Journal of Computer Applications
计算机应用,2018,38(5):1404-1409,1435
ISSN 1001-9081
CODEN JYIIDU
2018-05-10
http://www.joca.cn

文章编号:1001-9081(2018)05-1404-06
DOI:10.11772/j.issn.1001-9081.2017102861

多种存储环境下压缩数据库的缓存优化

张佳辰,刘晓光*,王　刚
(南开大学 计算机与控制工程学院,天津 300350)
(*通信作者电子邮箱 liuxg@nbjl.nankai.edu.cn)

摘　要:近年来,各行业数据量增速提升,对承担数据存储任务的数据库系统进行性能优化的需求也越来越强烈。利用关系型数据库I/O密集型、服务器CPU相对空闲的特点,在数据库中引入数据压缩技术,节省了数据存储空间和I/O传输带宽。但当今主流数据库系统的压缩技术都是针对传统的存储和运行环境设计,并未考虑固态硬盘(SSD)等新型存储设备和云数据库等虚拟化运行环境对系统性能的影响,因此,以数据库压缩系统在不同存储环境的缓存优化作为切入点,对系统整体性能的影响进行分析,给出了数据库压缩系统性能的分析模型,并以MySQL为例进行具体分析,给出了对应的缓存优化措施。在内核虚拟机(KVM)和MySQL数据库测试平台上的性能评估结果表明,所提出的优化方法使得系统性能最高有超过40%的提升,在某些配置下获得了优于物理机的性能。

关键词:数据库;数据压缩;虚拟化;磁盘I/O;固态硬盘
中图分类号:TP311.132.3　　**文献标志码**:A

Cache optimization for compressed databases in various storage environments

ZHANG Jiachen, LIU Xiaoguang*, WANG Gang
(*College of Computer and Control Engineering, Nankai University, Tianjin* 300350, *China*)

Abstract: In recent years, the amount of data in various industries grows rapidly, which results in the increasing of optimization demands in database storage system. Relational databases are I/O-intensive, take use of relatively free CPU time, data compression technology could save data storage space and I/O bandwidth. However, the compression features of current database systems were designed for traditional storage and computing environments, without considering the impact of virtualized environments or the use of Solid State Drive (SSD) on system performance. To optimize the cache performance of database compression system, a database compression system performance model was proposed, and the impact on the I/O performance of various system environments was analyzed. Take the open source database MySQL as an example, the corresponding cache optimization methods were given based on analysis. Evaluation results on Kernel-based Virtual Machine (KVM) and MySQL database show that the optimized version has an increase of more than 40% in performance under some configurations, even close to superior physical machine performance.

Key words: database; data compression; virtualization; disk I/O; Solid State Drive (SSD)

0　引言

各行业所产出的数据量增速提升,对数据存储和查询的需求急剧增加,对承担海量数据存取任务的数据库管理系统进行优化的需求也从未减小。在这种背景下,一般的关系型数据库系统中,磁盘I/O最为繁忙,当I/O成为性能瓶颈时,CPU占用率相对较低;同时,数据压缩技术中的压缩和解压缩操作则需要占用大量的CPU计算资源。因此,许多数据库存储系统中开始引入数据压缩技术,利用相对空闲的CPU计算资源,来减少存储空间和I/O传输带宽,具有很大意义。MySQL[1]、Oracle[2]、SQL Server[3]、DB2[4]等数据库都实现了各自的数据压缩功能来降低存储开销和提升I/O吞吐量。

近年来,更高性能的新型存储设备和虚拟化技术的快速普及,传统的数据库管理系统也开始面临新的局面。固态硬盘(Solid State Drive, SSD)等存取速度更快的存储设备作为加速缓存甚至整体代替传统磁盘(Hard Disk Drive, HDD)被用到数据库存储系统中,来进一步解决二级存储设备和主存储器之间的巨大速度差异。同时,随着云计算服务[5]的普及,越来越多的数据库开始从物理服务器迁移到云上的虚拟机中,甚至衍生出了数据库即服务(DataBase-as-a-Service, DBaaS)等以数据库为核心的云计算业务[6]。因此,本文以提升数据库压缩技术性能为目的,针对SSD和虚拟机等当前流行的数据库运行环境进行研究,主要贡献如下:

1)分析了(带压缩)数据库读写延迟的组成部分,逐一分析了影响系统性能的主要因素。

2)在应用数据库压缩技术的基础上,以MySQL数据库为例,对影响I/O性能的各种因素进行分析。

3)针对数据库压缩系统的不同部署环境,提出了优化数据库缓存来提升数据库I/O读性能的方法,并进行了实验验证。

为具体分析、优化和验证各种因素对数据库压缩系统I/O性能的影响,本文主要以MySQL这一被广泛使用的开源数据库为例进行分析和实验。在讨论虚拟化环境中的数据库压缩

收稿日期:2017-12-06;**修回日期**:2017-12-06;**录用日期**:2017-12-07。
作者简介:张佳辰(1994—),男,河北乐亭人,硕士研究生,主要研究方向:存储虚拟化;　刘晓光(1974—),男,河北安国人,教授,博士,CCF高级会员,主要研究方向:分布式系统、搜索引擎;　王刚(1974—),男,北京人,教授,博士,CCF会员,主要研究方向:云存储、搜索引擎。

时，本文主要使用以系统模拟器 QEMU(Quick Emulator)[7]作为虚拟机管理器(Virtual Machine Monitor, VMM)的内核虚拟机(Kernel-based Virtual Machine, KVM)[8]作为平台进行实验验证。

