置换算法,维持缓存内容稳定、可用,最小化缓存占用空间。本节通过分析影响密钥的使用要素,提出密钥使用的热度计算方法,差异化换出缓存中的密钥。

2.1 密钥缓存置换模块

密钥缓存的置换模块包括热度计算子模块、热度存储与排序子模块、置换触发子模块,共同完成密钥缓存的维护与管理。热度计算子模块根据实时访问密钥的访问请求,计算缓存内的密钥热度,按照时间周期迭代计算;计算后存储在密钥热度表中,由热度存储与排序子模块管理;置换触发模块监测缓存占用情况以及密钥热度表,缓存占用达到一定比例,或密钥热度降低,均会触发密钥置换,调整缓存内容。缓存置换模块架构如图1所示。

2.2 密钥缓存热度的潜在影响要素

密钥的使用热度受到多种因素影响,这些因素来源于密钥的直接使用、间接使用过程。密钥的直接使用包括管理操作的增删改查,主要与管理员相关。密钥的间接操作,主要来源于用户对所绑定加密区文件加解密时,对密钥的使用,因

此可通过加密区密钥的绑定关系进行判断。

加密区密钥是与文件夹绑定的,因此加密区密钥的热度与用户对文件夹中内容的访问密切相关,即文件/文件夹、用户访问,是两个主要影响因素。根据HDFS的透明加解密规则,加密区不可嵌套,同一路径中只能绑定一个加密区;一个密钥可以对应多个文件夹,多个文件夹的热度作为一个密钥的热度计算来源,如图2所示。

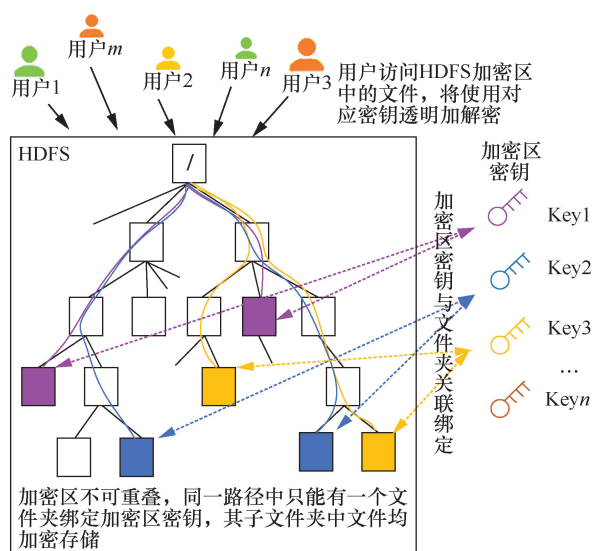


图2 影响密钥使用的因素
Figure 2 Factors affecting key usage

密钥使用的潜在影响要素分析如下。

(1) 文件对密钥访问的影响因素

文件夹和文件自身具有多种特征,并且许多特征会对文件夹的访问概率,即对密钥的访问概

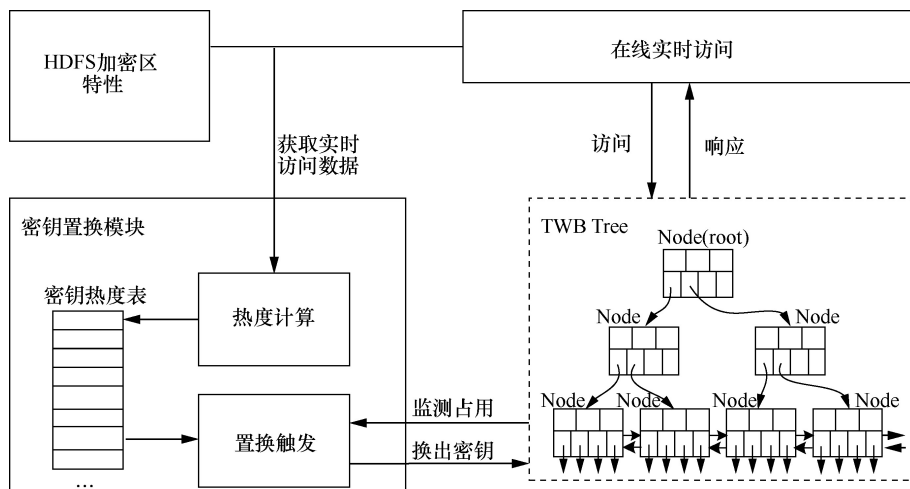


图1 缓存置换模块架构
Figure 1 Architecture of cache replacement

率和使用热度产生影响,包括但不限于如下内容。

1) 文件夹的平均创建/修改时间。根据冷热数据调查,数据使用热度随数据产生/访问的时间变化,大部分数据的创建/修改时间越早,访问概率越小,所绑定密钥的使用热度越小,反之使用热度越大。

