面向多租户云数据库的自适应缓存管理

摘　要: 缓存系统是目前数据库、操作系统、分布式系统等多个领域中用以提升查询性能的有效手段。随着云计算技术的发展，面向多租户的云数据库系统成为了新的趋势。由于不同租户对云数据库的访问模式存在较大差别，如果直接使用传统的单一缓存策略，无法适应所有租户的业务特点。针对云数据库应用场景下多租户缓存管理的需求，本文研究了租户自适应的缓存管理机制，并提出了一种自适应缓存替换策略ARP（Adaptive Replacement Policy）。ARP 通过区分一个时间周期内的热页面与冷页面，在页面失效时优先置换冷页面。ARP 包含了一个可以调节的参数 p，其值介于 0 到缓存大小之间。为了解决 ARP 需要人工设置参数 p 的问题，论文进一步对 ARP 进行了优化，提出了 ARP+策略，它能够根据工作负载自动调整参数 p 的取值。论文在多个人工生成的数据集上进行了实验。结果表明，ARP 和 ARP+的性能均优于传统的 LRU、LFU、LIRS 和 LRU-2 算法。同时，ARP+能够自动地适应不断变化地访问模式，更适合多租户云数据库应用场景。

关键词: 云数据库; 多租户; 缓存管理; 替换算法; 访问模式

Adaptive Buffer Management for Multi-Tenant Cloud Databases

Abstract: The cache system is an effective means to improve query performance in many fields such as databases, operating systems, and distributed systems. With the development of cloud computing technology, multi-tenant cloud database systems have become a new trend. Because the access patterns of different tenants to the cloud database are quite different, if the traditional single cache strategy is directly used, it cannot adapt to the business characteristics of all tenants. In view of the multi-tenant cache management requirements in cloud database application scenarios, this paper studies the tenant-adaptive cache management mechanism, and proposes an adaptive cache replacement policy ARP (Adaptive Replacement Policy). ARP distinguishes between hot pages and cold pages within a time period and gives priority to replacing cold pages when a page miss occurs. ARP includes an adjustable parameter p, whose value is between 0 and the cache size. In order to solve the problem that ARP needs to manually set the parameter p, the paper further optimizes ARP and proposes the ARP+ strategy, which can automatically adjust the value of parameter p according to the workload. The paper conducts experiments on multiple artificially generated datasets. The results show that both ARP and ARP+ outperform the traditional LRU, LFU, LIRS and LRU-2 algorithms. At the same time, ARP+ can automatically adapt to changing access patterns, which is more suitable for multi-tenant cloud database application scenarios.

Key words: Cloud databases; Multi-Tenant; Buffer management; Replacement algorithms; Access patterns

1　引 言

近年来，，随着云计算技术的发展，基于多租户（Multi-Tenant）的内容分发网络（CDN，Content Delivery Network）等云服务模式成为新的趋势。在云服务模式下，云数据库平台可以为不同的用户提供个性化的数据访问支持。但是，由于不同用户的业务模式通常存在较大区别，而且访问的数据集也不同，因此，云数据库平台如果采用传统的单一缓存策略将无法满足所有租户的业务要求[1]。此外，单一的缓存策略还容易影响不同业务之间的隔离性，导致性能下降和数据一致性被破坏。

针对云数据库环境下多租户在数据访问模式上的不同特点，本文提出了一种面向多租户云数据库

的自适应缓存机制 ARP（Adaptive Replacement Policy），进而对其进行了优化，提出了增强版的 ARP 策略：ARP+。总体而言，本论文的主要工作和贡献总结如下：

(1)针对云数据库场景中多租户对数据访问的不同特点，提出了自适应缓存机制 ARP。ARP 每隔一段访问序列对所有页面进行一次排名，根据排名将页面分为热页面和冷页面。当出现页面失效时，优先选择冷页面进行替换。同时，在每一次重新计算排名之前，根据上一轮页面引用的命中与失效情况，对本次排名确定的热页面个数进行更新。

（2）ARP 包含了一个需要人工调整的参数 p，其值介于 0 到缓存大小之间。为了避免人工调参的代价，本文提出了一种 ARP 的改进策略：ARP+，它能够根据工作负载自动调整参数 p 的取值。

