Journal of Computer Applications
计算机应用, 2020, 40(2): 321 - 327
ISSN 1001-9081
CODEN JYIIDU
2020-02-10
http://www.joca.cn

文章编号:1001-9081(2020)02-0321-07
DOI:10.11772/j.issn.1001-9081.2019091616

基于增量学习的RocksDB键值系统主动缓存机制

骆克云[1],叶保留[1*],唐　斌[1],梅　峰[2],卢文达[2]
(1. 计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023; 2. 国网浙江省电力有限公司,浙江 杭州 310007)
(*通信作者电子邮箱 yebl@nju.edu.cn)

摘　要:由于分层结构的约束,基于日志结构合并(LSM)树的RocksDB键值存储系统面临着读取性能低下的问题。一种有效的解决方法是对热点数据进行主动缓存,但其面临两个挑战:一是如何在数据分布持续动态变化时对热点数据进行预测,二是如何将主动缓存机制与RocksDB存储结构衔接起来。针对这些挑战,基于预测分析技术,构建了由数据采集、系统交互、系统测试等部分组成的面向RocksDB键值系统的主动缓存框架,能够将热点数据缓存在LSM树的较低层级中;并对数据访问模式进行建模,设计并实现了基于增量学习的热点数据预测分析方法,能够有效减少存储介质的I/O访问次数。实验结果表明该机制能有效提升RocksDB在不同动态工作负载下的数据读取性能。

关键词:RocksDB;主动缓存;增量学习;日志结构合并树
中图分类号:TP311　　**文献标志码**:A

Incremental learning based proactive caching mechanism for RocksDB key-value system

LUO Keyun[1], YE Baoliu[1*], TANG Bin[1], MEI Feng[2], LU Wenda[2]
(1. *State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing Jiangsu* 210023, *China*;
2. *State Grid Zhejiang Electric Power Company Limited, Hangzhou Zhejiang* 310007, *China*)

Abstract: RocksDB key-value storage system based on Log-Structured Merge (LSM) tree has the problem of low read performance caused by the constraints of its hierarchical structure. One effective solution is to cache hot spot data proactively, but it faces two challenges. One is how to predict the hot spot data when the data distribution keeps on changing constantly, the other is how to integrate the proactive caching mechanism with the RocksDB storage structure. To tackle these challenges, a proactive caching framework for RocksDB key-value system with multiple components including data collection, system interaction and system evaluation was built, which can cache the hot spot data at the low levels of the LSM tree. And with the modeling of data access patterns, an incremental learning based prediction analysis method for hot spot data was designed and implemented, which can reduce the number of I/O operations of storage medium. Experimental results show that the proposed mechanism can effectively improve the read performance of RocksDB under different dynamic workloads.

Key words: RocksDB; proactive caching; incremental learning; Log-Structured Merge (LSM) tree

0　引言

随着数据存储量的急剧增长,在跨区域的数据中心对海量数据的组织管理越来越困难。而随着分布式存储技术的日益完善,其设计思想不断启发着传统关系型数据库,出现了诸多以RocksDB[1]作为存储引擎的新型数据库系统。RocksDB是一种基于日志结构合并树(Log-Structured Merge Tree)[1]的键值对存储系统,它具有将随机I/O转化为顺序I/O的优点,可以大幅度提升数据写入时的性能,是当前绝大部分结构化大数据存储产品的首选存储引擎。

然而,RocksDB系统设计的场景是面向磁盘介质的批量写优化,通过分层结构实现批量操作,在查询数据时,读操作可能会在日志结构合并树的多个层级中发生I/O,导致不可避免的性能低下问题[2]。因此,如何高效地降低磁盘I/O访问次数,成为优化读性能研究中的一个关键问题。为了提升RocksDB中的查询效率,现有的系统设计大多从过滤器的角度出发,通过计算的方式减少不必要的I/O访问次数,如采用布隆过滤器(Bloom Filter)技术过滤那些数据一定不存在于磁盘上的请求[3],实现读取性能的提升。除此之外,现代计算机体系结构中的缓存机制也在读取性能优化中发挥着重要作用,通过尽可能地在内存或闪存中访问数据,减少磁盘寻道的时间,获得更好的查询性能。RocksDB中的缓存机制主要有内部精心设计的块缓存和操作系统的通用页缓存,适当提升

收稿日期:2019-07-31;**修回日期**:2019-09-24;**录用日期**:2019-09-29。
基金项目:国家重点研发计划项目(2018YFB1004704);国家自然科学基金资助项目(61832005);国家电网公司科技项目(52110418001M)。
作者简介:骆克云(1993—),男,安徽芜湖人,硕士研究生,主要研究方向:分布式存储; 叶保留(1976—),男,江苏如东人,教授,CCF杰出会员,主要研究方向:分布式计算、边缘计算; 唐斌(1986—),男,江苏东台人,助理研究员,博士,CCF会员,主要研究方向:分布式计算、编码理论;梅峰(1977—),男,浙江湖州人,高级工程师,硕士,主要研究方向:电力信息系统、大数据; 卢文达(1989—),男,吉林松原人,助理工程师,硕士,主要研究方向:数据挖掘、云计算。