2) 文件夹内文件的数量。文件数量越多,该文件夹的被访问概率越大,所绑定密钥的使用热度越大。

3) 文件夹距离根目录的层数。文件夹距离根目录的层数越小,该文件夹的被访问概率越大,所绑定密钥的使用热度越大。

4) 文件夹的可访问人员列表。同等情况下,文件夹的公共权限越多,可访问人员越多,被访问频率越大;仅一人可访问的私人文件夹,被访问概率最小。

5) 文件夹对应服务的应用类型。服务类型不同,文件的访问规律不同。例如,热门视频等服务数据,长时间使用;新闻等服务的数据,则随时间推移不再被频繁访问。

(2) 用户访问对密钥访问的影响因素

1) 文件夹的访问次数。单位时间内,用户访问文件夹的次数越多,文件夹的热度、所对应的密钥热度越大。

2) 文件夹内在线连接人数。在线连接人数越多,所在文件夹的使用热度越大,所绑定密钥的使用热度越大。

3) 用户拥有的文件夹,对应密钥。同一密钥下,拥有文件夹越多的用户,对该密钥的访问越频繁。

4) 多用户访问请求。根据访问请求到达模型,可以计算出对文件夹使用的到达概率,从而得出密钥使用热度。

2.3 密钥缓存热度计算

密钥使用的影响要素很多,主要集中在文件系统和用户访问这两方面,本文方案选择两个方面中各一个最直接和最主要的要素,作为热度计算的内容,构建密钥使用热度的基本模型。① 文件夹的平均创建/修改时间。大部分数据,时间越早,访问概率越小;形式化为文件夹中最新修改时间距当前时间的时长 T_n ,以秒为单位。② 时长为 countPeriod 单位时间的滑动窗口中,当前计算周期 t_n 中,用户查询该密钥的次数 visitCount 。

其余影响因素的实施难度较大,留待以后的研究中进行验证和优化提升。

采用指数平滑法计算密钥热度,对于每一个密钥,当前时间窗口内的热度计算公式如下。

$$\text{score}_n = \alpha \frac{\text{visitCount}}{\text{countPeriod} \cdot T_n} + (1 - \alpha) \text{score}_{n-1}, 0 < \alpha < 1$$

其中, countPeriod 即热度计算周期, visitCount 指当前热度计算周期中,该索引集合被访问的次数, T_n 即最新修改时间距当前时间的时长。历史热度 score_{n-1} 则反映集合累积的历史热度。参数 α 是衰减系数,用来确定当前周期累积的热度和历史热度在 score_n 中各自所占的权重。密钥的历史热度在本计算周期内以系数 $1 - \alpha$ 的速率衰减,经过多次迭代衰减,早期的累积热度对密钥热度的影响不断降低。

当 score_n 小于热度阈值 ε 时,说明该密钥的热度已较低,逐渐成为“冷数据”,无须放置在缓存中,可以被移出缓存。

2.4 密钥缓存热度存储与排序

在置换模块中,采用哈希表与小顶堆链表组合的方式,维护在用密钥的热度。若密钥缓存的总容量为 k ,采用哈希表存储密钥热度和统计值,采用小顶堆实现 Top-K 算法,对最多 $2k$ 个密钥的热度排名。具体而言,密钥热度哈希表的键值对中,键为密钥名,值为密钥热度 score ,当前时间窗口的访问次数 count ,经过时间窗口的个数 win ,文件创建时间 T_0 ,时间窗口进入下一周期时,迭代计算每个密钥的热度 score ,调整热度在小顶堆中的位置,并将 count 清零。小顶堆将热度低的密钥放置在顶端,便于在缓存达到阈值时,先将热度低的密钥换出缓存。未经过一个完整时间窗口的密钥,不计入小顶堆中,以避免换出刚刚换入的密钥。密钥热度的存储与排序示例如图3所示。