1　相关工作

虽然有损压缩算法的压缩率较高，可以节省更多存储空间[9]，但在数据库系统中，由于要保证数据的完整性，所以应采用无损压缩技术。Aghav[10]分析了不同类型的数据库系统中可能应用的多种无损压缩算法，指出在主流的关系型数据库中，用户记录是按行存储的，一般应采用基于字典的无损压缩算法。在一些主流的数据库系统中，MySQL InnoDB 存储引擎使用了基于 DEFLATE 字典无损压缩压缩算法的 zlib 库实现了表格式压缩功能[11]，Oracle[2]、SQL Server[3]、DB2[4]等数据库也都采用了改进的基于字典的无损压缩算法实现了各自的压缩存储功能。

压缩算法的压缩比和压缩性能之间是需要权衡的，压缩比越大的压缩算法，节省的空间越多，但同时压缩、解压缩操作的时间通常也会增大。马井玮等提出在 MySQL 压缩功能中用解压更快的 LZ4HC 算法[12]代替 zlib 的 DEFLATE 算法[13]，在磁盘空间节省变差 5% 的条件下将 SSD 上启用压缩功能的 MySQL 的读性能提升了 36%。虽然各种高性能的数据压缩算法被陆续地被应用到各种数据库存储系统中，但是对于不同的工作负载和系统运行环境，仍然没有一个绝对最优的压缩算法。

一般关系型数据库的逻辑结构分为数据库、表和记录等层次，而从存储结构角度则一般存在文件和块的概念。由于数据库系统中存在多个存储层次，所以数据压缩理论上也可以实现于数据表层次、数据块层次或用户记录层次等[10]。以 MySQL InnoDB 存储引擎为例，用户记录以文件形式进行存储，并定义了一种固定大小的页结构作为访存文件和管理缓冲池(Buffer Pool)缓存的基本单元。逻辑结构和存储结构之间通过用户行记录进行联系。用户行记录作为最小的逻辑单位被存储在页结构中[14]。由于 MySQL 自身定义的页结构默认为 16 KB，大小适中[15]，所以 MySQL 实现的压缩功能中，是以页结构作为压缩单元进行压缩和存储的[16]，本文也以此为基础进行进一步分析优化。

虽然上述很多工作都对数据库压缩进行过分析和改进，但还没有在考虑系统所运行环境的特点的情况下对缓存进行优化的。实际上，不论是存储设备类型的差异，还是云虚拟化环境与本地物理环境的差异，都会对系统的缓存效率、存储性能造成很大影响。特别是近年来，数据库有 HDD 逐步被 SSD 替代、本地数据库逐步向云端迁移的趋势，使这种差异引发的数据库性能的不稳定性越发明显。因此，本文针对这些不同存储环境的特点，对数据库压缩系统中的缓存部分进行分析，提出了针对 I/O 吞吐性能的缓存优化方法。

2　数据库压缩性能评价标准

2.1　存储效率

数据库压缩技术节省了数据存储空间，提高了存储空间利用效率。参照数据压缩比的概念，本文定义一个数据库压缩比 r 来描述数据库压缩后的存储空间节省情况，r 的值等于压缩前的数据文件原始大小 DS 和压缩后的数据文件大小 DS_{com} 之比。

$$r = DS/DS_{com} \tag{1}$$

数据库压缩比 r 的值越大，说明当前数据库系统压缩模块的空间节省效果越好。

2.2　I/O 性能

数据库系统的 I/O 吞吐量主要取决于每次读写请求的延迟。如图 1，在本文所讨论的数据库系统中，一次读或写所需的平均延迟时间 $T_{read/write}$ 可分为两部分：T_{trans} 表示所请求数据从数据库存储系统缓存区经过各个存储层次最终到达二级存储设备所需的总时间；$T_{process}$ 表示在数据库系统用户空间缓存中进行查找、更改、加密和压缩等操作所需的总时间。即：

$$T_{read/write} = T_{trans} + T_{process} \tag{2}$$

在 $T_{process}$ 中，数据库在自身缓存中对记录的查找和更改所需要的时间很短，只有加密解密或者压缩解压缩操作占用的时间和传输时间是可以比拟的。由于本文只涉及其中压缩性能的分析，所以式(2)中的 $T_{process}$ 可以替换为 T_{com} 或 T_{decom} 来分别表示一次写操作所需的压缩时间或一次读操作所需的解压缩时间。

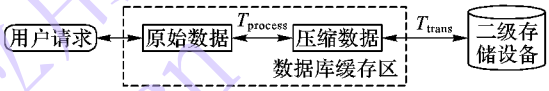

图 1　数据库压缩系统读写基本流程

Fig. 1　Read and write process in database compression system

另外由于数据库系统自身的缓存机制，部分数据在被请求时已经缓存在内存中了，数据库可以直接在缓存中进行操作，省去了从磁盘传输到内存的时间。本文将一次请求在数据库存储缓存区的命中率表示为 H，此时，所有的请求就只有 $(1-H)$ 的概率需要耗费 T_{trans} 时间。

因此，分别对应读操作 T_{read} 和写操作 T_{write}，可以将式(2)改写为：

$$T_{read} = T_{trans} \times (1-H) + T_{decom} \tag{3}$$

$$T_{write} = T_{trans} \times (1-H) + T_{com} \tag{4}$$

3　性能影响因素分析

3.1　解压缩时间 T_{decom} 和压缩时间 T_{com}

逐一分析式(3)和式(4)中的项，T_{decom} 和 T_{com} 主要取决于所使用的压缩算法的运行效率和 CPU 的运行速度。

要提高压缩算法的运行效率，可以针对数据库系统的数据特点对压缩算法进行专门的优化，如第 1 章所述，很多工作将重点集中于此。本文不重点讨论压缩算法的优化，而是在进行其他优化时采用了两种具有代表性的压缩算法进行分析：DEFLATE 算法具有较高的压缩比，但解压速度较慢；LZ4 算法解压速度很快，但压缩比相对较差。

要减小 T_{decom} 和 T_{com}，可以使用更高性能的 CPU。在虚拟化环境中，可以通过开启虚拟化专用指令加速来使虚拟机中 vCPU 的执行效率接近物理 CPU 性能，本文涉及虚拟化环境的所有测试，就是在使用了开启虚拟化指令加速的 KVM 虚拟机中进行测试的。