缓存大小能极大地提升整体的读取性能。

从上面两种优化机制来看，提升过滤器和缓存效率的核心在于使用概率计算或高速设备的方式避免磁盘I/O访问，过滤器的位数越大，缓存的空间越多，效果越好。但这两方面的优化均需要大量的内存空间，通常内存价格比磁盘昂贵，所以并不能无限制地通过上述机制来提升查询效率。因此，RocksDB虽然具有显著的写优势，但这些存在的问题也制约了其进一步应用。为了实现写操作、空间占用和读取性能之间开销权衡，现有的工作集中于如何在特定的场景中达到最佳的配置。在理论指导部分，Dayan等[4]提出的Monkey系统对查询、更新和内存占用三者中的最优平衡点进行建模，对每一层的布隆过滤器根据数据的规模差异分配不同的存储空间，达到最小化所有层的布隆过滤器误报率，同时显著降低内存占用的效果。对于空间和性能的折中，Dayan等[5]的Dostoevsky系统进一步探索了不同的合并策略对不同操作的I/O和空间占用的影响，并使用了一种惰性合并的方法同时为不同层的布隆过滤器配置不同大小的内存，从而在不显著增加内存占用的情况下提升整体性能。

上述解决方案都是从结构设计的角度来解决RocksDB中的性能问题，此外也有一些工作从数据分布和访问模式的角度来解决这个问题。例如在Hadoop分布式文件系统（Hadoop Distributed File System，HDFS）[6]中，为了容错，通常使用多个副本，但多个副本会增加系统的存储开销，在增加并行度的同时也增加了空间占用，在热点读写的场景中，很多文件并没有发生I/O，因此无法体现多副本的优势。针对这种默认配置不灵活的问题，Bui等[7]提出了根据HDFS数据块的访问频率动态设置副本数加以解决。具体操作是使用高斯过程回归及其优化技术对文件访问块的可能性进行预测：对于热点块使用多个副本，通过数据并行加快处理速度；对于冷数据块则只用一个副本，减少空间占用，提升Hadoop系统的整体数据处理性能。

在数据库系统中，数据分布和访问模式对数据库性能的影响极大，例如在RocksDB键值系统中，顺序读的性能通常比随机读要高出一倍，因此对特定工作负载进行针对性优化可以获得可观的性能收益。然而，在RocksDB中进行热点数据主动缓存会面临两个挑战：一是系统在实际运行时，数据分布会持续动态变化，导致热点数据难以准确建模；二是如何将主动缓存机制与RocksDB存储结构对接起来，在尽可能不增添内存占用的情况下实现性能的提升[8-9]。

针对上述挑战，本文以拟合热点数据分布为思路，设计并实现了利用增量学习进行预测分析的热点数据主动缓存机制。具体而言，本文设计了由数据采集、系统交互、系统测试等部分组成的基于热点数据预测分析的主动缓存框架，用于与RocksDB存储引擎进行交互，使得热点数据大部分存在于日志结构合并树靠近内存等高速设备的层级中；并对数据访问模式进行建模，设计并实现了基于增量学习的热点数据预测分析方法，能够有效减少存储介质的I/O访问次数。实验结果表明该机制能有效提升不同动态工作负载下的数据读取性能。

1 RocksDB键值系统

1.1 RocksDB基本原理

RocksDB是日志结构合并树的一个经典实现，由Facebook在参考谷歌的LevelDB[10]和Apache的HBase[11]基础上开发而来，复用LevelDB的大部分代码，在此基础上进行工业级的代码优化和功能增强，在实际生产中已经应用在了数据库系统、分布式文件系统以及区块链系统中[9]。

日志结构合并树是一种按照逻辑分层的有序存储结构，保存在内存中的部件称为Memtable（C0），其数目和大小可以分别通过max_write_buffer_number（默认为2）、write_buffer_size（默认为64 MB）设置，数据先写入Memtable中，当大小达到上限后变为只读的Immutable Memtable，并使用新的Memtable继续写入，当总的Memtable数目达到上限时则触发刷盘操作，将Immutable Memtable数据直接写入磁盘；保存在磁盘上的部件称为SSTable，其中SSTable在逻辑上又分为$L_0(C_1)$层、$L_1(C_2)$层等。通常下一层的数据量是上一层的k（默认为10）倍，整体呈现出一个金字塔结构。具体结构示意图如图1所示。

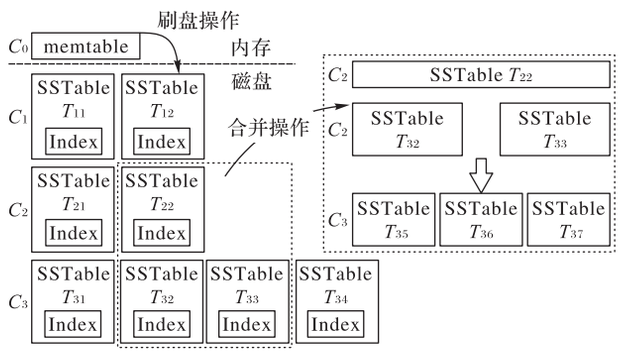

图1 日志结构合并树示意图

Fig. 1 Schematic diagram of log-structured merge tree

日志结构合并树在维护有序结构的过程中主要涉及到刷盘（Flush）和合并（Compaction）两大操作，其中刷盘操作在内存中的Memtable个数超过设定的阈值时自动触发，而当层级内的总数据量超过阈值时自动触发合并操作。其原理如图1所示，刷盘操作直接将Immutable Memtable写入到C_1层（以L标记为L_0层），形成多个SSTable文件。这些SSTable内部是有序的，但SSTable之间存在一定的范围重合。合并操作（Compaction）具体实施的方式与采用的合并策略有关，以默认的Level Style合并策略为例，选择C_2和C_3层中键有重叠的SStable进行合并，生成新的SSTable存放到C_3层中，C_1层以后的SSTable($L_1, L_2, \cdots$)每层的每个SSTable键区间是不重叠的，这与C_1层不同。