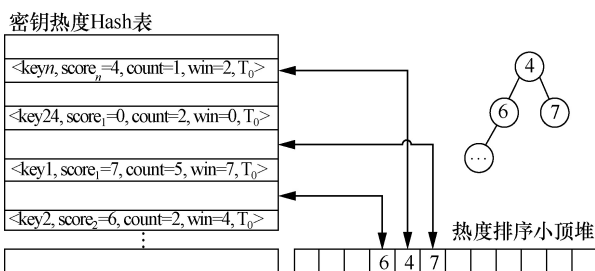


图3 密钥热度存储与排序示意
Figure 3 Key heat storage and storing diagram

热度计算与排序的变动和触发共 3 种方式：① 缓存新增密钥命中时，更新密钥热度哈希表中的次数统计；② 一个时间窗口结束时，更新密钥热度哈希表中的热度值，并调整小顶堆中的位置；同时，换出热度低于预期的密钥，并更新哈希表和小顶堆；③ 缓存中的密钥达到空间阈值时，需要换出密钥以保证缓存空间可用性，根据小顶堆的排序，换出缓存中热度最低的密钥，并更新哈希表和小顶堆。

(1) 缓存新增密钥命中

出现密钥查询和命中时，只是向密钥热度哈希表中增加一个查询次数统计，不会直接更新热度。若查询的密钥已在缓存中，在密钥热度哈希表中找到该密钥，并增加查询次数；若查询的密钥未在缓存和密钥热度哈希表中，需在密钥热度哈希表中创建该密钥的热度记录，再增加查询次数，具体过程如算法 1 所示。

算法 1 缓存新增密钥命中

Input:

- (a)hitKey: 当前命中的密钥
- (b)HashMap: 密钥热度哈希表

Output:

- (a)status: 执行是否成功

```

1) if(hitKey is not in HashMap) do
2)   HashMap.add(hitKey)
3) end if
4) hitKey.count++
5) return success

```

(2) 时间窗口结束

每个时间窗口结束时，触发热度计算和排序更新，依次更新密钥热度哈希表中每个密钥的热度值，并根据新的热度值调整小顶堆的排序。在更新时出现的缓存查询和命中，只需依次在统计中增加计数，由于计数是原子操作，增加计数和热度更新都无须上锁。

完成更新后，查看堆顶的密钥热度，若连续 t 个周期没有访问计数，且小于设定的热度阈值 ε ，则换出堆顶的密钥，并从密钥热度哈希表和小顶堆中删除该密钥记录，具体过程如算法 2 所示。

算法 2 时间窗口结束

Input:

- (a)HashMap: 密钥热度哈希表
- (b)Stack: 密钥热度小顶堆

(c)Cache: 密钥缓存

Output:

- (a)status: 执行是否成功

```

1) for keys in HashMap do
2)   if key.count == 0 do
3)     HashMap.remove(key)
4)     Stack.remove(key)
5)     Cache.remove(key)
6)   endif
7)   key.score =  $\alpha * \text{key.count} / t_n / t_0 + (1-\alpha)$ 
* key.score
8)   key.count = 0
9)   key.win++
10)  if key.score <  $\varepsilon$  do
11)    HashMap.remove(key)
12)    Stack.remove(key)
13)    Cache.remove(key)
14)  endif
15)  Stack.sort(key)
16) endfor
17) return success

```

(3) 缓存空间已满

置换模块监测缓存的空间占用情况，当缓存中的密钥达到空间阈值 φ 时，需要换出密钥以保证缓存空间可用性。根据小顶堆的排序，换出缓存中热度最低且上一时间窗口中访问计数为 0 的密钥，将其移出小顶堆和哈希表，具体过程如算法 3 所示。

算法 3 缓存空间已满

Input:

- (a)HashMap: 密钥热度哈希表
- (b)Stack: 密钥热度小顶堆
- (c)Cache: 密钥缓存

Output:

- (a)status: 执行是否成功

```

1) for keys in Stack do
2)   if (key.count == 0) or (key.score <  $\varepsilon$ ) do
3)     HashMap.remove(key)
4)     Stack.remove(key)
5)     Cache.remove(key)
6)   endif
7) endfor
8) return success

```

3 实验

本节内容的代码实现是基于文献[21]的加密区密钥管理方案,增加缓存管理模块,采用双向索引树结构组织缓存,对缓存的置换算法和预取算法进行实现,并通过HDFS的访问记录数据集,对关键算法进行测试。具体实验过程如下。

(1) 预取策略生成

在日志挖掘与预取策略生成方面,通过1 643 520条文件访问记录,对应加密区列表,聚类挖掘得出加密区密钥的使用规律,进而生成密钥预取策略。密钥的使用规律对应到24 h中,3类代表性密钥的周期性使用规律可视化结果如图4所示。