3.2　I/O 传输时间 T_{trans}

T_{trans} 主要由二级存储设备到物理内存的时间组成，也就是数据库一次读写磁盘的总延迟时间。HDD 和 SSD 之间的

性能差别巨大,HDD 顺序读写的性能远高于随机读写的性能,而 SSD 的随机读写和顺序读写性能的差距则不明显,并且 SSD 读写速度要远高于 HDD。选择不同的二级存储设备,会对式(3)和式(4)中的 T_{trans} 参数造成数量级的差异,因此,数据库存储系统的整体性能,在很大程度上取决于系统当前存储结构中二级存储设备的 I/O 性能。

当数据库存在多个连接同时处理 SQL 请求时,MySQL 会以多线程读写文件;数据库的请求操作取决于连接到数据库的用户,由于用户数量和请求的不确定性,一般数据库的随机读写远多于顺序读写,因此,不论是读还是写,SSD 的 T_{trans} 都要远小于 HDD。

在虚拟化环境中,由于多个虚拟机 Guest 的 I/O 是共享物理主机 Host 的,这就产生资源竞争问题[17],所以当 Host 存在多个运行有数据库系统的 Guest 时,每个数据库 I/O 吞吐量都会显著下降,并且由于各个 Guest 中数据库数据文件被用户访问的时间和空间的随机性,磁盘随机读写操作所占的比例将进一步增加,这将导致多 Guest 的 Host 的总 I/O 性能也会下降。另外,虚拟机的磁盘通常是以 Host 的文件模拟的,因此数据 I/O 的路径被加长,使得 T_{trans} 进一步增大;不同的虚拟机镜像文件格式也会为 Guest 的 I/O 性能带来不同程度的影响[18]。

3.3 缓存命中率 H

虽然缓存命中率 H 的精确计算涉及到数据库存储系统的具体实现,但其一定是与数据库缓存区大小 CS(Cache Size)和当前数据库数据总量大小 DS(Data Size)之比正相关的[16],可以表示为:

$$H \propto CS/DS \tag{5}$$

对于一定的数据量,较大的缓存区可以提高式(3)和式(4)中的命中率 H。足够的物理内存容量是数据库开辟足够大缓存区的必要条件,理想情况下,服务器的内存大于总数据量,此时可以期待所有数据最终都被缓存,命中率 H 将达到最大值 1,数据库性能达到最佳。但是在物理内存价格昂贵并且数据量巨大的情况下,让内存大小超过数据大小并不现实,缓存区只能缓存部分数据的情况在数据库系统的应用中广泛存在。因此,要提升数据库 I/O 性能,还需要根据具体工作负载,通过分析和测试来调整各种参数。

4 缓存优化

本章在式(3)的基础上,进一步分析和优化虚拟化环境中 MySQL 页级压缩的读性能。

开启压缩功能的数据库的缓存一般被分成两大部分:一部分存储未被压缩的数据,以提高对“热数据”的读写效率,避免过多的压缩解压缩操作;另一部分用于缓存已完成压缩的数据,来使得有限的缓存空间可以存储更多的数据,来提高缓存的命中率。

以 MySQL InnoDB 存储引擎为例,如第 1 章所述,压缩是以页结构为单位进行,原本 16 KB 大小的页被压缩为 1 KB、2 KB、4 KB、8 KB 等大小的压缩页,同时在 InnoDB 的 Buffer Pool 缓存中,压缩页和未压缩页都采用链表进行存储,并采用 LRU 算法进行管理。MySQL 倾向于将请求较多的页保存在未压缩页链表中,在 Buffer Pool 接近饱和的情况下,未压缩页占整个 Buffer Pool 大小的 10%,并且每个未压缩页一定对应一个压缩页;而占 Buffer Pool 大小 90% 的压缩页,则不一定能在 Buffer Pool 中找到与之对应的未压缩页,并且,存储在二级存储设备中的页都是压缩页,以节省存储空间。对于一般的修改,MySQL 也可以将修改信息直接写入压缩页中预留的 mlog 区域,减少压缩次数,进而减小页面压缩重构的代价。

4.1 优化模型

正是由于 InnoDB Buffer Pool 中同时存在压缩页和未压缩页,所以在进行读操作时,缓存命中率 H 可以表示为压缩页的缓存命中率 H_c 和未压缩页的缓存命中率 H_u 之和;并且以前所有页都会涉及的解压缩延迟 T_{decom} 在未压缩页命中时不再需要。另外,由于网络传输延迟不在本文讨论范围,本文将 T_{trans} 表示为 T_{disk_read},所以式(3)可以改写为:

$$T_{read} = T_{disk\_read} \times (1 - H_c - H_u) + T_{decom} \times (1 - H_u) \tag{6}$$

为寻求进一步优化 MySQL 数据库压缩的可能,本文将一些默认的或者原本固定的参数写为参数变量,来方便分析各参数值对性能影响:本文将压缩页占 Buffer Pool 的大小的固定比例 90% 设为变量 R,则未压缩页占 Buffer Pool 的大小为 $(1 - R)$;还将默认 8 KB 和 16 KB 的压缩和未压缩页的大小分别用 PS_{com} 和 PS 表示。为了方便表示,沿用了式(1)的 r,将其定义为原始数据大小 DS 和压缩后的大小 DS_{com} 之比;同时,将式(5)中出现的 CS 具体化为变量 BS(Buffer Pool Size)。假设命中率和 Buffer Pool 中页数与数据总页数的比成正比,H_c 和 H_u 可以分别表示为:

$$H_u = \frac{BS \times (1 - R)/PS}{DS_{com}/PS_{com}} \tag{7}$$

$$H_c = \frac{BS \times R/PS_{com} - BS \times (1 - R)/PS}{DS_{com}/PS_{com}} \tag{8}$$