1.2 RocksDB数据读取流程

RocksDB中的读取操作根据数据的版本差异分为当前读和快照读[12]，默认为当前读，整体的读取步骤如下：

1）在事务对应的writebatch中查找是否存在请求的数据。

2）在Memtable中基于跳跃表快速查找数据是否存在。

3）在L_0层级中，依次遍历每个SSTable，若SSTable的索引和布隆过滤器数据未被加载到操作系统的页内存中，则首先进行SSTable句柄的缓存，然后通过布隆过滤器确定键是否

存在,若存在则检查数据是否存在于Block Cache,若不存在发生一次I/O,并缓存至Block Cache。

4)若L_0层中未查找到数据,则在L_1、L_2等层中接着查找。由于数据在这些层中是严格有序的,因此使用二分查找的技巧,根据键的范围定位SSTable,然后进行步骤3)中的查找流程。

从上面的流程来看,RocksDB读取数据时会在多个层级中发生I/O,极端情况下的I/O次数为$N(L_0)+N(L)-1$,即与L_0的文件数目、日志结构合并树的层级数之和相关。这种分层结构意味着若要提升读取性能,则必须通过一些机制避免访问磁盘I/O。现有系统中的优化机制主要有布隆过滤器和缓存两类,其中布隆过滤器方法可以快速判断键是否一定不存在于SSTable中,当访问不存在于数据库系统中的键时,能有效减少I/O次数。布隆过滤器在打开SSTable文件时通常被载入到操作系统的堆内存中,除非通过配置BlockBasedTableOptions:: cache_index_and_filter_blocks=true主动进行缓存,否则当关闭SSTable文件时,系统根据max_open_files参数所定义的文件句柄缓存数目决定是否对缓存数据进行垃圾回收。布隆过滤器的缓存位置有两处:OS Cache[13](Page Cache,页缓存,数据有压缩)和Block Cache[14](块缓存,数据无压缩),这两种缓存的详情如下:

OS Cache:普通的堆内存,在Linux系统中一般是Page Cache,对于每个文件的第一个读请求,系统进行预读,即会同步读取请求的页面及其后面的几个文件,对于第二次读取,如果所读页面不在Cache中,则继续同步读取,否则请求命中,扩大此次预读的范围。最新进入的数据存储在链表的头部,当内存不够时进行缓存的替换,从尾部扫描回收不活跃的内容。在RocksDB中,处于OS Cache的数据是压缩过的,因此比Block Cache节省空间,但代价是需要CPU的参与进行数据的解压。

Block Cache:它是RocksDB缓存数据至内存中以提高读取性能的一种方法,用户通过创建指定存储大小的Cache对象供数据库中同一个进程中的多个实例使用。Block Cache内部存储的数据通常是非压缩的数据块,也可以存储压缩的数据块。如果开启Direct IO参数,则没有OS Cache产生,此时数据以压缩的形式存在。在系统实现中根据缓存的替换算法来分主要有LRU Cache和Clock Cache两大类,这两种实现为了方便锁的控制都将Cache分片,用户设置的容量会平均分配至每个分片中,默认的Cache分片数为64,每块大小至少512 KB。Block Cache中存储的通常是数据块,索引和布隆过滤器缓存在block cache的外部堆内存中。这是因为当数据很大时,索引和布隆过滤器会占用大量的空间,以10 bit的布隆过滤器为例,其占用的空间大小是number_of_keys×10 bit,即对于一个256 MB的SSTable文件,索引和布隆过滤器分别需要额外占用0.5 MB和5 MB的空间,当数据规模较大时会占据Block Cache的大部分乃至全部空间,由于排挤效应导致严重的数据命中率缺失问题。

具体体现到RocksDB键值对存储系统中,在cache_index_and_filter_blocks设置为false的条件下,当max_open_files设置为-1时,默认会缓存所有数据至OS Cache中,这在一些内存空间有限的机器中特别容易造成内存不足(Out Of Memory, OOM)问题,解决的机制就是根据内存使用情况控制打开的文件数,及时清除缓存句柄,保持系统资源的合理占用。当cache_index_and_filter_blocks设置为true时,则布隆过滤器数据存储在block cache中,为了避免性能下降,通常的解决方案是设置cache_index_and_filter_blocks_with_high_priority的优先级,此时布隆过滤器和数据缓存分别位于block cache的两端,根据两者实际占用内存数动态移动。另一个优化点就是设置pin_l0_filter_and_index_blocks_in_cache来固定第0层的索引和布隆过滤器数据至block cache中,在内存占用和布隆过滤器命中率之间取得一个较好的平衡。

由上面介绍可知,过滤器机制和缓存机制[15]是相辅相成的,过滤器数据在SSTale创建的时候生成,在调用时缓存在内存中,而热点数据也需要缓存,因而内存空间是瓶颈。具体分析,过滤器机制可以直接减少I/O次数,在RocksDB中是必不可少的,因而优化的重点在于缓存机制。在查询数据时,通过缓存机制,一方面可以减少磁盘I/O,提高读取效率,在另一方面同时也增加了内存占用,这是一种典型的利用空间占用提升读取性能的折中思想。如果合理地使用缓存机制,在尽量不增加空间占用的情况下,实现RocksDB读取性能的提升,那么将具有优良的性价比。