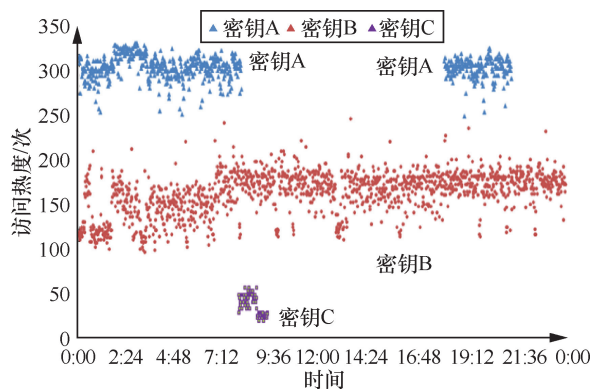


图4 密钥的周期性使用规律可视化结果
Figure 4 Visual result of periodic usage of keys

可以看出,密钥的使用规律包括3种情况。

1) 偶尔使用。如图4中的密钥C。大部分密钥不存在周期性使用规律,只是在偶尔用到该数据时,偶发查询。这类密钥无法形成周期性使用策略,或查询周期更长,难以预测。

2) 周期性间断使用。如图4中的密钥A。部分密钥的使用规律,呈现周期性变化,在每周的某些时段使用,因此可以在固定时间点进行预取,并在使用区间结束时确认并换出。

3) 不间断使用。如图4中的密钥B。少量密钥处于24 h使用的状态,这类密钥需持续驻留在缓存中,无须预取,除非缓存被更高频率的常驻密钥占满,否则不会换出。

根据密钥的使用规律,为周期性间断使用、可预测偶尔使用的密钥生成预取策略,结合时间控制器触发,即可实现密钥预取,提高命中率。

(2) 基于热度的置换算法

本节实现基于查询热度的置换算法,并与LRU置换算法、LFU置换算法进行同数据集的实验对比。数据集为10 000条密钥访问记录,共包括对4 160个密钥的多次访问。实验过程中,本节进行多次对比实验,将缓存的空间大小由100个密钥依次调整到600个密钥,每次增加100个,即6次实验的缓存空间中,密钥个数依次为{100, 200, 300, 400, 500, 600}。记录各算法下密钥访问的缓存命中率,如图5所示。在10 000次访问中,置换算法的缓存命中率均随着缓存空间增大而提升。

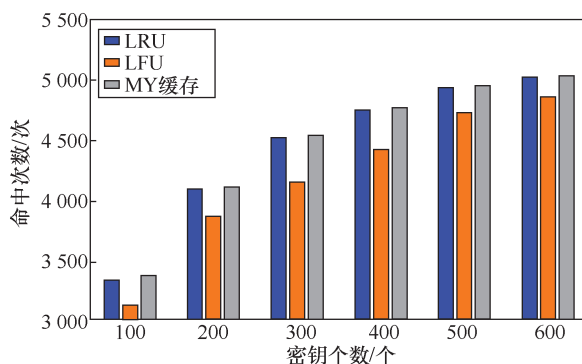


图5 各算法下密钥访问的缓存命中率
Figure 5 Cache hit ratio of key access under each algorithm

4 结束语

针对有限增长的内存容量不能包容无限增长的海量密钥索引问题,本文将密钥缓存细化为密钥索引数据结构、密钥置换算法两个方面考虑。具体地,密钥索引数据结构依然沿用文献[21]的双向索引树管理方式;从所绑定文件系统和用户的角度出发,综合考虑密钥的使用规律,基于热度识别设置密钥置换算法,动态更新缓存中的密钥。在降低内存占用的前提下,提升密钥查询效率,降低密钥文件I/O交互对查询性能的影响。实验表明,所提密钥置换可以有效提升缓存查询的命中率。

参考文献:

- [1] 第48次中国互联网络发展状况统计报告[R]. 中国互联网络信息中心. 2021.
The 48th statistical report on the development of China's internet[R]. China Internet Network Information Center. 2021.
- [2] SHIHABUL H M, PEDDERSEN J, JANAPSATYA A, et al. DEW: a fast level 1 cache simulation approach for embedded processors with FIFO replacement policy[C]//Proceedings of 2010 Design,