(3)我们在多个人工生成的数据集上开展了实验，并与多个已有的缓存替换算法进行了对比，包括 LRU、LFU 和 LIRS。结果表明，ARP 和 ARP+的性能均优于传统的 LRU、LFU 和 LIRS 算法。同时，ARP+能够自动地适应不断变化地访问模式，更适合多租户云数据库应用场景。

本文的后续内容安排如下：第 2 节介绍了经典的缓存替换算法并对其进行了分类，第 3 节介绍了本文提出的 ARP 与 ARP+算法，第 4 节给出了实验结果，最后第 5 节总结了全文并展望了未来研究方向。

2 相关工作

本节把经典的缓存替换算法分类进行具体阐述，总结其各自的特点以及适用场景。

2.1 基于时间局部性的缓存替换算法

LRU（Least Recently Used）以其简单并且容易实现而著称[15]。LRU 仅仅依靠最近访问时间信息决定替换缓存中的哪一个页面，在许多场景下都有优秀的表现。但是，对于一些不友好的负载，它的性能下降十分明显。LRU 在一些场景下表现不好的原因是它认为上一次访问时间最久的页面在将来也会等待最长的时间被访问。显然，它无法捕捉局部性较差的负载的访问模式。此外，对于特征不断变化的访问模式，它也缺乏相应有效的措施来适应。

CLOCK 算法是对 FIFO 算法的一种近似[12]。每个缓冲区页面都附加了一个使用位，同时有一个指针指示最近循环期间是否访问了该页面。当出现页面失效时，即将被替换的页面是通过逐步检查使用位来确定的[8, 15]。

2.2 基于频率的缓存替换算法

LFU（Least Frequently Used）策略根据页面的访问频率来选择将要替换的页面。相比于 LRU，LFU 的实现复杂度更高。在一些工作负载下，LFU 的性能是最好的 LFU 不区分最近的访问与很久之前的访问，所以对于局部性很强的工作负载，它的表现很差。同时，它无法很好地适应不断变化的访问模式。

FBR（Frequency-based Replacement）算法由 Robinson 和 Devarakonda 提出[2]。FBR 需要维护一个列表，该列表分为新的、中间的和旧的。他们在文章中声称 FBR 可以很好地利用维护引用计数的概念分解局部性。但是，FBR 包含许多可调整的参数，此外，缓存中的引用计数需要定期更新才能取得更好的表现。

LRU-K 算法有效解决了 LRU 的一些问题，同时也可以看作是 LFU 的近似[3]。在决定哪个页面进行替换时，它会考虑页面的倒数第 K 个引用做出决定。为了简单，作者建议将 K 设置为 2。LRU-2 在许多工作负载上已经取得了比 LRU 更加良好的性能。但是，LRU-2 也存在两个明显的缺陷：(1) 它

需要维护一个优先队列，这需要对数级别的复杂度。（2）它包含可以调整的参数 CRP（Correlated Reference Period)和 RIP(Retained Information Period)。目前无法将这些参数设置为一个定值，使得 LRU-2 可以在所有工作负载下都有不错的表现。

2.3 结合时间局部性和频率的缓存替换算法

在对 LRU 和 LFU 进行分析后，Lee 等人提出了 LRFU (Least Recently/Frequently Used) [5]。LRFU 方案将 LRU 与 LFU 进行结合，并且包含一个参数 λ。当λ接近 1 时，此时 LRFU 的表现与 LRU 十分接近。当λ接近 0 时，此时 LRFU 的表现与 LFU 十分接近。由此可见，LRFU 的性能在很大程度上取决于参数λ的设置。后来提出的 ALRFU 是对 LRFU 的一种改进[10]。

另一个值得关注的算法是 LIRS（Low Inter-reference Recency Set）[14]。LIRS 动态地将引用块分为两组：HIR(High Inter-reference Recency)块集和 LIR（Low Inter-reference Recency）块集。此外，LIRS 还定义了一种被称为“堆栈修剪”的操作。作者证明了 LIRS 在许多情况下比其他算法具有更好的表现，并且 LIRS 保持了 LRU 低开销的优点。作者建议将 Lhirs 设置为缓存大小的 1%，但是这对另外一些工作负载并不适用。参数 Lhirs 的设置对 LIRS 的性能有很大的影响。此外，“堆栈修剪”操作会操作缓存中的大量页面。2Q 算法[4]与 LIRS 类似也采用了两个队列，只不过采用的是 FIFO 队列和 LRU 队列。双队列的设计也同样被 ARC 等算法采用[6, 11]。AMG-Buffer 算法在双队列的基础上引入了元组缓存粒度，但它引入了额外的缓存管理代价，而且需要人工调整参数[13]。