2　热点数据主动缓存问题描述与解决思路

2.1　问题描述

本文将热点数据的主动缓存这一概念定义如下:在基于多个数据分布的查询工作负载中,通过某种机制将热点键值对主动缓存到内存或低层的SSTable中,同时求解一定时间内的热点键值对,从而提高整个工作负载的读取性能。

在不考虑数据分布的情况下,该问题解是一个典型的缓存控制问题。对于此问题,传统启发式算法的做法为先创建缓存空间,编写最近访问的键值对,在空间满时使用消除算法,如最近最少使用(Least Recently Used, LRU)算法,替换一些已存储数据。如果考虑数据分布背景,在概率密度分布为p的情况下,求解目标可以理解为对集合进行多次采样操作,求下一个周期内访问之前采样得到的数据的概率。

本文利用概率分布信息获取数据的访问特性,预测这些数据被再次访问的概率,然后筛选出概率较高的密钥,并进行主动缓存。这其中主要有两个难点:一是预测模型的选择,二是活动缓存机制的建立。下面将简要介绍这两点:

首先,对于预测模型选择问题,由于在通常情况下,数据访问的概率遵循80/20定律,即20%的数据占据了80%的访问频率[8],所以访问的这些数据具有明显的相关性特征。由此我们认为,热点数据预测的主要难点是数据范围会随着数据的不断插入和删除而动态变化,与此同时数据形式也呈现出复杂多样性。这种不确定性和不规则性意味着,在预测新数据时,很难直接使用预先训练好的模型进行实时在线预测。

第二个难点是有关主动缓存机制建立的问题,即活动缓存如何适应键值对存储结构。在键值对存储场景中,查询操作涉及到的数据量会非常大,如果使用特定的内存缓存结构(如Redis、Memcached),当空间设置得过小时,就会导致系统频繁的更新,从而不可避免地带来需要较大容量空间的问题,这样做就失去了优化的意义。因此,我们迫切需要一种新的缓存机制,在不占用大量额外空间的情况下实现与键值对存储系统的无缝对接。

2.2 解决思路

对于热点预测问题，考虑到在短期内整个系统环境是相对稳定的，所以在一个稳定的时间段里，可使用参数模型近似拟合这种方法。随着时间跨度增长，数据分布发生变化后，模型性能会不可避免地显著下降，此时就需要执行在线更新模型的操作，因此整个过程可以被视为一个增量学习[16]的过程。

与传统的批量学习技术相比，增量学习主要具有现有模型总结历史数据隐藏规律的优点，这就带来了无需显式保存历史数据、减少存储空间的好处。此外由于新的训练是基于历史数据中的隐含信息，所以能显著减少新数据的训练时间。就现有的增量学习实现技术而言，基于参数的模型（包括感知器和逻辑回归），通常可以很好地完成这一任务。该技术的核心是通过随机梯度下降优化算法实现参数的增量更新。定义整个模型的输入是一组特征，输出为该键是热点的概率，最后，根据输出概率对模型进行排序，选择概率较高的密钥对进行主动缓存，同时定期验证模型的准确性，当精度低于阈值时进行增量学习。

对于主动缓存，涉及到的机制有数据的重放操作（Splaying）和布隆过滤器等，一旦数据重放足够准确，布隆过滤器的使用就能自然减少，内存的使用也会随之减少，从而带来性能的提升。下面将详细介绍RocksDB特有的重放操作机制[17]：

日志结构合并树的一个特性是，顶部的数据总是比底部的数据更新，因此可以将频繁访问的数据“调度”到内存中，即重复写入，确保热点优先。在实践中，有很多方法可以确定一个键是否应该重播。在文献[17]中引入了两种重放策略，称之为AlwaysSplay和FlexSplay。

AlwaysSplay：这种策略的一个结果是，系统根本无法区分真实的频繁数据和热数据。如果键在很长一段时间内只有一次读取，则根本不需要写入，因此它不是最优的。这种方法带来的另一个问题是，系统可以创建同一个键的多个副本，从而造成不必要的空间浪费，降低查询性能。并且，当工作负载中的读写比发生更改时，这些热点将不会带来任何作用，造成更大的浪费。

FlexSplay：为了解决上述策略带来的问题，本文引入了弹性回放策略。此策略与原始的AlwaysSplay策略相同：put操作都是在get操作之后触发，但这些冗余的键值对被标记为重放过（通常将普通键分别标记为value、delete、merge、write、delete和modify）。在每次数据删除或合并期间，我们使用某种运行时策略来确定重新传递的数据是执行删除操作还是执行保留操作。所以，现在的问题变成了如何设计一种策略，以确保保留或删除副本的成本最小化、效益最大化和实现最简化。当频繁访问密钥集的比例超过阈值时，或者读操作的比例降低时，逐步删除重播数据，从而显示出自适应性。但该方法会涉及存储结构的修改，因此总体复杂度较高。

综合考虑上面两方面解决思路，本文提出的解决方案是：使用增量学习方法预测热点键，对这些热点键进行直接重放操作，同时依据热点键的分布区间动态调整max_open_files的大小，实现对内存占用量的精细控制等目标。