为方便化简,设 $\alpha = PS_{com}/PS$,$\beta = BS/DS$;并将式(1)、式(7)和式(8)代入式(6)中,得:

$$T_{read} = T_{disk\_read} \times (1 - \beta r R) + T_{decom} \times [1 - \alpha\beta r(1 - R)] \tag{9}$$

在 β 或数据库压缩比 r 较大时,由式(9)所计算出的 T_{read} 可能小于 0,这在实际中是不可能出现的;且根据前文所述,Buffer Pool 中每个未压缩页必然有一个压缩页与之对应,所以,未压缩页一定比压缩页少,即式(9)存在一些限制条件:

$$\begin{cases} 1 - H_c - H_u > 0 \\ 1 - H_u > 0 \\ BS \times R/PS_{com} - BS \times (1 - R)/PS > 0 \end{cases} \Rightarrow$$

$$\begin{cases} 1 - \beta r R > 0 \\ \alpha\beta r(1 - R) > 0 \\ R/(1 - R) > \alpha \end{cases} \tag{10}$$

4.2 模型分析

式(9)中的参数中,与 β 参数有关的 BS 和 DS 取决于数据库系统具体配置和数据库存储数据的大小。有关 r 参数的 DS 和 DS_{com} 取决于所用的压缩算法和存储数据是否适合当前的压缩算法。存储设备和操作系统的实现决定了 PS_{com} 是 2 的幂时效率较高,即 1 KB、2 KB、4 KB 等。而 PS 值默认为 16 KB,所以 α 参数可以取 1/16、1/8、1/4、1/2 和 1 等值。观察式(9),要减小总的读延迟时间,在其他条件不变的情况下,α 应该尽量大。但本文并不对 α 参数进行详细分析和实验,因为 α 参数值的选择主要应该考虑数据库所存储数据的类型,若数据记

录较大或者所存储的数据压缩效果不好，采用的压缩页大小 PS_{com} 过小，在有数据写入时，将会出现频繁的压缩失败情况，导致大量的高代价页分裂操作；当数据记录较小或者数据适宜压缩的情况下使用较小的压缩页，能够适当地降低读写量，增加缓存中页的总数，带来一定的性能提升。对于不同的工作负载和读写请求比例，α 应该根据实际情况选择合适的值。另外，由于多数需要优化数据库性能的情况中 $\beta = BS/DS$ 通常远小于 1，所以调整 $[1 - \alpha\beta r(1 - R)]$ 项中的 α 对读性能的影响并不明显，所以本章后半部分的分析及实验采用默认的 PS_{com} 大小，即 8 KB，此时 α 大小为 1/2。

关于参数 R，根据式(10)可以得出：

$$\begin{cases} \dfrac{\alpha}{\alpha + 1} < R < \dfrac{1}{\beta r} \\ 0 < R < 1 \end{cases} \tag{11}$$

由于在本文需要优化的情况中，缓存大小 BS 通常远小于数据量 DS，即两者之比 $\beta \ll 1$，同时数据库压缩比 r 取值一般在 1 ~ 3，所以有 $1 / \beta r > 1$；又由于 $\alpha > 0$，所以 $\alpha / (\alpha + 1) > 0$，所以有：

$$\frac{\alpha}{\alpha + 1} < R < 1 \tag{12}$$

值得注意的是，虽然可以调整参数 R 到 $R < \alpha/(\alpha + 1)$ 的情况，但这样一定会导致性能下降，这是因为 MySQL 开启压缩后，每个未压缩页在内存中必然有与之对应的压缩页，所以如果分配给压缩页的缓存比例过小，将会使未压缩页无法完全占满剩余的缓存，导致缓存空间的浪费。所以若令 $R < \alpha / (\alpha + 1)$，在其他条件相同的情况下，性能不可能达到最优。

4.3　优化方案

首先，在式(9)中，当传输时间 T_{disk_read} 明显大于解压缩时间 T_{decom} 时，应该增大 R 的值；当 T_{disk_read} 明显小于 T_{decom} 时，应该减小 R 的值。

其次，如果将数据库系统建立在文件系统之上，会导致数据库系统和文件系统页高速缓存[19-20]重叠的“双缓存”问题；同样，在虚拟化环境中部署数据库系统时，也会带来 Guest 文件系统和 Host 文件系统两层页高速缓存重叠的“双缓存”问题。两种情况的“双缓存”问题都会导致不必要的内存浪费，所以应尽量绕过 Guest 和 Host 操作系统虚拟文件系统的页高速缓存，尽量消除缓存的重叠情况，来保证大部分内存都用于数据库缓存，进而保证整个系统性能的稳定。

因此，本文的缓存优化方案具体总结为如下两条：

1)在数据库系统中应用压缩时，避免双缓存问题。关闭操作系统中虚拟文件系统的页高速缓存，以提高数据库对内存的利用率。

2)对应不同的运行环境和压缩算法，选取合适的压缩数据占缓存的比例 R，以获得最高的吞吐量。

具体实现时，绕过数据库系统所在操作系统页高速缓存的方法并不复杂，只需要在配置文件 my. cnf 文件中将 innodb_flush_method 项设为 O_DIRECT。在虚拟化环境下，要绕过 Host 操作系统的页高速缓存，可以采用和 MySQL 绕过 Host 虚拟文件系统缓存类似的方式。比如，虚拟机管理器 QEMU 支持在开启虚拟机时加入一个后端存储文件控制参数 cache = none 来关闭对页高速缓存的使用。在本文讨论的虚拟化环境下运行的数据库压缩系统的情况中，避免“双缓存”问题后的 I/O 及缓存结构如图 2。