3 面向多租户的自适应缓存替换策略

本节首先介绍面向多租户的自适应缓存替换策略 ARP，然后讨论 ARP 的改进版本 ARP+。

3.1 ARP 的设计思路

ARP 的工作流程如算法 1 所示。ARP 中包含了一个可以调节的参数 p，其值介于 0 到缓存大小 c 之间。ARP 每隔一段固定长度的访问序列周期，就会对所有页面进行一次排名。ARP 将这个固定长度设为缓存的大小 c，因为在最极端的情况下，在经过 c 个访问后，缓存中之前的所有页面可能都被替换，ARP 才能够捕捉到更有用的信息。我们将这个固定长度的访问序列周期称为排名周期。

算法 1 ARP

Input: 页面访问序列 x1, x2, ..., xt, ..., xT
Output: 无

```
1   for i = 1; i < T+1; i++ do
2       if hit then_
3           Hits ← Hits + 1
4       else
5           Faults ← Faults + 1
6           if this is the first page miss during this ranking period then
7               replace a page among all pages in the cache according to LRU
8           else then
9               replace a page among cold pages in the cache according to LRU
10          end if
11      end if
12      if i % c = 0 then
13          recompute the LRU-2 rank g and reference-time rank f during this ranking period
14          t ← ( t + f ) / 2
15          r ← ( 2 * r + g + t ) / 4
16          set the top p pages as hot pages and other page as cold pages according to r
17      end if
18  end for
```

每次排名周期的最终排名 r 是由上次排名周期的 r 排名、本次排名周期的 f 排名和 t 排名得到的（13~15 行)。ARP 将 r 排名的前 p 个页面设为下一个周期的热页面，而其他所有页面将在下一个周期中都被视为冷页面 (16 行)。在下一个排名周期中，

当出现页面失效时，如果这是本次周期内的第一次失效，则在缓存的所有页面中按照 LRU 原则进行替换（6~7 行），否则在缓存的冷页面中按照 LRU 原则进行替换（8~9 行）。ARP 这样做是为了提升算法的鲁棒性。如果在所有情况下，ARP 都在冷页面中选择页面进行替换，那么缓存中的所有热页面在本次排名周期内将一直驻留在缓存中。当出现一些极端情况时，ARP 的性能将会受到十分明显的影响。更加深入的原因将会在介绍 ARP+的时候进行详细阐述。

对于一个缓存大小固定的数据库系统，如果 p 的值越接近 c，则 ARP 认为之前的访问记录对本周期访问的参考作用越大，如果 p 的值越接近 0，则 ARP 认为之前的访问记录对本周期访问的参考作用越小。

3.2 ARP 的改进：ARP+

ARP 策略需要人工设定参数 p，这对于实际的云数据库应用而言增加了人工代价。因此，本小节考虑对 ARP 策略进行优化。

ARP 优化的主要目标是消除人工调参的代价，因此我们在 ARP 的基础上提出了根据工作负载的状态来自动调整参数 p 的策略，并将改进后的 ARP 策略命名为 ARP+。

ARP+的工作流程如算法 2 所示。对于一个固定的 p，ARP+与 ARP 有着完全相同的表现。但是，ARP+与 ARP 的不同之处在于，ARP+在整个工作负载上不使用一个单一的固定的 p 的值。

ARP+在每次对所有页面进行重新排名之前，会对 p 的值进行相应的更新（14~21 行）。值得注意的是，ARP+在增加 p 的值时，每次增加$\sqrt{c-p}$。p 的值越小，增量就越大。在减小 p 的值时，每次减小$\sqrt{p}$。p 的值越大，减量就越大。ARP+这样设置的目的是在 p 距离目标值差距很大时，能够减少更新的次数，从而使 p 的值尽快落到目标值附近。当然，如果每次的增量或者减量过大，则 p 很难逐渐逼近目标值，导致 ARP 会变得十分敏感，从而导致严重的性能下降。

