DOI:10.7544/issn1000-1239.2018.20170868
55(2):319-326,2018

X-DB:软硬一体的新型数据库系统

张铁赢　黄　贵　章颖强　王剑英　胡　炜　赵殿奎　何登成
(阿里巴巴集团　杭州　310023)
(tieying.zhang@alibaba-inc.com)

X-DB: Software and Hardware Co-Designed Database System

Zhang Tieying, Huang Gui, Zhang Yingqiang, Wang Jianying, Hu Wei, Zhao Diankui, and He Dengcheng
(*Alibaba Group Inc*, *Hangzhou* 310023)

Abstract The field of database system has three stages of development. The first stage is when relational model was proposed by E. F Codd. Relational model establishes the foundation of the database theory and database system. It contributes many database market giants, like IBM DB2, Microsoft SQLServer and Oracle. The second stage is due to the rapid development of Internet, which produces NoSQL database system. NoSQL focuses on system scalability but sacrifices transactional features. The third stage is called modern database era represented by new hardware features. Alibaba X-DB is such kind of database system. X-DB fully utilizes new hardware in different areas including storage, network, multi-core, parallel and heterogeneous computing. X-DB co-designs hardware and software and is compatible with MySQL ecosystem with the goal to renovate the relational database system.

Key words relational model; database system; distributed system; software and hardware co-design; new hardware

摘　要　数据库领域经历了3次发展时期:第1个时期起源于Codd提出的关系模型,奠定了数据库理论和系统的基础,并造就了早期的数据库商业巨头IBM DB2,Microsoft SQLServer和Oracle等;第2个时期由互联网的快速发展所推动,催生了NoSQL数据库系统,这一类数据库系统关注于系统可扩展性,但是牺牲了数据库的事务特性和SQL功能;第3个重要时期是以新硬件为基础的现代数据库时期,阿里巴巴数据库系统X-DB便属于这一时期的现代数据库.X-DB基于阿里巴巴大规模业务需求,充分利用新硬件的特性,围绕存储、网络、多核、并行和异构计算进行软硬一体协同设计,同时兼容MySQL生态,重塑关系型数据库体系结构.

关键词　关系模型;数据库系统;分布式系统;软硬协同设计;新硬件

中图法分类号　TP391

从20世纪70年代至今,关系型数据库系统已经发展了40余年.随着计算机系统硬件的迅猛发展,基于传统硬件体系结构的数据库系统很难发挥出应有的性能.同时,互联网时代的到来对传统关系型数据库提出了新的挑战.阿里巴巴作为全球最大的互联网商务平台,数据库系统是最重要的基础软件之一,并在某种程度上推动了阿里巴巴技术体系的变革.阿里巴巴数据库系统经历了4个阶段:第1阶段在阿里初创时期,业务体量相对较小,数据库架构为单机房MySQL单机数据库.第2阶段,随着

收稿日期:2017-11-15;**修回日期**:2017-12-18
通信作者:何登成(dengcheng.hedc@alibaba-inc.com)

阿里巴巴业务的快速发展,单机房单机数据库的架构已成为瓶颈,阿里巴巴将单机 MySQL 升级为单城市多机房 IOE 体系,瓶颈得以暂时缓解.第 3 阶段,单城市的容灾风险和商业 IOE 的成本逐渐成为风险点,阿里巴巴开发了基于 MySQL 的优化分支,并基于该分支实现了一整套异地多活架构.第 4 阶段,新硬件的不断出现,数据库向高效能、一体化、软硬件协同设计发展,阿里巴巴的数据库技术也进入了全新的阶段.

本文介绍阿里巴巴新一代分布式数据库系统 X-DB.基于大规模业务场景需求,X-DB 充分利用新硬件的特性,围绕存储、网络、多核、分布式和异构计算进行软硬一体协同设计,同时兼容 MySQL 生态,重塑关系型数据库体系结构.

1 背景与相关工作

数据库领域发展至今,经历了 3 个重要时期:

第 1 个时期起源于 1970 年 Codd 博士提出的关系模型[1].在关系模型中,Codd 论述了范式理论和关系系统.在此之前,网状模型和层次模型可以解决数据集中和共享的问题,但是在数据抽象上仍然有很大欠缺,用户需要明确数据的存储结构、存取路径等.关系模型的出现较好地解决了这些问题,其具有严格的数学基础,抽象级别较高,并且简单清晰,便于理解和实现,很快成为数据库市场的主流,造就了一批早期的数据库巨头,如 IBM DB2, Microsoft SQLServer, Oracle, Sybase 等.