- Automation & Test in Europe Conference & Exhibition (DATE 2010). 2010: 496-501.
- [3] INOUE H. Multi-step LRU: SIMD-based cache replacement for lower overhead and higher precision[C]//Proceedings of 2021 IEEE International Conference on Big Data. 2022: 174-180.
- [4] ZHENG Q, YANG T, KAN Y Z, et al. On the analysis of cache invalidation with LRU replacement[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(3): 654-666.
- [5] MATANI D, SHAH K, MITRA A. An O(1) algorithm for implementing the LFU cache eviction scheme[J]. arXiv preprint arXiv:2110.11602, 2021.
- [6] LIU Y, ZHI T, XI H D, et al. A novel cache replacement scheme against cache pollution attack in content-centric networks[C]//Proceedings of 2019 IEEE/CIC International Conference on Communications in China (ICCC). 2019: 207-212.
- [7] HASSAN C A U, HAMMAD M, UDDIN M, et al. Optimizing the performance of data warehouse by query cache mechanism[J]. IEEE Access, 2022, 10: 13472-13480.
- [8] BALAJI V, CRAGO N, JALEEL A, et al. P-OPT: practical optimal cache replacement for graph analytics[C]//Proceedings of 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). 2021: 668-681.
- [9] CHEN H, XIAO Y, VRBSKY S V. An update-based step-wise optimal cache replacement for wireless data access[J]. Computer Networks, 2013, 57(1): 197-212.
- [10] PRIYA B K, KUMAR S, BEGUM S, et al. Enhancing the lifetime of STT-RAM with MRU replacement algorithm[C]//Proceedings of 4th International Conference on Recent Advances in Information Technology (RAIT). 2018: 1-6.
- [11] MEGIDDO N, MODHA D S. ARC: a self-tuning, low overhead replacement cache[C]//Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03). 2003: 9.
- [12] MEINT D, LIEBALD S. From FIFO to predictive cache replacement[J]. Network, 2019(25).
- [13] BUTT A R, GNIADY C, HU Y C. The performance impact of kernel prefetching on buffer cache replacement algorithms[C]//Proceedings of IEEE Transactions on Computers. 2007: 889-908.
- [14] OSMAN A M, OSMAN N I. A comparison of cache replacement algorithms for video services[J]. International Journal of Computer Science and Information Technology, 2018, 10(2): 95-111.
- [15] 刘恒, 谭良. 并行计算框架Spark中一种新的RDD分区权重缓存替换算法[J]. 小型微型计算机系统, 2018, 39(10): 2279-2284.
- LIU H, TAN L. New RDD partition weight cache replacement algorithm in spark[J]. Journal of Chinese Computer Systems, 2018, 39(10): 2279-2284.
- [16] LEVANDOSKI J J, LARSON P Å, STOICA R. Identifying hot and cold data in main-memory databases[C]//Proceedings of 2013 IEEE 29th International Conference on Data Engineering (ICDE). Piscataway: IEEE Press, 2013: 26-37.
- [17] 陈建, 沈满军, 姚一杨, 等. 基于密文策略属性基加密系统访问机制的缓存替换策略[J]. 计算机应用, 2017, 37(10): 2964-2967.
- CHEN J, SHEN X J, YAO Y Y, et al. Cache replacement strategy based on access mechanism of ciphertext policy attribute based encryption[J]. Journal of Computer Applications, 2017, 37(10): 2964-2967.
- [18] 葛微, 罗圣美, 周文辉, 等. HiBase: 一种基于分层式索引的高效HBase查询技术与系统[J]. 计算机学报, 2016, 39(1): 140-153.
- GE W, LUO S M, ZHOU W H, et al. HiBase: a hierarchical indexing mechanism and system for efficient HBase query[J]. Chinese Journal of Computers, 2016, 39(1): 140-153.
- [19] 王文楚. 基于相变存储器的平行架构混合内存性能优化研究[D]. 武汉: 华中科技大学, 2017.
- WANG W C. Improving the performance of parallel organization of hybrid memory system based on phase change memory[D]. Wuhan: Huazhong University of Science and Technology, 2017.
- [20] 李崇杰. 分布式文件系统缓存技术研究[D]. 南京: 南京大学, 2019.

LI C J. Research on cache techniques for distributed file system[D]. Nanjing: Nanjing University, 2019.

- [21] JIN W, GENG K, YU M J, et al. Efficiently managing large-scale keys in HDFS[C]//Proceedings of 2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/ DSS/SmartCity/DependSys). Piscataway: IEEE Press, 2022: 353-360.

[作者简介]



金伟 (1994-), 女, 北京人, 中国科学院信息工程研究所博士生, 主要研究方向为大数据访问控制与密钥管理。



李风华 (1966-), 男, 湖北浠水人, 博士, 中国科学院信息工程研究所研究员、博士生导师, 主要研究方向为网络与系统安全、信息保护、隐私计算。



周紫妍 (1998-), 女, 河北秦皇岛人, 中国科学院信息工程研究所博士生, 主要研究方向为访问控制。



孙喜洋 (1996-), 女, 黑龙江齐齐哈尔人, 中国科学院信息工程研究所硕士生, 主要研究方向为访问控制。



郭云川 (1977-), 男, 四川营山人, 博士, 中国科学院信息工程研究所正高级工程师、博士生导师, 主要研究方向为访问控制、形式化方法。