

连接操作在 SIMFS 和 EXT4 上的性能比较

赵利伟¹ 陈咸彰¹ 诸葛晴凤^{1,2}

(重庆大学计算机学院 重庆 400044)¹

(信息物理社会可信服务计算教育部重点实验室 重庆 400044)²

摘要 连接操作是关系数据库系统中最基本、最昂贵的操作,对数据库性能有巨大的影响。由于连接表存放在文件系统中,因此文件系统的性能对连接操作的性能有决定性的影响。不同文件系统的连接操作性能测试对数据库研究有重要意义,但目前相关测试较少。首先对比分析了新型内存文件系统 SIMFS(Sustainable In-Memory File System)的数据读写路径与磁盘文件系统 EXT4(Fourth Extended File System)I/O 路径等方面的差异;然后设计实验测试了不同文件系统对连接操作的影响,其中对 SIMFS 和 EXT4 分别设置了不同的数据读写块大小和 I/O 块大小等测试指标。实验表明,连接操作在 SIMFS 和 EXT4 上的性能优化、块大小影响、性能提升瓶颈、硬件约束等方面均存在明显差异。在实验结果比较分析的基础上,给出了针对新型内存文件系统连接操作的优化建议。

关键词 连接操作,内存文件系统,磁盘文件系统,性能优化

中图分类号 TP302.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.6.037

Performance Comparison of Join Operations on SIMFS and EXT4

ZHAO Li-wei¹ CHEN Xian-zhang¹ ZHUGE Qing-feng^{1,2}

(College of Computer Science, Chongqing University, Chongqing 400044, China)¹

(Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing 400044, China)²

Abstract Join is the most primary and expensive operation in relational database, and it has a great impact on the performance of the database. Since the data tables are stored in file system, the performance of the file system essentially determines the performance of join operation. The tests of join among different file systems have great meaning in database research, but now there are few of such tests. First, the differences between the data access of new in-memory file system SIMFS (Sustainable In-Memory File System) and the I/O path of disk-based file system EXT4 (Fourth extended file system) were compared. Then experiments were designed to test the effect of different file systems on join operations. And test metrics such as different data block and I/O block sizes were set for SIMFS and EXT4 respectively. The experimental results show that the join operation on SIMFS and EXT4 has obvious difference in performance optimization, effect of block size, the bottleneck of performance improvement, constraints of hardware, and so on. Based on the analysis of the experimental results, suggestions on in-memory file system were proposed to optimize the join operations.

Keywords Join operations, In-memory file system, Disk-based file system, Performance optimization

1 引言

文件系统作为数据库的基础设施,对数据库操作的性能有至关重要的影响。然而,目前基于传统磁盘文件系统的I/O操作已经成为计算机系统的性能瓶颈^[1],限制了数据库性能的提升。

新型非易失性内存(Non-Volatile Memory, NVM)以及建在其上的内存文件系统具有突破传统系统 I/O 性能瓶颈的潜

力。NVM 具有非易失、存储密度大、内存级访问速度和抗震动等优点,未来将取代磁盘成为新的数据存储设备。研究界已提出针对 NVM 的新型内存文件系统,例如 SCMFS^[2]、BPFS^[3]和 PMFS^[4]等,颠覆了传统面向块设备文件系统的 I/O 路径,对应用程序的设计和使用造成了深层次的影响。因此,新型内存文件系统对数据库性能的影响成为一个亟需探讨的问题。

为此,本文选取数据库系统中性能昂贵、使用广泛的连接

到稿日期:2015-05-07 返修日期:2015-08-18 本文受“863”国家高技术研究发展计划(2013AA013202, 2015AA015304),国家自然科学基金项目(61472052, 61173014)资助。

赵利伟(1989—),男,硕士生,主要研究方向为数据库算法优化, E-mail: lwzhao@cqu.edu.cn; 陈咸彰(1989—),男,博士生,主要研究方向为面向大数据应用的新型体系结构和系统优化、内存计算系统, E-mail: xzchen@cqu.edu.cn; 诸葛晴凤(1970—),女,博士,教授,主要研究方向为面向大数据应用的新型体系结构和系统优化、内存计算系统、物联网系统、嵌入式系统、先进存储体系结构、软/硬件协同设计等, E-mail: qzhuqe@cqu.edu.cn.