算法 2 ARP+

Input: 页面访问序列 x1, x2, ..., xt, ..., xT
Output: 无
Initialization: $p \leftarrow c / 2$

```
1   for i = 1; i < T+1; i++ do
2       if hit then
3           Hits ← Hits + 1
4       else
5           Faults ← Faults + 1
6           if this is the first page miss during this ranking period then
7               replace a page among all pages in the cache according to LRU
8           else then
9               replace a page among cold pages in the cache according to LRU
10          end if
11      end if
12      if i % c = 0 then
13          recompute the number of hot pages in the cache m now
14          if w > m or w = p then
15              p = min( p + sqrt(c − p) , c − 1 )
16
17          else then
18              if w < m then
19                  p = max( p - sqrt(p) , 1 )
20              end if
21          end if
22          recompute the LRU-2 rank g and reference-time rank f during this ranking period
23          t ← ( t + f ) / 2
24          r ← ( 2 * r + g + t ) / 4
25          set the top p pages as hot pages and other page as cold pages according to r
26          recomputer the number of hot pages in the
27          cache w now
28      end if
29  end for
```

之所以在每个排名周期出现第一次页面失效时需要在缓存的所有页面中按照 LRU 原则进行替换，是因为 ARP+是根据 m 与 w 的大小关系来决定如何更新 p 的。因此，如果 ARP+在所有情况下，

都只在冷页面中选择页面进行替换，那么 w 将会一直大于或者等于 m，p 的值将永远不会减小。如果 ARP+让每次缓存中的热页面一直驻留在缓存中，那么当更小的 p 值更加适应此时的工作负载时，ARP+算法将无法捕捉到这个信息。在极端情况下，ARP+的命中率可能会受到很大的影响。

4 实验与分析

4.1 实验设置

本文使用 4 种不同的访问模式来模拟不同租户对缓存的需求，每种访问模式代表了一个租户。访问模式 1 是基于 Zipf 分布的[3, 4]。访问模式 2 是基于 Floratou 等人的的实验中使用的基于频率的工作负载[7]。类似地，访问模式 3 是基于 Floratou 等人的实验中使用的基于时间局部性的工作负载[7]。访问模式 4 是基于循环访问模式的[9]。

在数据集 1 中，我们首先生成了 400000 个访问模式 1 的引用、400000 个访问模式 2 的引用、400000 个访问模式 3 的引用，以及每隔 200000 个引用一次顺序扫描，这对应了访问模式 4。具体来说，在数据集 1 中，页面总个数 N 为 131072，每个租户能够访问的页面个数为 32768，并且互相没有交集。访问模式 1 的参数 a 和 b 为 0.8 和 0.2。访问模式 2 的参数 pf 为 0.4，coreSet 包含了 1024 个页面。访问模式 3 的参数 pr 为 0.6，stickiness window 包含了 1500 个页面。访问模式 4 的引用间隔在在其他三个访问模式的引用中，每隔 200000 个引用对 32768 个页面进行一次顺序扫描。

数据集 2、数据集 3、数据集 4 以及数据集 5 都是基于数据集 1 的。具体来说，数据集 2、数据集 3、数据集 4 分别将数据集 1 中访问模式 1、访问模式 2、访问模式 3 的引用个数扩大了 1 倍，数据集 5 每隔 100000 个引用进行一次顺序扫描。

在数据集 6 中，我们将引用的个数增加了一个数量级，并且每个租户能够访问的页面数量也不一样。具体来说，数据集 6 包含了访问模式 1、访问模式 2、访问模式 3 的引用各 2000000 个，此外，循环访问的周期仍然是 200000。在数据集 6 中，页面总个数仍然是 131072，访问模式 1 的租户和访问模式 2 的租户各能够访问 49152 个页面，访问模式 3 的租户和访问模式 4 的租户各能够访问 16384 个页面，并且 4 个租户访问的页面互相没有交集。访问模式 1 的参数 a 和 b 为 0.45 和 0.2。访问模式 2 的参数 pf 为 0.3，coreSet 包含了 4096 个页面。访问模式 3 的参数 pr 为 0.5，stickiness window 包含了 1000 个页面。访问模式 4 的引用间隔在在其他三个访问模式的引用中.