第 2 个重要时期是 2000 年以后,随着互联网的快速发展,低成本的横向扩展成为数据库的首要需求,NoSQL 数据库应运而生.典型的系统如 BigTable[2], HBase[3], Cassandra[4], DynamoDB[5], MongoDB[6]等.这一类数据库系统与传统关系型数据库最大的区别是牺牲了事务特性(ACID),从而不提供或仅提供有限的 SQL 功能.访问模式较为单一,功能有限.NoSQL 数据库出现的根本原因是互联网快速发展需求与传统关系型数据库扩展性不强的矛盾导致的.

第 3 个重要时期开始于 2008 年前后,Harizopoulos 等人发表了论文[7],其明确指出新硬件的出现提供了传统关系数据库变革的基础,并在数据库系统 Shore[8]上进行了数据库关键模块的成本分析,指明了研究方向.这一时期,我们称之为“现代数据库”的元年.在此之后,工业界和学术界围绕“现代数据库”展开了一些列的研究,主要集中在多核(众核)[9-12]、多 CPU(NUMA)[13-14]、内存[15-16]、NVM[17-20]、网络(RDMA)[21-24]等.其中,对业界产生深远影响的数据库系统有 HStore[25](VoltDB[26]为其商业版本)、Hyper[27-28]、HANA[29]、Hekaton[30]、Spanner[31]等.阿里巴巴 X-DB 便属于这一时期的现代数据库.

X-DB 充分利用新硬件的特性,基于大规模业务需求,围绕存储、网络、多核、分布式和异构计算进行软硬一体协同设计,同时兼容 MySQL 生态,重塑关系型数据库体系结构.本文介绍了 X-DB 的设计理念和软硬一体模块,给出了相应测试结果并加以分析,证明了 X-DB 设计的有效性.

2 X-DB 系统设计

本节主要阐述阿里巴巴 X-DB 数据库的系统架构和设计思想.

2.1 设计概要

X-DB 定位为金融级全球可部署分布式数据库.金融级表明 X-DB 面向交易类型,支持多种隔离级,不仅可用于阿里巴巴大规模线上交易场景,还可以用于银行、证券等金融机构多种交易类型;全球可部署的含义为不同地理位置部署多副本,可以理解为通常所说的异地多活.但不同于传统 Oracle 或者 MySQL 的备份机制,X-DB 采用的是一体化设计的数据强一致多副本.其目的是提供简单高效的数据库运维方案,同时保障数据的高可用和强一致.

X-DB 分布式数据库采用单数据库实例多租户多数据库管理模式,通过租户来进行资源分配.X-DB 只包含一个数据库实例,在数据库实例中可以创建多个租户,每个租户中可以创建多个数据库和多个用户.租户是 X-DB 资源分配和资源隔离的基本单位,可以为每个租户分配不同的系统物理资源(CPU、内存、IOPS、磁盘空间、网络带宽等),租户间的资源是完全隔离的(不可相互占用).每个租户可以指定表数据的默认位置和副本位置.

图 1 为 X-DB 的系统架构图.X-DB 提供跨 Zone 和跨 Region 的数据强一致副本方案.数据分片为 share-nothing 的切分方式,每个分片即为 Partition(图 1 中 P_1, P_2, P_3, …).数据一致性使用 X-DB 优化的 Paxos 协议(X-Paxos)作为保障.使用 X-Paxos 进行强一致副本同步和选主,参与的所有成员构成一个 Paxos Group,其中选主成功的副本称为 Leader,