(Join)操作^[5],分别在自主设计实现的内存文件系统 SIMFS (Sustainable In-Memory File System)^[6]和磁盘文件系统 EXT4^[7]上进行对比测试。根据实验结果分析不同的文件系统和文件读写路径对 Join 算法的影响,并且提出系统设计和优化建议。

2 内存文件系统

近年来出现的新型非易失性内存,例如相变存储器 (Phase Change Memory, PCM)^[8]、阻变式存储器 (Resistive Random Access Memory, RRAM)^[9]、忆阻器 (Memristor)^[10]和磁阻式存储器 (Magnetoresistive Random Access Memory, MRAM)^[11]等,具有访问速度快、可按字节寻址、功耗低、存储密度大^[12]等特点。PCM 的读写性能与 DRAM 相近,存储密度接近闪存,能耗不及磁盘能耗的 1/10^[12]。鉴于其更强的抗震动和抗辐射干扰能力,NVM 将在未来成为主流的存储设备^[13]。

NVM 已引起传统存储架构的变革^[14,15]。NVM 可以直接连接在内存总线上,使用 CPU 的 load/store 指令和虚拟地址实现按字节直接访问 NVM。然而传统磁盘文件系统和 I/O 路径的设计却是针对连接在 I/O 总线上的慢速块设备。因此,传统磁盘文件系统及其 I/O 路径并不适于管理 NVM,需要使用针对 NVM 特性的内存文件系统。

图 1 示出 Linux 系统中文件系统的软件层次。如图所示,进程发出文件读写请求后,通过虚拟文件系统层访问文件所在的文件系统。传统磁盘文件系统,例如 EXT4,要经过通用块层、I/O 调度层和块设备驱动等软件层次才能访问存储设备即磁盘。内存文件系统却不再需要经过这些层次,而是利用高度优化的内存管理访问 NVM。内存文件系统,例如 SCMFs 和 PMFs,使用虚拟地址和硬件内存管理单元 (Memory Management Unit, MMU) 直接访问作为存储的内存。

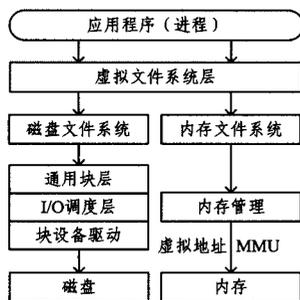


图 1 Linux 系统中的文件系统软件层次

本文使用的内存文件系统 SIMFS 是一种基于文件虚拟地址空间的新型内存文件系统。SIMFS 中的每个文件都有独立且连续的文件虚拟地址空间,其表现形式为文件页表。文件页表的结构与内存页表的结构类似,即上层页表项存放下一层页表某一页的起始物理地址,底层页表存放指向数据页的指针。SIMFS 实现在 Linux 内核中。当文件打开后, SIMFS 分配一段内核虚拟地址空间作为文件的虚拟地址,所以 SIMFS 可以通过内核虚拟地址在文件与用户缓存区之间直接拷贝数据。当由于 SIMFS 中的文件有连续的虚拟地址空间,因此无论文件的物理页面是否连续,都可以利用 MMU 与文件页表将连续的虚拟地址高速地转换为物理地址。所以 SIMFS 访问文件时不需要查找文件元数据,可以借助 MMU 连续地高速访问文件数据。文件关闭时, SIMFS 将解除文件