4.2 实验结果分析

ARP(q)表示参数 p 的值与缓存大小 c 的比例为 q 时的情形。此外，在 LIRS 的实现中，我们根据作者在文献中推荐的设置，将 Lhirs 设置为缓存大小的 1%进行实验。从图 1 我们可以看到，ARP(50%)在数据集 1 上有较好的性能，所以我们在数据集 1-5 上默认使用 ARP(50%)。

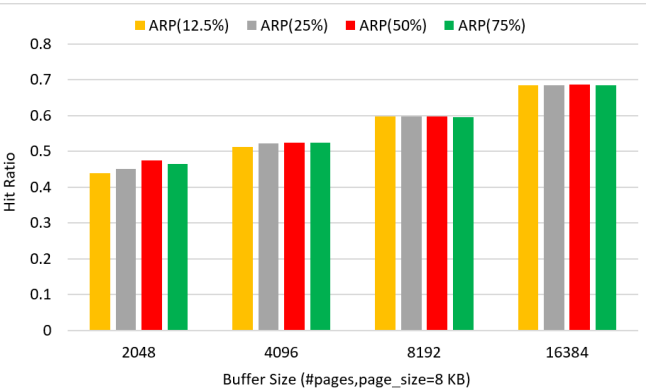

Fig.1 The hit ratio of ARP on Dataset1

图 1 ARP 在数据集 1 上的命中率

图 2 中比较了 ARP、ARP+与其他算法在数据集 1 上的命中率。随着缓存大小的变化，ARP+的命

中率曲线与 LRU 最接近。这是由于 ARP+在出现页面失效时，大部分情况下会选择缓存中的冷页面根据 LRU 原则进行替换，这本质上还是依据页面的最近访问时间信息。只不过 ARP+在每一个排名周期内，最多只会有一次在缓存中所有页面根据最近访问时间进行替换，而 LRU 算法会在所有情况下考虑缓存中所有页面的最近访问时间信息。缓存大小为 8192 时，LRU 与 ARP 和 ARP+的命中率几乎一样，这是因为当缓存大小足够大时，访问模式的特征已经相对不明显。针对 ARP，虽然我们可以选取 ARP 性能最好时的 p 值，但是这种做法并不具有普遍性。相比于 ARP，ARP+能够根据每个排名周期中的访问情况对 p 的值进行及时的更新，从而更好地适应访问模式的变化，而不是在整个过程中将 p 的值固定为某一常数。

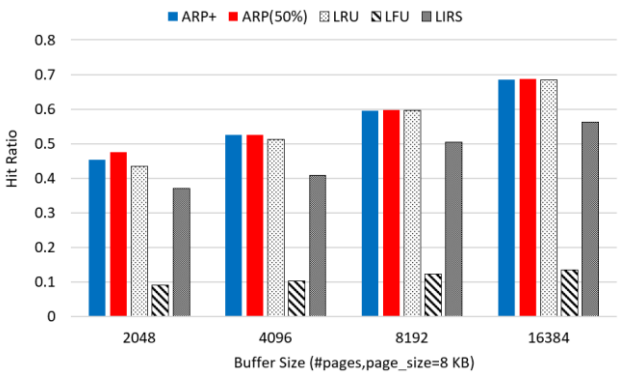

Fig.2 Comparison of hit ratios on Dataset1

图 2 在数据集 1 上的命中率对比

值得注意的是，LFU 在数据集 1 上的命中率出乎意料的低，并且随着缓存大小的增大，命中率只是略微地增加。我们认为这是由于 LFU 几乎不关注页面的最近访问历史，所以当一个页面在过去的访问频率很高，而最近没有被访问时，它将不会被替换。此外，LIRS 在所有缓存大小上的命中率都是低于 ARP 与 ARP+的。由于 LIRS 算法保证所有 LIR 页面都在缓存中，所以如何设定 Lhirs，将对 LIRS 算法的命中率有直接的影响。如果 LIRS 能够以在线的方式根据某种策略对 Lhirs 和 Llirs 的大小进行相应的调整，可能会对性能的提升有一定的帮助。