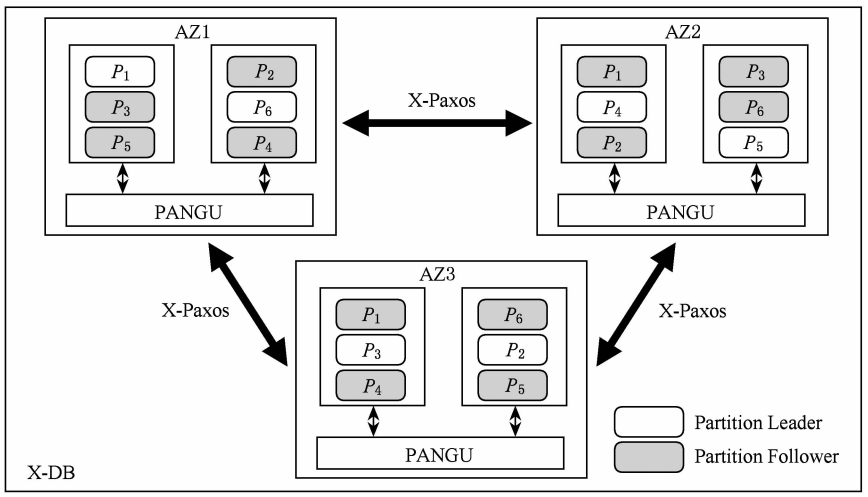


Fig. 1 X-DB architecture

图 1 X-DB 系统架构

其他副本每一个都称为 Follower,任意时刻只有 Leader 对外提供读写服务,Follower 同步 Leader 数据.每组 Paxos Group 至少需要 3 个参与成员.Router Service(SQL 路由服务)缓存表的分片位置信息和节点负载信息,然后根据 SQL 的分区键进行节点路由.可与 X-Driver 进行集成作为X-DB 专有驱动部署在客户端,也可以独立部署在云端,支持标准的 MySQL 驱动的客户端.对于确定的分区键条件则直接路由到数据所在的节点.对于不确定的分区键或无分区键条件的 SQL 则根据负载和就近计算原则进行节点选择.

数据存储使用了 X-DB 的数据引擎 X-Engine.X-Engine 包含了高效的内存索引结构,写入异步流水线处理机制以及一系列与硬件结合加速引擎性能的技术(将在第 3 节详细论述).同时,X-DB 也可以使用计算存储分离的方式,即无缝插拔阿里巴巴的分布式存储系统 PANGU.使用这种方式(计算存储分离)解决了数据库的快速弹性伸缩能力和存储的可扩展能力.

2.2 系统模块

本节我们介绍 X-DB 主要的系统模块,包括存储引擎、内存索引、并发控制和混合存储格式.

2.2.1 存储引擎

X-DB 的存储引擎称之为 X-Engine.X-Engine 使用分层存储的理念,利用数据不同的访问特点,采用相对应的存储格式,存放在适合的存储介质上,从整体上降低存储成本.X-Engine 借鉴了 LSM-Tree[32] 思想,写入的数据不会直接替换掉已有的数据,而是追加的方式写入内存表,并写入 WAL(write ahead log),内存表写入到一定尺寸会 Flush 到存储,写为有序只读的 SST(sorted string table)文件,当 SST 堆积到一定数量后,通过合并操作将相同 Key 的多个版本合并掉(只保持活跃事务引用的多版本,其他不再引用的只保留最新版本),保持尽可能少的 SST 文件.

本质上来说,这种存储结构是延迟了写入刷盘的操作,把所有的随机写入转换为顺序追加操作,同时使用后台任务批量将数据写入磁盘,避免了传统存储引擎由于大量随机写频繁刷脏页带来的写入放大的问题.这种结构对大量写入是非常友好的,使得我们在提升数据库写入吞吐上有机会做大量的优化,比起基于固定大小页面的传统存储引擎来说,写入速度有了量级的提升;但是伴随而来的的代价是读取路径变长,以及比单纯刷脏页更重量级的合并操作.

由于数据量的不断膨胀,很多应用每天新写入的数据量以 TB 计,而数据的热点非常明显,使用同一种设备或是方法来存储数据是一种浪费,而使用不同的存储系统来存储数据,又不得不面临多份数据搬迁同步,应用面对异构系统的复杂性等额外问题;因此在一套系统中使用混合存储成为一种必然.X-Engine 的核心理念依然是根据数据的冷热程度对数据分层,以最大程度的匹配当前多种不同的存储介质,对不同热度的数据使用不同的存储方法,以求在存储成本和性能之间取得平衡,获取最大收益.