3 主动缓存框架设计

如图2所示，整个框架包括数据采集、系统交互和系统测试三大模块，以客户机-服务器的形式进行设计，同时还建立了热点预测模型。

下面将分别介绍每个模块的功能和设计情况。

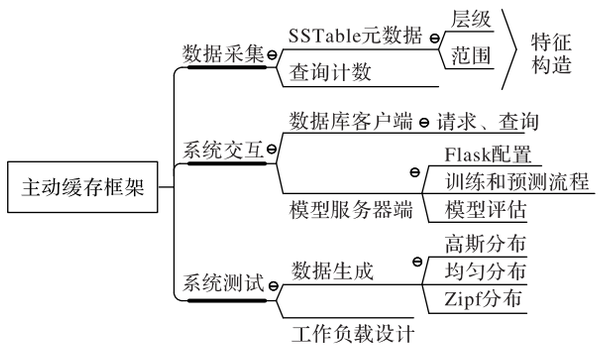

图2 基于预测分析的主动缓存框架

Fig. 2 Prediction and analysis based proactive caching framework

3.1 数据采集

在热点预测场景中，如果所有的数据都是间隔读取的，那么就会产生大量对模型没有影响的冗余键，这将极大地增加系统的负担。因此，在此模型中将直接忽略这些数据的读取行为，改为主要考虑以下数据：

1）频率统计：每个工作负载的主要记录是读操作下的键数。具体来说，每次执行get操作时，在访问命中时，都会记录操作的key和key的计数。具体数据格式为<key，count>。

2）分布统计：在每个工作负载启动前记录SSTable元数据信息，包括时间戳、层次结构、SSTable文件名、最小键、最大键等，数据格式为<timestamp，Level，SSTable，key_min，key_max>。

3）工作负载信息：记录每个工作负载数据信息，包括工作负载标签号、开始时间、结束时间、持续时间间隔、读取次数、总间隔、每秒查询、索引和布隆过滤器内存占用、Memtable内存占用和进程内存占用量等。

为了避免人为干扰，重放期间的读和写操作不被包括在统计数据中。

数据采集完毕后，将对数据特征进行进一步的构造。在这个步骤中，输入数据被处理成模型可以读取的形式。方法是将两种频率统计表和分布统计表的统计量进行汇总，并对其进行标记，构造正样本和负样本。以key为主键，构造特征目前包括：时间窗内的累计访问频率、SSTable的命中率、SSTable区间内key的访问比例、key的层次结构等。系统将上述特性拼接到一行记录中，并在一段时间后计算数据命中率；同时还会执行对标签标注的工作，例如再次访问的键被标记为1，否则被标记为0。

3.2 系统交互

该系统的通信体系结构为客户机-服务器模式。RocksDB数据库实例充当客户机，向模型服务器请求模型服务。这两个角色所提供的功能分别为：

1）RocksDB数据库实例：读写数据，向服务器发送任务请求，获取服务器端状态，获取热点键集合，通过max_open_files设置打开的SSTable文件数。

2）模型服务：提供模型服务，包括加载/培训模型、评估结果、热键预测、提供max_open_files设置等。

该模型被设计为一个带有Thread类线程的类ModelServer。它作为守护线程在后台运行，同时整个系统的

设计风格是Restful的。flask-Restful轻量级框架用于封装模型服务器的功能。其中涉及到的路由如表1所示。

RocksDB数据库实例与模型服务器的交互模式如图3所示。

表1　主动缓存服务器端Rest路由

Tab. 1　Rest routing on server side with proactive caching

URL	请求类型	参数/返回值	说明
/state	Get	返回状态	客户端请求系统状态
/keys	Get	返回键集合	客户端请求重放的键集合
/files	Get	返回max_open_files	客户端请求打开的文件数设置
/task	Post	发送任务请求	客户端发送训练或预测请求

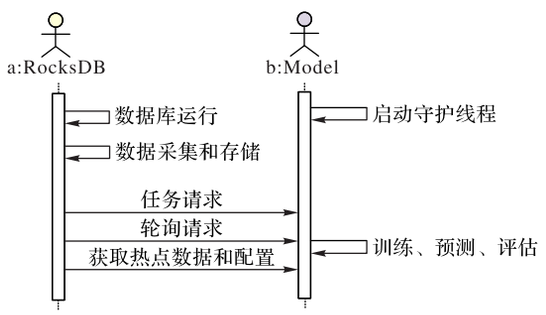

图3　主动缓存系统时序图

Fig. 3　Sequence diagram of proactive caching system

3.3　系统测试

该模块负责模拟数据工作负载的生成和组成操作。由于生成的数据需要反映数据的分布特征,所以主要考虑在文件访问模式下几种典型的分布[18]:

1)正态分布:此分布由均值和标准差决定。概率密度曲线以均值为中线,形状由标准差决定。标准差越小,曲线越接近均值。该系统计算给定数据区间的均值,并根据三标准差原理确定标准差。

2)均匀分布:该分布由最小值和最大值决定,概率密度是固定的。为了简化系统测试,将直接选取具有特定后缀的数据进行采样分析。

3)Zipf分布:在Zipf分布$P(r)=C/r^{\alpha}$中,将事件发生概率与其频率排序优先级之间的关系视为反比,其中:C为常数,r为根据发生频率排序,α为参数。在系统测试中,首先生成一个固定容量的累积概率数组,然后根据生成的随机数遍历数据,并取大于或等于随机数的第一个数组下标采样。

为了便于比较测试结果,需要将上述数据保存到一个文件中,以便工作负载读取。对于复合工作负载,主要从上述三种分布中抽取重复样本,形成复杂的工作流加以测试。

4　算法设计和实验

4.1　算法实现流程

增量学习模型使用sklearn. linear_model. SGDClassifier[19]实现,利用joblib库[20]保存模型的二进制文件,这样做的好处是打开数据库时若模型存在则会自动导入,节省模型训练时间。如果数据库中不存在现有模型,则需要触发模型训练操作:数据库端主动发送请求,启动后台训练流程。训练之后,数据库客户机可以通过轮询获得模型的输出。