虚拟地址空间与内核虚拟地址空间的联系。

SIMFS 提供与虚拟文件系统层对应的访问接口,不再经过传统文件系统的各种软件层次,也不在页高速缓存中缓存文件读写的数据,而是直接在 SIMFS 的文件和用户进程缓冲区之间拷贝数据。因此, SIMFS 没有缓存、合并和调度 I/O 请求的开销,可以避免在多个层次之间拷贝文件数据的开销。与基于磁盘的文件系统相比, SIMFS 的数据读取速度有较为明显的提升^[6]。

3 Nest-Loop-Join 算法

算法 1 Nest-Loop-Join

输入:待连接的两个关系表 R 和 S(关系 R 中的记录用 r 表示,关系 S 中的记录用 s 表示)

输出:R 和 S 的连接结果表 Q

1. for 关系 R 的每条记录 r do
2. for 关系 S 的每条记录 s do
3. if r 和 s 满足连接条件
4. then 输出结果记录 (r, s) 到表 Q

Nest-Loop-Join 算法是数据库连接算法中最基本的连接算法。它根据连接操作的定义,将需要连接的关系表中的一个作为关系内表,另外一个作为关系外表。对于关系外表中的每一条记录,该算法循环读取关系内表的所有记录与其进行比较,连接符合连接条件的记录并将结果存放到结果表中^[16]。

实际上, Nest-Loop-Join 算法的具体执行是使用的 Block-Nest-Loop-Join 方法^[17],也就是将记录组织为 512B、1kB 和 4kB 等大小的块进行读写。该方法每次读取关系内表的一个块,再根据可用内存读取关系外表的数据;将已读取的关系内表的所有记录与读入的关系外表的所有记录进行连接;将连接结果组织为一个块并写回文件系统。重复此过程,直到获取所有的连接结果。

磁盘以块为基本单位读写数据,例如 512B 和 1kB 等,而记录的大小通常小于块大小,因此在磁盘文件系统中使用 Block-Nest-Loop-Join 有两个好处:第一,以块为单位进行 I/O 操作可以减少 I/O 请求的数量,从而减小 I/O 软件层次带来的开销;第二,将记录组织为块能够提高单次 I/O 读写磁盘数据的效率。因此,访问磁盘文件系统时,连接算法的性能会随着每一次 I/O 操作的数据量大小而产生明显变化。在内存文件系统上,连接算法的性能也随着数据读写操作的数据量的大小有一定变化。不同的是,因为内存文件系统读写数据开销小、带宽基数较大,所以内存文件系统连接算法的性能受到数据读写操作的数据量大小的影响并没有磁盘文件系统明显。按块读取数据是磁盘文件系统连接操作常用的一种优化方法,其他关于如何改变数据读写来优化连接算法的研究很多,如文献^[18]介绍的如何利用高速缓存来优化磁盘连接算法等。

4 实验结果分析与优化建议

本文在 Linux 内核中实现 Block-Nest-Loop-Join 算法,分别测试它在磁盘文件系统 EXT4 和内存文件系统 SIMFS 上的性能。

4.1 实验环境与方法

本实验使用的计算机为 Dell OptiPlex 390,系统配置如表 1 所列。实验划分 8 GB DRAM 用于模拟 NVM,将 SIMFS

挂载在模拟的 NVM 上进行实验。

表 1 测试系统配置

系统组件	配置情况
CPU 核心	Intel(R) Core(TM) i5-2400 3.10GHz, 4 核 4 线程
内存	16GB, 最大带宽约为 20GB/s
磁盘	500GB, 转速为 7200rpm, SATA 接口
内核	Linux 3.11.8

待连接关系表 R 中有 100 条记录。实验按照关系表 S 的大小分 4 个测试组。测试组 1、组 2、组 3 和组 4 中关系表 S 分别有 10^3 条、 10^4 条、 10^5 条和 10^6 条记录, 每条记录的大小为 64 字节。实验中, 内存文件系统对关系表的读取和符合连接操作记录的写回以及磁盘文件系统对数据的 I/O 操作均采用相同大小的块。后文中, 对内存文件系统的数据读写块大小和磁盘文件系统的 I/O 块大小都统称为 I/O 大小, 测试得到的连接操作总耗时简称为总时间。实验测试不同 I/O 操作的块大小对连接操作的性能影响。