2.2.2 内存索引

内存表是数据写入的第一落入点,也是元数据存储结构,要求读写都有极高的并发吞吐,X-Engine

采用了 Lock-Free(无锁)SkipList. SkipList 相对于 B-Tree 来说,实现比较简单,没有 B-Tree 的节点分裂操作,容易实现为 Lock-Free,缺点是因为数据是用链表链接起来,相邻的数据没有存储在连续的内存中,因此 CPU cache miss 会比较高. 我们比较过一些内存索引数据结构的性能: SkipList, B-Tree, MassTree[33], Bw-Tree[34],目前的实现中 MassTree 性能最好,MassTree 是一个 B-Tree 组成的前缀树,使每个节点适配 cache line,利用 prefetch 加速,获得比较好的性能. 但 MassTree 在稳定性上依然有问题,是我们下一步的优化实现的方向.

2.2.3 并发控制

目前主要的 MVCC 技术有 2 种:

1) 基于两阶段锁(two phase lock). 事务开始时对数据集加锁,事务提交以后放锁. 不同的隔离级别,加锁类型和时机不同. 对于只读事务,为了提高性能,通常并不加锁,而是采用快照读(snapshot read)的方式,在事务开始时取得全局已提交事务的版本,在读的过程中通过可见性判断,读取这个版本之前的已提交数据. X-DB 对只读事务的可见性判断比较简单,存储的所有数据都是多版本的,每次写事务开始都会为每条数据分配一个唯一的递增的事务版本号,写事务提交之后更新全局事务号;每个只读事务开始前会读取当前全局的事务号,然后与查询到的数据进行版本号的比较,取小于等于读事务版本号并且在所有版本中最大的那一条记录.

2) 乐观并发控制(optimistic concurrency control, OCC). 事务开始之前计算事务涉及的数据集(WriteSet & ReadSet),事务执行的过程不加锁,提交前对 WriteSet 和 ReadSet 进行校验(Validate),如果有其他的事务对当前事务的数据集进行了更新,则回滚当前事务并进行重试,否则提交事务. OCC 适用于事务之间数据冲突很少的场景,在此种场景下事务的吞吐量会高于 Lock-Based MVCC,但在热点行冲突比较多的情况下,会产生大量的事务回滚重试,性能大大退化.

X-DB 同时支持 2 种并发控制技术,可以根据应用的特点进行选择,在热点冲突很少的应用场景,我们可以选用 OCC,反之在热点冲突明显的场景,则适用基于锁的并发控制.

2.2.4 混合存储格式

互联网庞大的数据量带来高昂的存储成本,因此对数据进行高效的压缩,提升数据存储密度成为一个重要议题;另外,数据库需要在各种负载上运行,传统的 OLTP/OLAP 的界限不再清晰,用户总是希望在一份数据上执行所有种类的请求,HTAP (hybrid transaction/analytical processing)应运而生.

X-DB 面向的主要业务场景是 OLTP,主要格式还是按行存储,不过为了求取更好的压缩比率,在局部使用了按列存储的格式,在这个基础上,X-DB 使用了基于列编码进行压缩的方式替代通用压缩算法进行压缩,是因为通用压缩算法并不关心数据的行列结构,一旦进行压缩以后,原有的结构是被完全破坏掉的,无法在压缩的数据上进行直接查询. 而 X-DB 的存储引擎存储了数据表的 schema,可以按 schema 解释数据的行列结构,识别每一列的数据类型和数据的特点寻求更合适的编码方式进行压缩,取得更好的压缩比,更关键的是,进行压缩编码后的数据仍然是有结构的,可以无需解压缩即可直接查询.

3 软硬一体设计

在本节中,我们详细介绍 X-DB 中软硬一体模块的设计和实现.

3.1 基于 FPGA/QAT 的数据 compaction 设计

X-DB 的存储结构都是基于 Copy-On-Write 实现的,所有的修改操作都是写入新版本增量数据,所以无论是写入数据还是修改元数据,都是可以在后台进行的,对读写操作没有影响. X-Engine 会存储数据的多个版本用于 MVCC,在 compaction 过程中会对不再被事务所引用的旧数据版本进行回收.

compaction 过程本身是非常占用系统资源的,需要对不同级别之间 key range 范围交叉的文件进行多路归并. 首先读取文件所有内容,解析所有的行,消耗大量 IO;然后进行大量的 key compare 操作,消耗 CPU; 最后写入结果,同样占用 IO 带宽. 在开启全速 compaction 的情况下,写入性能下降约 40%.