增量学习的一般过程可以理解为:流数据输入→特征处理→partial_fit拟合→评价→应用。以下将重点介绍模型实现过程中的几个细节:

1)评估:很明显评估模型的学习效率是个二分类问题,所以为了直接筛选出热点键,将使用F1值度量[21]的方式。F1标准会综合考虑精确率与召回率两方面因素。首先,定义TP代表预测结果为1,这也代表实际为1的数目;TN为预测为0,实际也为0的数目;FP为预测是1,实际是0的数目;FN为预测是0,实际是1的数目。涉及到的各种计算公式如下所示:

精确率:

$$Precision = TP/(TP + FP)$$

召回率:

$$Recall = TP/(TP + FN)$$

F1得分是精确率与召回率的几何平均:

$$F1 = 2*TP*FN/(TP + FN)$$

2)应用:预测结果主要应用于两个方面,一是筛选热键,二是设置max_open_files参数,两者具体的处理方式如下:对于热点键的筛选,先对预测概率进行排序,然后选取预测概率大于或等于0.5的作为热点键。随后计算这些热点键所覆盖的SSTable热点数,选取SSTabel覆盖率大于100的SSTable热点数作为SSTable热点数进行记录。对于max_open_files设置,会根据经验值公式Max(min(SSTable number,2×SSTable hotspot number),SSTable number/2)来设置,其中,SSTable number指SSTable文件数,SSTable hotspot number指SSTable热点文件数。除此之外,还可以通过控制max_open_files来达到控制内存使用的目的。

整个算法系统的流程如图4所示,接下来将对图中的部分操作细节进行简要介绍:

1)训练模型:获取前一时间段各键的统计特征,得到当前命中的样本X和y_true,然后将数据放入模型partial_fit(X,y_true)中拟合。

2)模型预测:获取当前各键统计特征X,调用函数predict(X)进行预测,得到预测结果y_pred。

3)模型评估:执行函数f1_score(y_true,y_pred),计算得到F1。

其中,是否重新训练的标准为F1的下降幅度,本文中将阈值设为10%,即如果F1的下降幅度低于此阈值就开始使用新的数据训练模型。

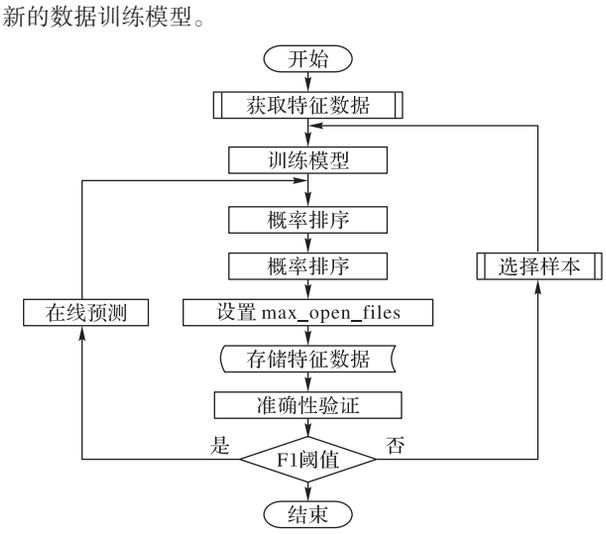

图4　主动缓存模型训练和预测流程

Fig. 4　Training and prediction process of proactive caching model

4.2 工作负载设计

本文的研究目标是面向读操作的工作负载优化，这其中最重要的部分就是工作负载设计。

我们认为的设计目标具体有两个：一个是模拟特定服务的读取性能，在此之上研究数据的缓存行为；另一种是通过改变数据分布来测试算法在动态环境中的适应性。为了验证本文设计是否符合这两方面的要求，设计了以下两阶段工作负载：

1）数据准备：启动 RocksDB，通过调用“rocksdb::NewLRUCache（4×1024*1024*1024LL）”配置 block cache 为 4 GB。键是长度为 8 的十六进制字符串，值是长度为 20 的十六进制字符串。第一阶段生成范围为 0x00000000~0x04ffffff 的键值对，数量设为 5 000 万。第二阶段生成范围为 0x00000000~0x04ffffff*2 的的键值对，数量也为 5 000 万，并且对于重复生成的键，使用第二阶段生成的去覆盖第一阶段生成的，可将此覆盖视为修改操作。在数据生成完毕后，还将构造符合以下条件的三个键集合用来执行查询操作：

① 数据分布 a：键的区间在 0x00000000~0x04ffffff，执行高斯采样，采样数量为 500 万；

② 数据分布 b：键的区间在 0x00000000~0x04ffffff，所有键均以 4 或 8 或 f 结尾；

③ 数据分布 c：构造近似 Zipf 分布模型（1.005，1 000 000），其中 rank 定义为 $(x + 100)/100$（减小数值过大的影响）。把区间 0x00000000~0x04ffffff 划分为 $n(n = 10)$ 份，在每份区间内选择一个数作为基数，再按上述类 Zipf 分布采样，采样数量为 100 万。

2）工作负载组合 1：写入第一阶段的键值对，按顺序依次以不同的数据分布 a（10 次）→b（10 次）→c（4 组数据，每组 5 次）执行，其中，键的值以时间为随机数种子随机采样 80%。这种设计的目的是多次运行，模拟商业周期和数据分布的流动。