4.2 实验结果分析与优化建议

从 5 个角度分析连接测试实验结果并给优化建议。

4.2.1 不同文件系统中连接操作的性能

表 2 和表 3 分别列出 4 个测试组的连接操作在 SIMFS 和 EXT4 上的总执行时间。

表 2 测试组 1 和测试组 2 中连接操作的总时间

I/O 大小	测试组 1		倍率	测试组 2		倍率
	EXT4 (s)	SIMFS (ms)		EXT4 (s)	SIMFS (ms)	
512	9.3	14.2	658.4	261.9	60.0	4362.6
1k	12.0	9.2	1301.9	121.3	40.4	3004.8
4k	4.1	6.9	585.5	29.7	20.2	1466.8
16k	1.3	6.6	194.2	11.9	16.6	719.4
64k	0.3	6.5	47.6	4.0	15.7	256.0
256k	0.1	6.6	17.6	1.4	13.8	97.4
512k	0.1	4.6	25.6	0.7	13.9	53.6
1M	0.1	5.6	13.0	0.5	15.6	29.3
4M	0.1	5.5	13.5	0.4	15.7	24.3
16M	0.1	5.5	13.1	0.4	15.6	26.0

表 3 测试组 3 和测试组 4 中连接操作的总时间

I/O 大小	测试组 3		倍率	测试组 4		倍率
	EXT4 (s)	SIMFS (ms)		EXT4 (s)	SIMFS (ms)	
512	2746.7	509.8	5387.5	26776.5	4960.2	5398.2
1k	1575.5	299.1	5267.6	14391.2	2905.6	4952.9
4k	329.9	144.3	2286.3	3766.2	1412.5	2666.3
16k	128.3	105.1	1221.0	1271.8	1027.3	1237.9
64k	41.1	102.3	401.6	392.5	937.2	418.8
256k	12.2	108.8	112.0	96.6	986.1	97.9
512k	8.1	107.0	75.7	76.0	991.6	76.6
1M	7.1	110.7	64.4	66.5	989.6	67.2
4M	7.3	137.5	53.1	60.7	1254.5	48.4
16M	7.8	163.0	47.9	58.9	1508.7	39.1

相较于磁盘文件系统 EXT4, 内存文件系统 SIMFS 能够显著提高连接算法的性能。如表 3 所列, SIMFS 的性能提升可达 5000 倍以上。原因有两点: 第一, 内存的读写速度要高于磁盘的读写速度; 第二, 内存文件系统不经过磁盘文件系统的块设备驱动层、I/O 调度层、通用块层等层次, 软件开销小。所以, 内存文件系统可以突破磁盘文件系统中连接算法的 I/O 瓶颈, 提高连接算法的性能。

4.2.2 连接操作在不同文件系统中的 I/O 与计算瓶颈

分析图 2 可知连接操作在两种文件系统中的性能瓶颈。连接算法在内存文件系统和磁盘文件系统上的性能瓶颈各不相同, 需要采取不同的优化策略。

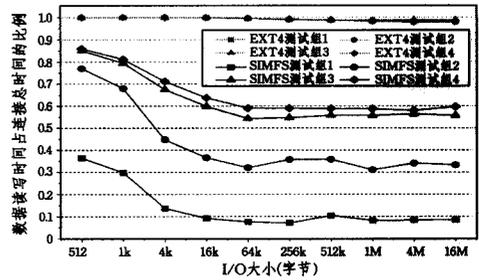


图 2 数据读写时间占总时间的比例

如图 2 所示, 在全部测试组和 I/O 大小中, 连接操作在 EXT4 上的 I/O 时间占总时间的 97% 以上, 所以连接操作在 EXT4 上的性能瓶颈是 I/O 操作。因此, 优化 EXT4 文件系统中的连接操作时, 着重优化 I/O 操作会有较好的效果。