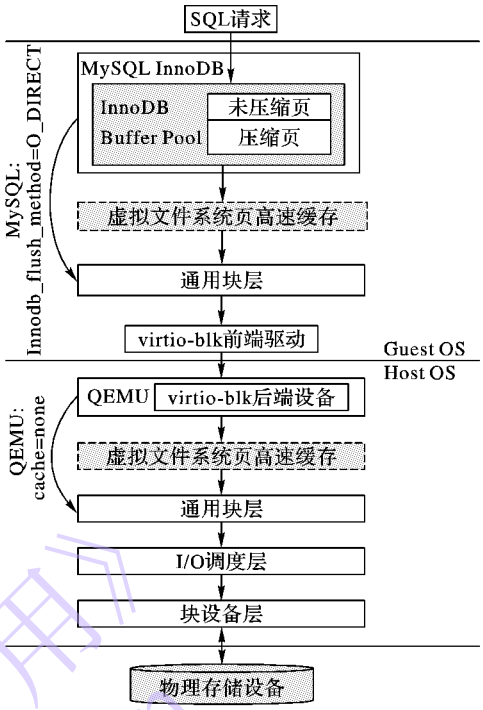

图 2　虚拟化环境中解决“双缓存”问题

Fig. 2　“Double caching” issue in virtualized environment

为了调整压缩数据在数据库缓存中所占比例 R，只需要修改数据库源码中对应的限制条件。例如，在 InnoDB Buffer Pool 的具体实现中，当启用页级压缩时，InnoDB 将额外增加一个称为 unzip_LRU 的 LRU 链表单独管理 Buffer Pool 未压缩页[21-23]，本章所述 R 值是在需要进行页面逐出时，通过选择从 LRU 链表还是 unzip_LRU 链表进行逐出来控制的，本文改变了这部分代码中的判断逻辑和所涉及的参数，进而实现了对 MySQL 源码中的 R 值的改变。

5　实验

本文采用 sysbench[24]作为测试工具，测试了磁盘 I/O 性能和数据库 OLTP 性能，同时使用了 KVM 虚拟机作为虚拟化环境对优化方案进行实验评估，具体实验的软硬件环境如表 1。

表 1　实验环境

Tab. 1　Experimental environment

项目	Host	Guest
CPU	2 × Intel Xeon E5 2609v4 8C 1.7 GHz	16 × QEMU vCPU
内存	128 GB	10 GB
SSD	DELL 480 GB	raw 格式镜像文件
HDD	WD 3TB 5400 rpm	raw 格式镜像文件
操作系统	CentOS 7	CentOS 7 Minimal
数据库	MySQL 5.5.37	MySQL 5.5.37 及其优化版本
VMM	QEMU 2.8.0	QEMU 2.8.0

5.1　HDD 和 SSD 性能差异

本文首先模拟实验验证了数据库压缩系统部署于不同存储设备时的性能差异：分别在 SSD 和 HDD 上生成 4 个总量为 4 GB 的文件，来模拟数据库的多表文件；采用了 1 KB、2 KB、4 KB、8 KB、16 KB 读写块大小，来模拟数据库的实际读写单

位;以 4 个线程进行随机读写测试,来模拟数据库的多链接请求情况;同时为了仅对比二级存储设备造成的性能差异,用 O_DIRECT 标志位打开文件,即关闭操作系统的页高速缓存,以消除系统缓存的影响。测试结果如表 2。

表 2 SSD 与 HDD 数据库读写速度 MB/s

Tab. 2 Database read/write speed on SSD or HDD MB/s

测试项目	读写块大小/KB				
	16	8	4	2	1
SSD 读	253.33	197.28	117.96	68.79	37.02
SSD 写	366.32	233.42	123.07	61.91	30.71
HDD 读	3.46	1.75	0.96	0.48	0.24
HDD 写	2.53	1.26	0.64	0.11	0.06

由测试结果可以看出,SSD 和 HDD 的读写速度差距悬殊,SSD 无论读还是写都明显优于 HDD。因此,如果在数据库存储系统部分或整体用 SSD 代替当前主流的 HDD 磁盘,就可以显著提升其 I/O 性能。此外,应用磁盘阵列(Redundant Array of Independent Disks, RAID)等并行存储技术,也可以提升存储系统的性能。但是,更高性能的二级存储设备往往带来更高额的存储成本[25],比如,在 2017 年 11 月,SSD 和 HDD 的单位 GB 存储成本相差 10 倍左右。因此,在数据库系统中引入数据压缩技术来节省存储成本很有意义。

5.2 虚拟化环境中的测试

按照 4.3 节中的第一条优化方法,本文绕过了 Host 操作系统和 guest 操作系统的页高速缓存。为了对比虚拟机和非虚拟机环境中的数据库压缩系统的性能,本文采用联机事务处理过程(OnLine Transaction Processing, OLTP)工作负载进行测试。图 3 中,本文分别在 SSD 和 HDD 上测试了不应用压缩(nocom)和应用压缩(LZ4HC)两种情况下的数据库的 OLTP 性能。测试结果的单位是每秒事务处理量(Transactions Per Second, TPS),数值越大,表明数据库系统的事务处理性能越好。

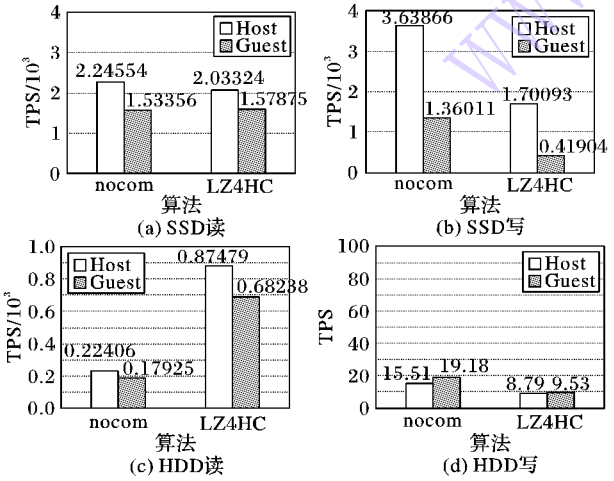

图 3 虚拟化与非虚拟化环境中数据库压缩性能对比

Fig. 3 Database compression performance in virtualized and non-virtualized environments