3）工作负载组合 2：编写第二阶段的键值对，将数据分布的上限乘以 2，其余的操作与工作负载组合 1 相同。这是为了在更大的数据集中进行模拟，并测试模型的迁移能力。

实验总共测试 80 个工作负载，前 40 个数据范围大致相同。在随机插入或修改了一半数据之后，数据分布会发生显著变化，实验中将使用相同的工作负载继续测试这种动态适应性。

重复使用 10 次数据分布 a 是为了测试对于高斯分布数据，随机采样其中的一部分，重复一定次数后，可以模拟轻度热点读取模式。相应地，重复使用 10 次数据分布 b（键的区间在 0x00000000~0x04ffffff 的以 4 或 8 或 f 结尾的键）是为了验证对于均匀分布的数据，如果顺序采样其中的一部分，一方面会破坏前面已有的分布模型，另一方面会导致模型需要进一步的训练。最后，选取 4 组不同符合数据分布 c 的数据每组重复 5 遍，是为了证明：这里每组的数据分布长尾特征明显，每个读取为重度热点模式，但热点会经常变化。

实验记录的数据主要有两种类型：第一种是对系统状态执行监测的数据，格式为<工作负载类型，开始时间和结束时间，运行时间和数量的查询，每秒查询数，索引和布隆过滤器，内存 Memtable 内存，进程占用内存>；第二种是数据估算模型，格式为<时间，精确率，召回率，F1>。第一类数据记录在本地和调度模型中，使用它们来分析系统性能；第二种数据则被用于分析模型性能，它只存在于调度模型实验中。

4.3 实验性能分析

首先分析该模型的预测性能，该部分的性能由验证集上的测量标准 F1（Precision，Recall）来测量。除了前两个工作负载用作训练集和标注标签，其余工作负载皆用于训练和预测。

接下来使用前三个工作负载作为示例进行解释。首先模型以工作负荷 0 和 1 为训练集，进行模型训练。在 0 和 1 这两个阶段中预测的键值将会被用作热键，与工作负载 2 阶段中实际访问到的键值进行比较。其中，工作负载 2 中最上方的折线代表前两个工作负载中被访问到的键在工作负载 2 中再次出现的比例。将召回率定义为预测的热键数和访问的热键数与所有访问的键数之比，准确率定义为预测的热键数和访问的热键数与总预测的热键数之比。

如图 5 所示，这是其中一个组合的工作负载，可以看到精确率约为 0.83，召回率约为 0.62，F1 的值最大 0.72。如果在此情况下突然运行负载组合 b，因为 b 是均匀采样数据，所以，查全率和查准率自然会急剧下降，从而超过阈值触发训练，最终 F1 可以稳定在 0.58 左右。随后将运行工作负载组合 c，此时每个数据包数据都是分离的，所以需要一些时间去改变工作状态。如图 5 所示，模型在一段时间内相对稳定，这代表了模型处在学习访问模式，并在工作负载发生显著变化时进行重新训练。第二阶段的结果相似，这里不再重复。

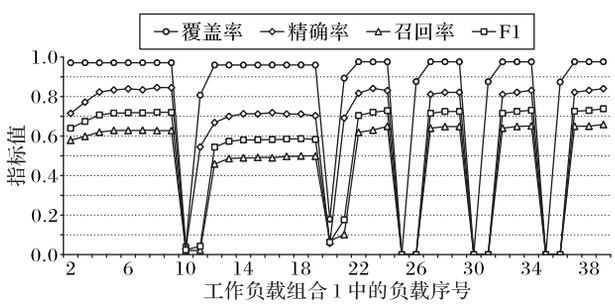

图 5 主动缓存模型性能折线图

Fig. 5 Line chart of performance of proactive caching model

在系统内存性能方面，通过跟踪进程占用的内存，可以发现原始数据库在其峰值时会使用 7.4 GB 内存，平均内存消耗则为 6.6 GB，活动缓存模型在其峰值使用 7.1 GB，不存在任何额外的内存使用。因此，模型本身带来的开销几乎可以忽略不计，可以认为系统在内存效率上有一定程度的提高。

就系统读取性能而言，查询测试时的吞吐量如图 6 所示：横轴表示运行的工作负载；左侧纵轴表示每秒查询的数量（QPS），单位为 kop/s；右侧纵轴表示模型 QPS 相对于原始 QPS 改进的百分比。

从图 6 可以发现，除了工作负载切换阶段吞吐量有所损失外，其余阶段查询吞吐量均提高，不同工作负载的组合带来的提升效果不尽相同，组合 a 查询性能平均提高了约 9.5%，组合 b 查询性能平均提高了约 6.3%，组合 c 查询性能平均提高了约 6.6%。总的来说，使用该模型能够将总体性能提高了 7.5%。因此，可以认为该模型具有一定的性能优势。

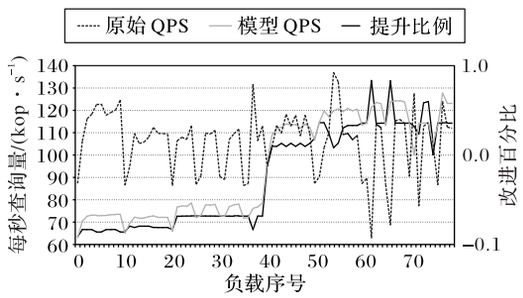

图6　主动缓存模型每秒查询数折线图

Fig. 6　Line chart of number of queries per second of proactive caching model

5　结语

本文将机器学习技术应用于RocksDB键值对存储系统中热点数据的主动缓存问题。整个系统不仅采用增量学习方法对热点数据进行预测,还建立了基于预测分析的主动缓存框架。实验结果表明,该方法能有效提高动态负载下的热点阅读性能。

以后,我们将针对热点预测模型的复用问题作进一步的研究,即如何将训练后的模型应用于其他不同键值分布的存储数据库问题。

参考文献(References)

[1] Facebook. A library that provides an embeddable, persistent key-value store for fast storage[EB/OL]. [2019-03-30]. https://github.com/facebook/rocksdb.