在 SIMFS 上使用连接操作时, 时间比例表现出两个特征。第一, 数据读写时间占连接总时间的比例较大, 且关系表越大, 数据读写时间占总时间的比例越大。例如测试组 1 中, I/O 时间占总时间的比例较小, 在 40% 以下; 测试组 4 中, I/O 时间占总时间的比例较大, 在 55% 以上。原因在于关系表越大, 完成连接操作所需的 I/O 操作越多, 因此连接操作中 I/O 请求导致的其他额外开销累积也越大。第二, I/O 请求的大小超过 64k 字节时, 连接操作的计算时间占总时间的比例都在 40% 以上, 计算时间占较大比重。所以在内存文件系统中进行连接操作时, 优化连接算法的计算量可以大幅提高性能。

此外, 在 EXT4 和 SIMFS 中, 所有测试组中连接操作的数据读写时间占总时间的比例都随 I/O 大小的增大有一定的减少。因为在每个测试组的数据量一定的情况下, 增大 I/O 大小可以减少 I/O 请求次数。因此, 应当尽量增大连接操作中单个 I/O 操作读写文件的数据量, 减小 I/O 时间在总时间中的百分比, 提高连接操作中数据的读写性能。

4.2.3 I/O 大小与连接操作的性能优化

图 3 和图 4 分别展示了不同文件系统中连接操作的总时间受 I/O 大小的影响情况。纵轴所示的增幅是用 I/O 大小为 512 字节时的连接操作的总时间除以其他相应 I/O 大小对应的连接操作的总时间得到的。

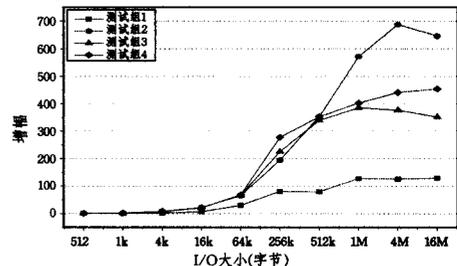


图 3 EXT4 文件系统 I/O 大小对连接性能的提升情况

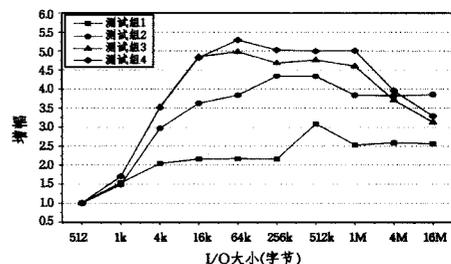


图 4 SIMFS 文件系统 I/O 大小对连接性能的提升情况

由图3可以看出,连接操作在EXT4上运行时,I/O大小越大,连接操作的性能越好。在不同的测试组中,连接操作最大提升性能在100到700倍不等。因为磁盘文件系统相对内存文件系统I/O软件层次较多,开销大,对于单个测试组来说,增大I/O可以减少I/O操作的次数,能够较大程度地减小软件层次带来的开销。

由图4可以看出,连接操作在SIMFS上运行时,随着I/O的增大,连接操作的性能也在提升,但提升效果并没有EXT4文件系统的明显。在不同测试组中,连接操作的性能提升1.5~5.5倍。然而不同的测试组在不同的I/O大小达到最优性能,而且当I/O大小超过1M时,连接操作的性能有所下降。所以在内存文件系统中实现连接操作时,应当根据连接表的大小选取适当大小的I/O操作,以取得最优性能。