图 3 中展示了 LRU-2 在不同的 CRP 设置下在数据集 1 上的命中率与 ARP+的对比。LRU-2(k)表示 CRP 与 c 的比例为 k 的情况。在 LRU-2 的对比实验中，参数 CRP 同样对算法的命中率有很大的影响。举例来说，当缓存大小为 4096 时，LRU-2 算法在数据集 1 上的最高命中率是在将 CRP 设为缓存大小的 75%时得到的，而当缓存大小为 8192 时，将 CRP 设为缓存大小的 50%是最佳的选择。ARP+在绝大部分情况下的命中率都超过了 LRU-2，并且即使 LRU-2 设置了最优的参数，命中率最多也只比 ARP+高出 0.07%。Megiddo 等人在文献中也通过实验说明了，明智地选择参数对于实现 LRU-2 良好的性能至关重要[6]。如果以离线的方式为 LRU-2 设定相应的参数，那么成本将过高，不具有普适性。

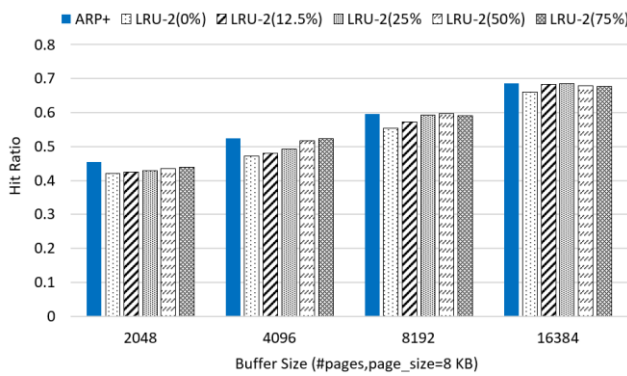

Fig.3 The hit ratio of ARP+ and LRU-2 on Dataset1

图 3 ARP+和 LRU-2 在数据集 1 上的命中率

图 4 显示了在数据集 4 上的命中率对比结果。ARP+仍然是性能相对稳定的替换算法。虽然在数据集 2 上，ARP 在某些缓存大小上，最优参数设置下的命中率都略微高于 ARP+，但是我们发现 ARP+命中率最多只差 1.27%。由于 ARP 需要预先设定参数并且不能修改，所以我们认为 1.27%是可以接受的差距。此外，ARP 与 ARP+仍然在所有缓存大小上超过了 LRU、LFU 和 LIRS。