可以发现,在读性能方面,不论是 SSD 还是 HDD 平台,也不论是否应用数据库压缩,Guest 相对 Host 都有 20% 到 30% 的下降;对于写性能,在 SSD 上 Guest 会有一半以上的性能损失,在 HDD 上由于写速度慢,Guest 和 Host 的差别不大。由于 Guest 中的数据库读性能更加接近 Host,以读操作为主要工作负载的数据库系统,更适合部署于虚拟化环境。本文也因此继续验证了 4.3 节所提出的第二条优化方法在虚拟机中的读性能优化效果。

5.3 优化方法的性能评估

按照 4.3 节中的第二条优化方法,调整 MySQL 压缩表格式功能开启时的 R 值,让其在 0 ~ 1 变化。每次改变 R 值后,在 SSD 和 HDD 上分别进行 MySQL 的 OLTP 性能测试来对优化进行评估。

实验时 InnoDB Buffer Pool 大小为 128 MB,使用 HDD 进行实验时,生成的测试数据大小为 512 MB,使用 SSD 时,生成的测试数据量为 4 GB。被测试的压缩算法包括 DEFLATE 和 LZ4HC,并以未启用数据库压缩功能的情况作为基准进行比较,实验结果如图 4 所示。

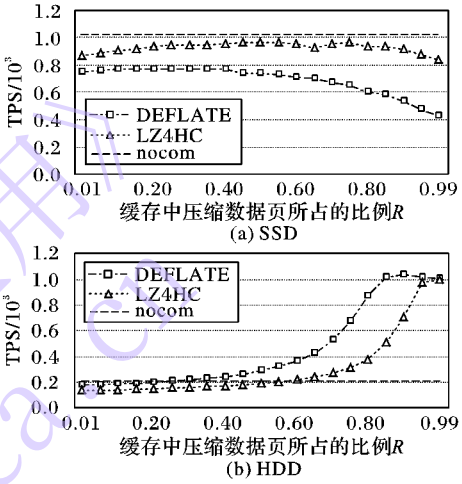

图 4 改变参数 R 后的数据库压缩系统 OLTP 性能

Fig. 4 Database compression performance after changing parameter R

如前文所讨论可知,HDD 性能明显差于 SSD,T_{disk_read} 更大;DEFLATE 解压速度明显差于 LZ4HC,T_{decom} 更大;并且对于所测试的系统,LZ4HC 解压缩速度可达上千 MB/s,HDD 随机读写数据的速度只有几 MB/s,故 LZ4HC 解压速度远远高于 HDD 随机读写速度。观察图 4(a)中的曲线,SSD 上应用 DEFLATE 压缩算法时,T_{disk_read} 明显小于 T_{decom},应尽量减小 R 值,且此时由式(6)和此时使用的 α 值 0.5,R 理论上最小取值为 0.33,实验中,R 值在 0.3 左右达到最好的性能,符合预期结论;HDD 上应用 LZ4HC 或 DEFLATE 压缩算法时,T_{disk_read} 明显大于 T_{decom},此时较大的 R 值会得到较好的性能,观察图 4(b)中的曲线,应用 DEFLATE 算法时,性能在取值大于 0.85 时最佳,而 LZ4HC 算法在最大取值 0.99 达到最佳,符合预期。

图 5 中分别列出了优化前后虚拟机 Guest 和宿主机 Host 中 MySQL 页级表压缩的 OLTP 性能。进行对比的三种情况均开启了数据库压缩功能,其中 Guest 代表优化前虚拟化环境下部署数据库的 OLTP 性能,Host 代表在非虚拟化环境且没有进行过缓存优化的情况下的数据库性能,Guest_Opt 代表在虚拟化环境下且应用了最佳缓存优化参数后的数据库的性能。

可以看出,在 SSD 上应用 DEFLATE 压缩算法时,优化后相对优化前的性能提升了 43.6%;SSD 上应用 LZ4HC 压缩算法时,性能提升了 5.2%。在图 5(b)中,HDD 上应用 LZ4HC 压缩算法时,性能提升了 41.3%。可见,MySQL 固定的 R 值 0.9 仅在 HDD 上应用 DEFLATE 算法的情况下效果较好,在使用

SSD 等更快的二级存储设备或使用 LZ4HC 等更快的解压缩速度的压缩算法,未优化版本均不能达到令人满意的性能。

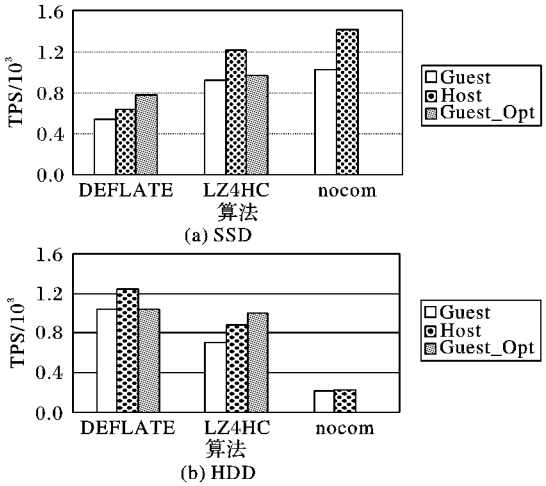

图 5　优化前后的 OLTP 性能对比

Fig. 5　OLTP performance before and after optimization

从实验结果中,还可看出虚拟机中数据库压缩优化后的最佳性能(Guest_Opt)与非虚拟化环境中未优化数据库压缩的 OLTP 性能(Host)。通过对比可以发现,在使用 SSD 存储的同时应用 DEFLATE 压缩算法,或者在使用 HDD 存储的同时应用 LZ4HC 压缩算法两种情况下,由于优化效果更为明显,Guest 数据库压缩性能从优化前的差于 Host 变为了优化后的超过 Host。其中,SSD + DEFLATE 情况的 Guest_Opt 相对 Host 性能提升了 21.3%,HDD + LZ4HC 情况的 Guest_Opt 相对 Host 性能提升了 13.72%。虽然这两种性能优化组合并非最佳,但在很多特定情况下仍具有很大意义,例如,SSD 的成本昂贵,DEFLATE 算法具有较高的压缩比,SSD + DEFLATE 的优化组合可以在控制存储成本的同时达到更好的性能。因此,在虚拟化环境中,本文提出的优化方案有很大的应用价值。