4.2.4 适于内存文件系统连接操作的硬件优化

图5和图6分别展示连接操作读写不同文件系统的吞吐量。在读写SIMFS时,I/O大小超过1M字节后,数据量较大的测试组3和测试组4吞吐量有较大下降。原因在于SIMFS使用CPU指令load/store读写内存,当I/O增大或文件增大后,数据量较大引起更多的CPU Cache miss和TLB miss,最终使得系统性能有所降低。然而在读写EXT4时,并没有出现与SIMFS类似的现象,这是因为磁盘文件系统利用硬件DMA控制器读写磁盘,由DMA控制器直接控制总线,占用的CPU资源相对较少。内存文件系统的性能不仅受到CPU的影响,还受到内存控制器和内存总线的约束。因为内存文件系统使用内存总线传输数据,所以内存文件系统吞吐率的极限就是内存总线的带宽。内存总线的带宽由内存控制器和内存总线决定。

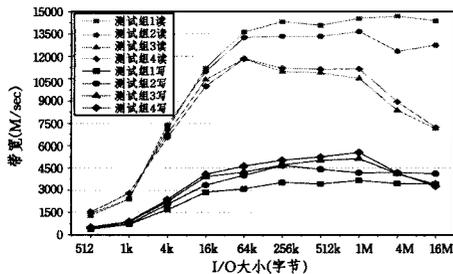


图5 SIMFS连接操作的吞吐量

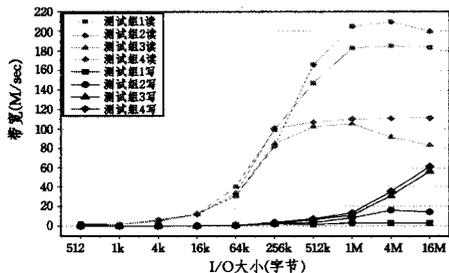


图6 EXT4连接操作的吞吐量

综上,SIMFS受CPU、内存控制器和内存总线的影响较大。本文认为,未来需要增设独立的硬件设备用于访问作为存储的内存,保证计算不受I/O操作影响,提高系统的整体性能。

4.2.5 内存文件系统连接算法的优化建议

表4和表5分别以百分比列出连接操作读和写SIMFS时的内存带宽利用率。

表4 连接操作读SIMFS时的内存带宽利用率

I/O大小	读带宽(%)			
	测试组1	测试组2	测试组3	测试组4
512	6.06	6.73	7.17	7.29
1k	11.36	11.38	13.12	13.21
4k	33.51	34.72	31.90	30.76
16k	52.76	51.65	49.10	46.94
64k	64.03	62.27	55.60	55.67
256k	67.29	62.67	51.60	52.68
512k	66.13	62.67	51.25	52.36
1M	68.33	64.20	49.46	52.48
4M	69.06	58.04	39.37	42.04
16M	67.54	59.96	33.64	34.00

表5 连接操作写SIMFS时的内存带宽利用率

I/O大小	写带宽(%)			
	测试组1	测试组2	测试组3	测试组4
512	1.82	2.18	2.39	2.42
1k	3.31	3.42	4.12	4.24
4k	7.97	9.69	10.52	11.16
16k	13.44	15.76	18.40	19.21
64k	14.54	18.83	19.91	21.73
256k	16.64	21.92	22.04	23.59
512k	16.17	20.77	23.56	24.70
1M	17.26	19.68	24.12	26.11
4M	16.29	19.76	19.38	19.64
16M	16.17	19.42	16.08	15.37

从表4和表5中可以看出,连接操作在读SIMFS时的内存带宽利用率为6.06%~69.06%,写SIMFS时的内存带宽利用率为1.82%~26.11%,内存带宽仍长期处于不完全利用的状态。

由于SIMFS支持多个进程同时读文件系统,也支持多个进程使用记录锁写同一个文件,因此在多核或多处理器系统中,使用多核或多处理器并行执行的连接操作可以大幅提高系统资源的利用率和连接操作的性能。

此外,内存设备具有可字节寻址的特征,所以无论关系表使用哪种数据存储方式,都可以直接按位读取关系表中的待连接属性进行连接比对,不用读取整条记录,可以减少连接操作的数据读取量,并且避免其他属性在不同内存位置之间的拷贝,提高连接操作的性能。