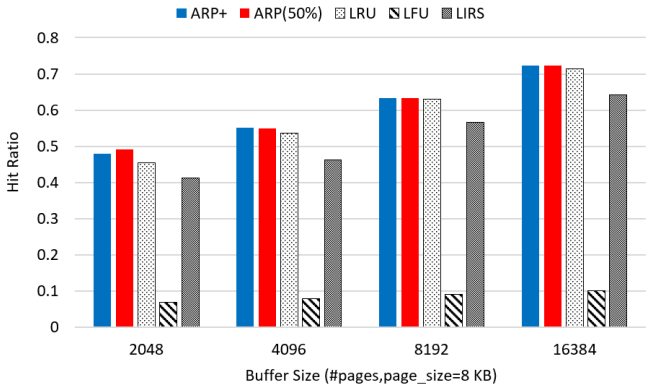

Fig.4 Comparison of hit ratios on Dataset2

图 4 在数据集 2 上的命中率对比

图 5 给出了数据集 3 上的实验结果。由于访问模式 1 和访问模式 2 都是基于频率的，所以虽然每个算法的命中率有所不同，但是能够得出的结论是类似的。

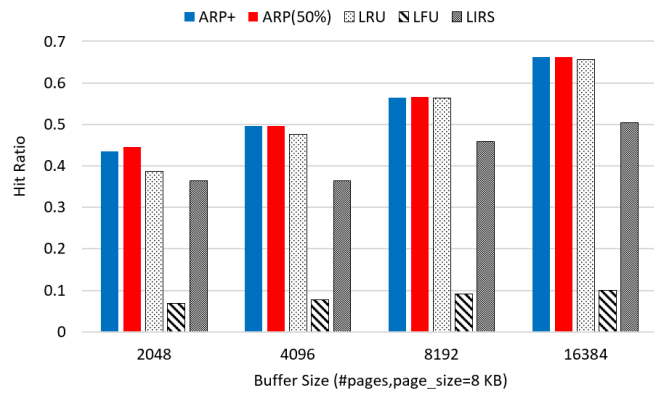

Fig.5 Comparison of hit ratios on Dataset3

图 5 在数据集 3 上的命中率对比

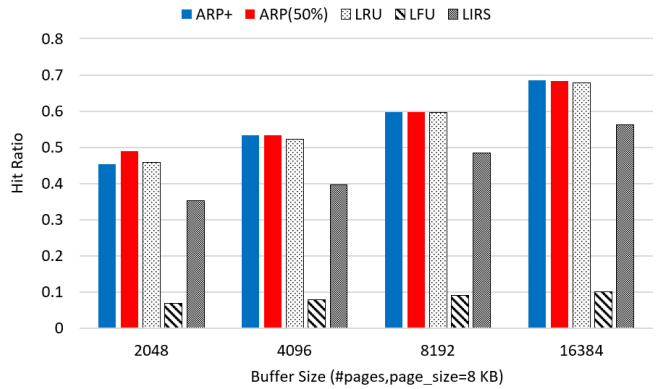

Fig.6 Comparison of hit ratios on Dataset4

图 6 在数据集 4 上的命中率对比

图 6 显示了数据集 4 上的命中率对比结果。虽然 LRU 的命中率仍然低于 ARP+，但是与数据集 1 相比，LRU 与 ARP+的差距明显缩小了，并且在缓存大小为 2048 时，LRU 的命中率甚至略微超过了 ARP+。我们认为这是由于访问模式 3 是基于最近访问时间的，所以 LRU 在数据集 4 上有更好的性能。

图 7 给出了数据集 5 上的实验结果。从图 7 中我们可以看出，与数据集 1 相比，所有算法的命中率都受到了一定程度的下降，包括 ARP 与 ARP+。但是，ARP 与 ARP+仍然在所有的缓存大小上的命中率超过了 LRU、LFU 和 LIRS。由此可见，ARP 与 ARP+在面对更加频繁的循环访问时，仍然能够保持良好的性能。因此，本文提出的 ARP 与 ARP+是能够抗扫描的。

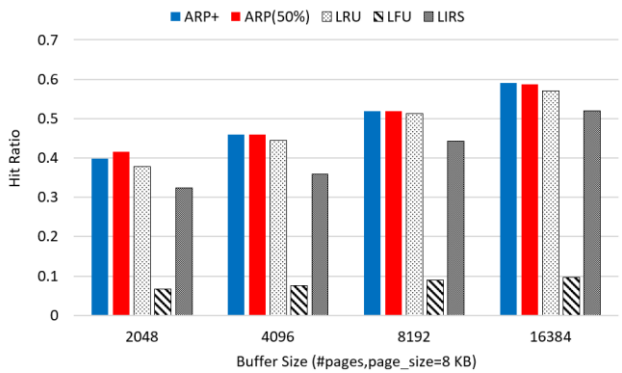

Fig.7 Comparison of hit ratios on Dataset5

图 7 在数据集 5 上的命中率对比

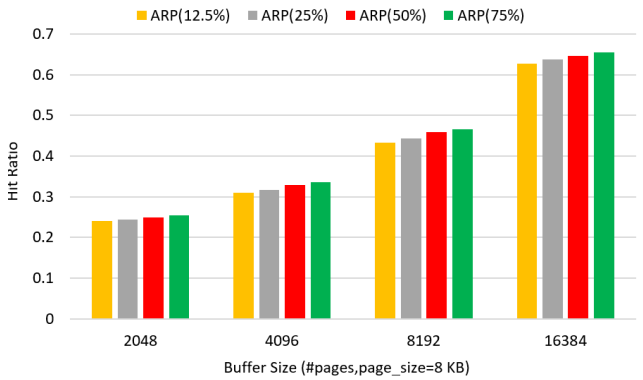

Fig.8 The hit ratio of ARP on Dataset6

图 8 ARP 在数据集 6 上的命中率

图 8 显示了 ARP 在数据集 6 上的命中率。从图 8 中我们可以发现，ARP 在数据集 6 上的最高命中率是在将 p 固定为 c 的 75%时取得的，这也说明，没有一个固定的 p 的值能够在所有数据集上取得最好的效果。