6　结语

本文首先阐述了在数据库系统中应用数据压缩技术的优势。然后给出了针对数据库压缩系统的分析模型,分析和实验验证了影响数据库压缩性能的因素。最后,在前人模型的基础上进行改进和分析,提出了一种在虚拟化环境中,通过优化缓存系统结构和使用结构来优化以读操作为主的数据库压缩性能的方法,使性能提升了 40% 以上。这种优化方法也使得在几种特定情况下,虚拟化环境中的数据库压缩性能达到并超过非虚拟化环境下的性能,具有很大意义。

本文也存在一些相关问题有待进一步研究。比如对数据库压缩的压缩算法的优化没有讨论;对虚拟化环境中数据库压缩系统的性能优化也仅限于虚拟机用户空间的数据库系统,以后的工作中还可以在虚拟机或宿主机操作系统层面开展进一步的分析优化工作。

参考文献(References)

[1] MySQL InnoDB table compression[EB/OL]. [2017-06-01]. https://dev.mysql.com/doc/refman/5.7/en/innodb-table-compression.html.

[2] Oracle advanced compression[EB/OL]. [2017-06-01]. http://www.oracle.com/technetwork/database/options/compression/index-095686.html.

[3] SQL Server page compression implementation[EB/OL]. [2017-06-01]. https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/page-compression-implementation.

[4] SANDERS R, FANGHAENEL T. Optimize storage with deep compression in DB2 10[EB/OL]. (2012-03-17)[2017-06-01]. https://www.ibm.com/developerworks/data/library/techarticle/dm-1205db210compression.

[5] ARMBRUST M, FOX A, GRIFFITH R, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4): 50-58.

[6] CURINO C A, JONES E PC, POPA R A, et al. Relational cloud: a database-as-a-service for the cloud[C]// Proceedings of the 5th Biennial Conference on Innovative Data Systems Research. London: DBLP, 2011:235-240.

[7] BELLARD F. QEMU, a fast and portable dynamic translator[C]// Proceedings of the 2005 Annual Conference on USENIX Technical Conference. Berkeley, CA: USENIX Association, 2005:41.

[8] KIVITY A, KAMAY Y, LAOR D, et al. KVM: the Linux virtual machine monitor[EB/OL]. [2017-06-20]. http://www.landley.net/kdocs/ols/2007/ols2007v1-pages-225-230.pdf.

[9] SAYOOD K. 数据压缩导论[M]. 4 版. 北京:人民邮电出版社, 2014:4. (SAYOOD K. Introduction to Data Compression[M]. 4th ed. Beijing: Post & Telecom Press, 2014:4.)

[10] AGHA S. Database compression techniques for performance optimization[C]// Proceedings of the 2010 2nd International Conference on Computer Engineering and Technology. Piscataway, NJ: IEEE, 2010:714-717.

[11] Zlib technical details[EB/OL]. [2017-06-01]. https://zlib.net/zlib_tech.html.

[12] LZ4: Extremely fast compression algorithm[EB/OL]. [2017-06-01]. https://github.com/lz4/lz4.

[13] ZIV J, LEMPEL A. A universal algorithm for sequential data compression[J]. IEEE Transactions on Information Theory, 1977, 23(3):337-343.

[14] InnoDB page structure[EB/OL]. [2017-06-01]. https://dev.mysql.com/doc/internals/en/innodb-page-structure.html.

[15] MAKATOS T, KLONATOS Y, MATAZAKIS M, et al. ZBD: using transparent compression at the block level to increase storage space efficiency[C]// Proceedings of the 2010 International Workshop on Storage Network Architecture and Parallel I/Os. Piscataway, NJ: IEEE, 2010:61-70.

[16] MA J, YIN B, KONG Z, et al. Leveraging page-level compression in MySQL - a practice at Baidu[C]// Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA. Piscataway, NJ: IEEE, 2017:1085-1092.

[17] TICKOO O, IYER R, ILLIKKAL R, et al. Modeling virtual machine performance: challenges and approaches[J]. ACM SIGMETRICS Performance Evaluation Review, 2010, 37(3):55-60.

[18] CHEN Q, LIANG L, XIA Y, et al. Mitigating sync amplification for copy-on-write virtual disk[C]// Proceedings of the 14th USENIX Conference on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016:241-247.

[19] AXBOE J. Linux block IO—present and future[EB/OL]. [2017-06-01]. http://www.linuxinsight.com/files/ols2004/axboe-reprint.pdf.

(下转第 1435 页)

从图 4 可以看出,本文算法得到的检测结果相比另外 5 种算法更接近人工标注图。对比本文算法和 LBE 算法的检测结果,在第四和第五行中,本文算法引入的颜色相异图准确地区分了深度相近颜色不同的目标和背景。在第六和第七行,目标形状结构复杂情况下,本文算法检测出的目标区域更加完整清晰,这是由于加入的多级分割处理提高了本文算法对结构复杂目标的检测能力。

3　结语

本文在 LBE 算法基础上进行改进,在计算显著性特征前对输入图像进行多级分割能充分利用相邻区域间的相关性;引入颜色相异图,与深度信息相互补充。改进后的算法能够很好地发挥深度信息和颜色信息的作用,提高了原始 LBE 算法的检测效果。

参考文献(References)

[1] ZHU J Y, WU J, XU Y, et al. Unsupervised object class discovery via saliency-guided multiple class learning[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, 37(4): 862 – 875.