为了减少 compaction 对系统性能的冲击,X-DB 从多个方面对其进行优化,其中很重要的一点是利用 FPGA/QAT 对数据进行压缩和解压缩.

得益于新型存储设备的 IO 能力大幅上升,compaction 过程中对存储 IO 能力的占用问题得到一定缓解,但是对 CPU 资源的消耗对普通事务造成的影响愈发突出. 为了解决根本问题,X-DB 针对 compaction 完全异步化的特点,将 compaction 任务放到 FPGA/QAT 设备上执行的方案(如图 2 所示).

数据 compaction 的过程在本质上是一个多路归并操作,X-DB 将其抽象为一个多路 datablocks

按一定规则合并为一路 datablocks 的过程;FPGA 内部的流水线可以将数据的读取/解码/合并/编码同时执行,可以提供单路处理达 2 GBps 的性能,远优于 CPU 自己做 compaction 操作;同时可以将 compaction 中非常耗费 CPU 的通用压缩操作也一并完成,最终 CPU 在这个阶段只负责数据的 IO.

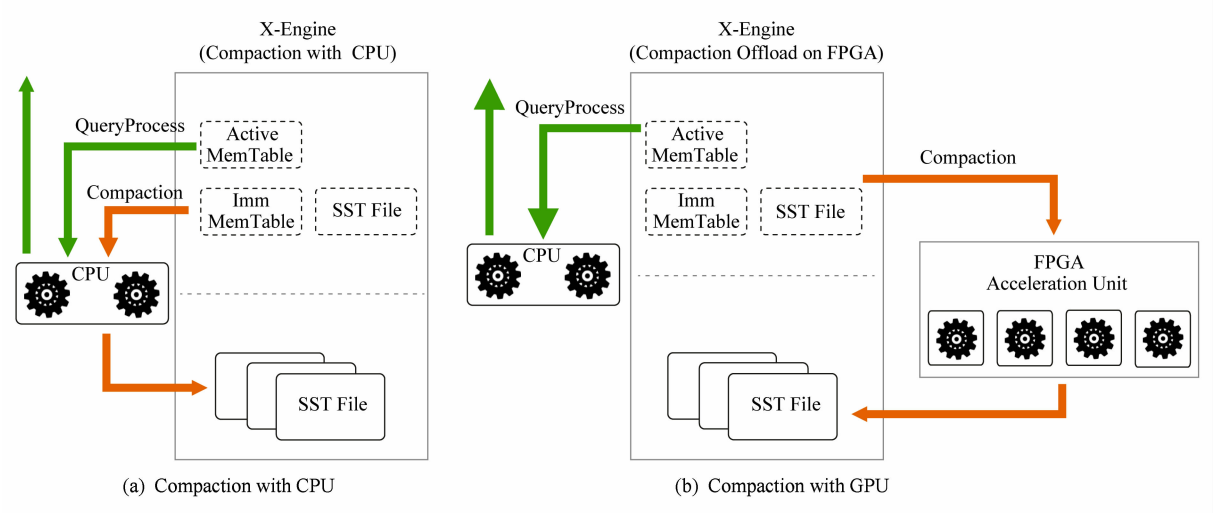


Fig. 2 Data compaction in X-DB

图 2 X-DB 数据 compaction

3.2 RDMA 协同下的数据传输方案

X-DB 拥有分布式强一致能力,内部通过 Paxos 算法实现多个数据库副本之间的数据强一致. 传统的 Paxos 依赖于操作系统内核协议栈的套接字实现数据通信. 在重负载情况下,多个副本之间的数据通信任务的并发和吞吐都非常大,不但占用了大量的系统资源,同时影响了数据库的响应时间.

为了提升数据传输的性能,同时降低其对系统资源的消耗,X-DB 从多个方面进行了优化,其中非常重要的一个方面是利用 RDMA(remote direct memory access)优化了日志写入和数据传输流程. RDMA 是一种远程数据直接存取技术,需要网络适配器以及其他网络基础设施的支持. 通过结合 RDMA 技术,数据库可以像访问本地内存一样访问远程节点的内存,避免了数据拷贝和上下文切换,整个传输过程由硬件实现,不需要 CPU 和操作系统参与.