[2] O'NEIL P, CHENG E, GAWLICK D, et al. The Log-Structured Merge-tree (LSM-tree)[J]. Acta Informatica, 1996, 33(4): 351-385.

[3] BRODER A, MITZENMACHER M. Network applications of Bloom filters: a survey[J]. Internet Mathematics, 2004, 1(4): 485-509.

[4] DAYAN N, ATHANASSOULIS M, IDREOS S. Monkey: optimal navigable key-value store[C]// Proceedings of the 2017 ACM International Conference on Management of Data. New York: ACM, 2017: 79-94.

[5] DAYAN N, IDREOS S. Dostoevsky: better space-time trade-offs for LSM-tree based key-value stores via adaptive removal of superfluous merging[C]// Proceedings of the 2018 International Conference on Management of Data. New York: ACM, 2018: 505-520.

[6] SHVACHKO K, KUANG H, RADIA S, et al. The Hadoop distributed file system[C]// Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies. Piscataway: IEEE, 2010: 1-10.

[7] BUI D M, HUSSAIN S, HUH E N, et al. Adaptive replication management in HDFS based on supervised learning[J]. IEEE Transactions on Knowledge and Data Engineering, 2016, 28(6): 1369-1382.

[8] PARETO. Pareto principle[EB/OL]. [2019-02-22]. https://en.wikipedia.org/wiki/Pareto_principle.

[9] 骆克云. 基于机器学习的RocksDB存储引擎配置优化[D]. 南京:南京大学,2019:16-21. (LUO K Y. Configuration optimization of RocksDB storage engine based on machine learning[D]. Nanjing: Nanjing University, 2019: 16-21.)

[10] Google. LevelDB is a fast key-value storage library written at Google that provides an ordered mapping from string keys to string values[EB/OL]. [2019-03-30]. https://github.com/google/leveldb.

[11] GEORGE L. HBase: The Definitive Guide: Random Access to Your Planet-Size Data[M]. Sebastopol, CA: O'Reilly Media, 2011.

[12] GRUMMON J L, FRANKLIN C R. System and method for disk control with snapshot feature including read-write snapshot half: US 6341341[P]. 2002-01-22[2019-01-12].

[13] LIEDTKE J, HARTIG H, HOHMUTH M. OS-controlled cache predictability for real-time systems[C]// Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium. Piscataway: IEEE, 1997: 213-224.

[14] DONG S, CALLAGHAN M, GALANIS L, et al. Optimizing space amplification in RocksDB[EB/OL]. [2019-02-22]. http://cidrdb.org/cidr2017/papers/p82-dong-cidr17.pdf.

[15] 秦秀磊,张文博,魏峻,等. 云计算环境下分布式缓存技术的现状与挑战[J]. 软件学报,2013,24(1):50-66. (QIN X L, ZHANG W B, WEI J, et al. Progress and challenges of distributed caching techniques in cloud computing[J]. Journal of Software, 2013, 24(1):50-66.)

[16] 周志华. 机器学习[M]. 北京:清华大学出版社,2016:108-109. (ZHOU Z H. Machine Learning[M]. Beijing: Tsinghua University Press, 2016: 108-109.)

[17] LIVELY T, SCHROEDER L, MENDIZÁBAL C. Splaying log-structured merge-trees[C]// Proceedings of the 2018 International Conference on Management of Data. New York: ACM, 2018: 1839-1841.

[18] ROSS S M. 应用随机过程:概率模型导论[M]. 龚光鲁,译. 11版. 北京:人民邮电出版社,2016:39-49. (ROSS S M. Introduction to Probability Models[M]. GONG G L, translated. 11st ed. Beijing: Posts and Telecom Press, 2016: 39-49.)

[19] PEDREGOSA F, VAROQUAUX G, GRAMFORT A, et al. Scikit-learn: machine learning in Python[J]. Journal of Machine Learning Research, 2011, 12: 2825-2830.

[20] VAROQUAUX G, GRISEL O. Joblib: running python function as pipeline jobs[EB/OL]. [2019-02-22]. https://joblib.readthedocs.io/en/latest/.

[21] GOUTTE C, GAUSSIER E. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation[C]// Proceedings of the 2005 European Conference on Information Retrieval, LNCS 3408. Berlin: Springer, 2005: 345-359.

This work is partially supported by the National Key Research and Development Program of China (2018YFB1004704), the National Natural Science Foundation of China (61832005), the Science and Technology Project of State Grid Corporation of China (52110418001M).

LUO Keyun, born in 1993, M. S. candidate. His research interests include distributed storage.

YE Baoliu, born in 1976, Ph.D., professor. His research interests include distributed computing, edge computing.

TANG Bin, born in 1986, Ph. D., assistant research fellow. His research interests include distributed computing, coding theory.

MEI Feng, born in 1977, M. S., senior engineer. His research interests include power information system, big data.

LU Wenda, born in 1989, M. S., assistant engineer. His research interests include data mining, cloud computing.