图 9 给出了 ARP、ARP+与其它算法在数据集 6 上的命中率对比。ARP 与 ARP+在数据集 6 上的命中率均高于其他算法。当引用数量增加时，ARP+有更多的机会来对 p 的值进行更新，这样就能够使 p 的值越来越接近目标值。而当某个访问模式的引用数量过小时，p 可能还没有达到目标值附近，目标值就随访问模式发生了变化。由此可见，当工作负载的规模突然增大，并且与工作负载特征相关的参数发生了改变的时候，ARP 与 ARP+仍然是值得信任的替换策略。

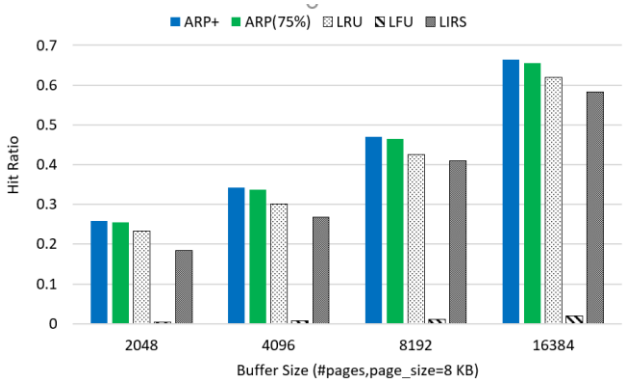

Fig.9 Comparison of hit ratios on Dataset5

图 9 在数据集 6 上的命中率对比

5 结束语

本文提出了新的面向多租户的云数据库缓存机制，包括 ARP 和 ARP+。实验结果表明，在面对访问模式不断变化的工作负载时，ARP 与 ARP+在绝大部分情况下命中率都超过了经典的缓存替换算法。与 ARP 相比，ARP+能够实时地对参数 p 的值进行更新，展现出了更好的自适应性。

References:

[1] Yang M, Guo K, Zhang Y. Cooperative Data Caching for Cloud Data Servers. EAI Endorsed Trans. Scalable Inf. Syst. 2016, 3(8): e3.

[2] Robinson J T, Devarakonda M V. Data cache management using frequency-based replacement [C], in SIGMETRICS. 1990: 134-142.

[3] O'neil E J, O'neil P E, Weikum G. The LRU-K page replacement algorithm for database disk buffering [J]. ACM SIGMOD Record, 1993, 22(2): 297-306.

[4] Shasha D, Johnson T. 2q: A low overhead high performance buffer management replacement algorithm [C] in VLDB. 1994: 439-450.

[5] Lee D, Choi J, Kim J H, et al. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies [J]. IEEE transactions on Computers, 2001, 50(12): 1352-1361.

[6] Megiddo N, Modha D S. ARC: A Self-Tuning, Low Overhead Replacement Cache [C], in FAST. 2003.

[7] Floratou A, Megiddo N, Potti N, et al. Adaptive caching algorithms for big data systems [J]. Computer Science, RJ10531 (ALM1509-001), 2015.

[8] Bansal S, Modha D S. Car: Clock with adaptive replacement [C], in FAST. 2004: 187-200.

[9] Choi J, Noh S H, Min S L, et al. Towards application/file-level characterization of block references: a case for fine-grained buffer management[C], in SIGMETRICS. 2000: 286-295.

[10] Lee D, Choi J, Kim J H, et al. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies [C], in SIGMETRICS. 1999: 134-143.

[11] Wang X, Chen K, Jin P. Adaptive Buffering Scheme for PCM/DRAM-Based Hybrid Memory Architecture. In NPC, 2021: 118-130

[12] Bach M J. The design of the UNIX operating system [M]. Englewood Cliffs: Prentice-Hall, 1986.

[13] Wang X, Jin P, Adaptive Multi-Grained Buffer Management for Database Systems. Future Internet, 2021, 13(12): 303

[14] Jiang S, Zhang X. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance [J]. ACM SIGMETRICS Performance Evaluation Review, 2002, 30(1): 31-42.

[15] Effelsberg W, Härder T. Principles of Database Buffer Management. ACM Trans. Database Syst. 9(4): 560-595 (1984)