现有的连接算法都是先将关系表的文件数据从文件系统中读到内存中,例如系统的页高速缓存或用户的缓冲区。因为关系表的数据已经在内存中,所以这种数据拷贝的过程对于内存文件系统而言是多余的。例如SIMFS之类的内存文件系统都已经支持内存映射(即mmap)的文件读写方式。这种方式可以直接将关系表的文件数据所在的物理内存映射到用户进程的虚拟地址空间中,用户不用拷贝就可以直接读写关系表的数据。因此,可以设计新的连接算法,直接在文件系统中执行就地(In-place)连接操作,而不再在文件和用户缓冲区之间拷贝数据。就地连接操作没有读取关系表数据的开销,并且节省了内存空间。

这种基于内存映射的就地连接操作有两点不足:1)建立映射需要耗费较多时间;2)把连接结果写回新的关系表时,会因为存放结果的文件不断增大而引起缺页异常。更进一步的优化方法应该是设计内存文件系统连接操作的接口,内核态下在文件系统内部就地完成连接操作。

5 相关工作

文献[19]结合磁盘连接优化方法对固态硬盘连接算法进

(下转第207页)

ordered information systems[J]. Knowledge-Based Systems, 2012, 27(3): 78-91

[9] Inuiguchi M, Yoshioka Y. Several reducts in dominance-based rough set approach[M]//Interval/Probabilistic Uncertainty and Non-Classical Logics. Springer Berlin Heidelberg, 2008: 163-175

[10] Kusunoki Y, Inuiguchi M. A unified approach to reducts in dominance-based rough set approach[J]. Soft Computing, 2010, 14(5): 507-515

[11] Susmaga R. Reducts and constructs in classic and dominance-based rough sets approach[J]. Information Sciences, 2014, 271(7): 45-64

[12] Du W S, Hu B Q. Approximate distribution reducts in inconsis-

tent interval-valued ordered decision tables[J]. Information Sciences, 2014, 271(7): 93-114

[13] Shao M W, Zhang W X. Dominance relation and rules in an incomplete ordered information system[J]. International Journal of Intelligent Systems, 2005, 20(1): 13-27

[14] Yang X B, Yang J Y, Wu C, et al. Dominance-based rough set approach and knowledge reductions in incomplete ordered information system[J]. Information Sciences, 2008, 178(4): 1219-1234

[15] Yang X B, Yu D J, Yang J Y, et al. Dominance-based rough set approach to incomplete interval-valued information system[J]. Data & Knowledge Engineering, 2009, 68(11): 1331-1347

(上接第 187 页)

行比较分析,提出针对固态硬盘连接算法的优化方法和建议。但固态硬盘仍是块存储设备,文件读取速度不能与非易失性内存相比。文献[20]提出了对非易失性内存的有限写操作的排序连接算法,通过先排序的方法调整数据的读写量,达到通过读来减少写的连接优化方法。随着大数据时代的到来,文献[21]对大数据连接的研究进展进行了分析介绍,并提出了一些相关问题和解决方案。

结束语 本文首先对比分析了传统磁盘文件系统和新型内存文件系统的特点,然后在两种文件系统中对连接操作进行性能测试分析。实验结果表明,内存文件系统能够显著提高连接操作的性能。相信内存文件系统能够突破传统的 I/O 瓶颈,给数据库性能带来巨大的提升。

对实验结果进行进一步分析,需要根据内存文件系统的特点设计全新的连接算法、优化系统配置甚至硬件设计等。本文结合实验结果和内存文件系统的特点对这些问题进行了讨论,并且提出了优化建议。我们将在未来的工作中实现并验证这些优化建议。

参 考 文 献

[1] Myers D C. On the Use of NAND Flash Memory in High-Performance Relational Databases[D]. Massachusetts Institute of Technology, 2008

[2] Wu X, Qiu S, Narasimha Reddy A L. SCMFS: A File System for Storage Class Memory and its Extensions[J]. ACM Transactions on Storage (TOS), 2013, 9(3): 1-11