[2] CHRISTOPOULOS C, SKODRAS A, EBRAHIMI T. The JPEG2000 still image coding system: an overview[J]. IEEE Transactions on Consumer Electronics, 2000, 46(4): 1103 – 1127.

[3] 刘尚旺, 李名, 胡剑兰, 等. 基于视觉显著性检测的图像分类方法[J]. 计算机应用, 2015, 35(9): 2629 – 2635. (LIU S W, LI M, HU J L, et al. Image classification method based on visual saliency detection[J]. Journal of Computer Applications, 2015, 35(9): 2629 – 2635.)

[4] 吴喆, 曾接贤, 高琪琪. 显著图和多特征结合的遥感图像飞机目标识别[J]. 中国图象图形学报, 2017, 22(4): 532 – 541. (WU Z, ZENG J X, GAO Q Q. Aircraft target recognition in remote sensing images based on saliency images and multi-feature combination[J]. Journal of Image and Graphics, 2017, 22(4): 532 – 541.)

[5] LIU T, YUAN Z, SUN J, et al. Learning to detect a salient object[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2011, 33(2): 353 – 367.

[6] CHENG M M, MITRA N J, HUANG X, et al. Global contrast based salient region detection[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, 37(3): 569 – 582.

[7] YAN Q, XU L, SHI J, et al. Hierarchical saliency detection[C]// Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition. Washington, DC: IEEE Computer Society, 2013: 1155 – 1162.

[8] 张巧荣. 利用背景先验的显著性检测算法[J]. 中国图象图形学报, 2016, 21(2): 165 – 173. (ZHANG Q R. Saliency detection algorithm based on background prior[J]. Journal of Image and Graphics, 2016, 21(2): 165 – 173.)

[9] TU W C, HE S, YANG Q, et al. Real-time salient object detection with a minimum spanning tree[C]// Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition. Washington, DC: IEEE Computer Society, 2016: 2334 – 2342.

[10] ZHANG Y, JIANG G, YU M, et al. Stereoscopic visual attention model for 3D video[C]// Proceedings of the 16th International Conference on Advances in Multimedia Modeling. Berlin: Springer-Verlag, 2010: 314 – 324.

[11] PENG H W, LI B, XIONG W, et al. RGBD salient object detection: a benchmark and algorithms[C]// ECCV 2014: European Conference on Computer Vision. Berlin: Springer, 2014: 92 – 109.

[12] JU R, LIU Y, REN T, et al. Depth-aware salient object detection using anisotropic center-surround difference[J]. Signal Processing: Image Communication, 2015, 38(C): 115 – 126.

[13] REN J, GONG X, YU L, et al. Exploiting global priors for RGB-D saliency detection[C]// Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops. Washington, DC: IEEE Computer Society, 2015: 25 – 32.

[14] SONG H, LIU Z, DU H, et al. Saliency detection for RGBD images[C]// Proceedings of the 7th International Conference on Internet Multimedia Computing and Service. New York: ACM, 2015: Article No. 72.

[15] FENG D, BARNES N, YOU S, et al. Local background enclosure for RGB-D salient object detection[C]// Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition. Washington, DC: IEEE Computer Society, 2016: 2343 – 2350.

[16] FELZENSZWALB P F, HUTTENLOCHER D P. Efficient graph-based image segmentation[J]. International Journal of Computer Vision, 2004, 59(2): 167 – 181.

[17] JIANG H, WANG J, YUAN Z, et al. Salient object detection: a discriminative regional feature integration approach[C]// Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition. Washington, DC: IEEE Computer Society, 2013: 2083 – 2090.

This work is partially supported by the Chongqing Postgraduate Research Innovation Fund (CYS15166).

YUAN Quan, born in 1967, M. S., senior engineer. His research interests include digital image processing, communication network.

ZHANG Jianfeng, born in 1992, M. S. candidate. His research interests include image saliency detection.

WU Lizhi, born in 1992, M. S. candidate. His research interests include video object tracking.

(上接第 1409 页)

[20] MAUERER W. 深入 Linux 内核架构[M]. 北京:人民邮电出版社, 2010:761 – 763. (MAUERER W. Professional Linux Kernel Architecture[M]. Beijing: Post & Telecom Press, 2010: 761 – 763.)

[21] 马煜翔. MySQL 页面级压缩性能分析及优化研究[D]. 天津:南开大学, 2015:22 – 27. (MA Y X. Performance analysis and optimization research of page-level compression in MySQL[D]. Tianjin: Nankai University, 2015:22 – 27.)

[22] FRÜHWIRT P, HUBER M, MULAZZANI M, et al. InnoDB database forensics[C]// Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications. Washington, DC: IEEE Computer Society, 2010: 1028 – 1036.

[23] How compression works for InnoDB tables[EB/OL]. [2017-06-01]. https://dev.mysql.com/doc/refman/5.7/en/innodb-compression-internals.html.

[24] Sysbench[EB/OL]. [2017-06-01]. https://github.com/akopytov/sysbench.

[25] CONRAD A. Database economic cost optimization for cloud computing[EB/OL]. [2017-06-01]. https://cs.brown.edu/research/pubs/theses/masters/2009/conrad.pdf.

ZHANG Jiachen, born in 1994, M. S. candidate. His research interests include storage virtualization.

LIU Xiaoguang, born in 1974, Ph. D, professor. His research interests include distributed system, search engine.

WANG Gang, born in 1974, Ph. D, professor. His research interests include cloud storage, search engine.