[3] Condit J, Nightingale E B, Frost C, et al. Better I/O through byte-addressable, persistent memory[C]// Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles. ACM, 2009: 133-146

[4] Dullloor S R, Kumar S, Keshavamurthy A, et al. System software for persistent memory[C]// Proceedings of the Ninth European Conference on Computer Systems. ACM, 2014: 1-15

[5] Li Guan-zhao, Chen Si-tong, Zhen Zhen, et al. Join Algorithms Based on Fermi Architecture[J]. Computer Science, 2013, 40(3): 62-67 (in Chinese)
李观钊,陈思桐,甄真,等.基于 Fermi 架构的 Join 算法[J].计算机科学, 2013, 40(3): 62-67

[6] Sha E H M, Chen Xian-zhang, Zhuge Qing-feng, et al. Designing an efficient persistent in-memory file system[OL]. <http://cacs.cqu.edu.cn/wp-content/uploads/2015/02/TR-2014-02-Designing-an-efficient-persistent-in-memory-file-system.pdf>

[7] Ext4 wiki[OL]. <https://ext4.wiki.kernel.org>

[8] Raoux S, Burr G W, Breitwisch M J, et al. Phase-change random

access memory: A scalable technology[J]. IBM Journal of Research and Development, 2008, 52(4/5): 465-479

[9] Jung M, Shalf J, Kandemir M. Design of a large-scale storage-class RRAM system[C]//Eugene, Oregon, USA; Proceedings of the 27th International ACM Conference on International Conference on Super Computing. ACM, 2013: 103-114

[10] Chua L. Resistance switching memories are memristors[J]. Applied Physics A, 2011, 102(4): 765-783

[11] Kerman J. Toward a Universal Memory[J]. Science, 2005, 308(5721): 508-510

[12] Chen S, Gibbons P B, Nath S. Rethinking Database Algorithms for Phase Change Memory[C]//Proceedings of the 5th Biennial Conference on Innovative Data System. 2011: 21-31

[13] Jung M, Shalf J, Kandemir M. Design of a large-scale storage-class RRAM system[C]//Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. ACM, Eugene, Oregon, USA, 2013: 103-114

[14] Liu R, Shen D, Yang C, et al. NVM duet: unified working memory and persistent store architecture[J]. ACM Sigplan Notices, 2014, 42(1): 455-470

[15] Jung J, Cho S. Memorage: emerging persistent RAM based malleable main memory and storage architecture[C]// Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. ACM, Eugene, Oregon, USA, 2013: 115-126

[16] Mishra P, Eich M H. Join processing in relational databases[J]. ACM Comput. Surv., 1992, 24(1): 63-113

[17] Elmasri R A, Navathe S B. Fundamentals of Database Systems [M]. Addison-Wesley Longman Publishing Co., 1999: 1009

[18] Han Xi-xian, Yang Dong-hua, Li Jian-zhong. DBCC-Join: A Novel Cache-Conscious Disk-Based Join Algorithm[J]. Chinese Journal of Computers, 2010(08): 1500-1511 (in Chinese)
韩希先,杨东华,李建中. DBCC-Join: 一种新的高速缓存敏感的磁盘连接算法[J].计算机学报, 2010(8): 1500-1511

[19] Do J, Patel J M. Join processing for flash SSDs: remembering past lessons[C]// Proceedings of the Fifth International Workshop on Data Management on New Hardware. ACM, Providence, Rhode Island, 2009

[20] Viglas S. Write-limited sorts and joins for persistent memory [J]. PVLDB, 2014, 7(5): 413-424

[21] Pang Jun, Yu Ge, Xu Jia, et al. Similarity Joins on Massive Data Based on MapReduce Framework[J]. Computer Science, 2015, 42(1): 1-5 (in Chinese)
庞俊,于戈,许嘉,等.基于 MapReduce 框架的海量数据相似性连接研究进展[J].计算机科学, 2015, 42(1): 1-5