X-DB 通过利用 RDMA 能大幅度降低系统资源消耗,同时降低了 Paxos 达成一致过程的时延,从而降低查询响应时间. Paxos 协议达成一致过程中的网路通信,可以抽象成将 Proposer 上的日志复制到法定多数派的节点上并持久化,同时将当前节点状态汇报给 Proposer. 其中最主要日志的复制过程需要经过内存拷贝、内核态协议栈组装/发送等过程,是当前的主要瓶颈. RDMA 的 One-Side API 中提供了 Remote Write 原语,可以不需要操作系统内核/CPU 的参与,将本地内存中的内容复制到远程内存中. 我们在 X-DB 中设计了高效的日志缓存数据结构,对 RDMA 支持友好;同时使用了 Remote Write 原语实现日志复制和消息汇报过程,不但降低了 CPU 负载,同时降低了时延,提升了数据库的响应速度.

3.3 X-DB 的存储引擎重构

基于阿里巴巴大规模的应用需求,我们观察到数据的冷热特性是非常明显的,而且与时间相关. 随着时间的流逝,越是久远的数据访问的频度越低. 因此,X-DB 的存储引擎中使用了分层存储的结构. 我们将整个存储体系划分为多个层级,每一级使用不同的存储介质,存储不同热度的数据,并且使用不同的存储格式. 访问热度比较低的数据,存储在慢速介质中,使用高压缩的方式;与此对应,访问热度较高的数据,存储在低延时高吞吐的存储介质中,采用高效的索引方式. 在设计中,我们首先将数据库所有的写入更新都以追加的方式写入内存,在内存中建立了一套高效的无锁索引结构来存储这些更新数据,这一层被认为是最热的数据,因为大部分应用都会倾向读最近写入的数据;内存中的数据也是分层的,越新近写入的数据,层级越高,内存不足时,将相对低层级的数据写出到下一级存储中,形成一个新的层级,我们称为 L_0,这一层级的数据仍然被认为是热度较高的数据,采用 NVM(non-volatile memory)的存储介质,L_0 层的数据充当了内存交换区(swap)

的角色,需要频繁读取,而且这一层数据也会逐渐变大,需要定期与更低层级的数据进行合并,选用NVM提供更低的读延时和更大的IO带宽.更低层级的数据属于相对热度比较冷的数据,这其中也分为若干层次(L_1/L_2),每一层都分割为固定大小的块(extent),每个层次的块大小不同,使用统一的索引结构.L_1以下所有层的数据会根据读写访问的频率进行分类识别,按extent为单位进行统计,较热的数据块会存放在SSD中,使用性能比较高的压缩算法(Snappy, LZ4, Zstd自动适配).而最冷的数据块会存放在HDD中,因为其极少被访问,使用按列编码,再进行高压缩率的压缩方式存储.采用分层存储结构,X-DB可以降低综合存储成本,并且保证性能不会下降.

4 实验与结果

在本节中,我们基于本文提出的设计和实现,给出了X-DB存储引擎的测试结果;同时,我们也给出基于QAT的数据压缩测试结果并加以分析.测试机型均为CPU: 64core, DRAM: 512 GB, Disk: NVME SSD 7T, NIC: 10Gb.

4.1 X-DB存储引擎

在本测试中,我们使用MySQL生态通用的测试集Sysbench[35].表数量设置为10,每个表包含20万条记录,并发连接数量为1 000,每个事务包含一条读或写操作.读写混合测试中,读写比为10∶4.

图3给出了Sysbench在吞吐率上的测试结果.Read-only方面,X-DB存储引擎对MySQL(InnoDB)有超过1倍的提升;Write-only上超过5倍的提升;读写混合下约有2.5倍的提升.相对于InnoDB,X-DB存储引擎将更新写入到高效内存索引中,没有基于页面的原地更新,只需要记录Redo日志,而无需Undo日志,大大简化了事务处理的逻辑;同时大部分热数据是在内存中,使用是无锁的数据结构,相对于页面扫描提取数据来说,在热点读性能上更为高效.该项测试是使用CPU进行compaction,并没有使用FPGA或QAT(QAT测试请见第4.2节).

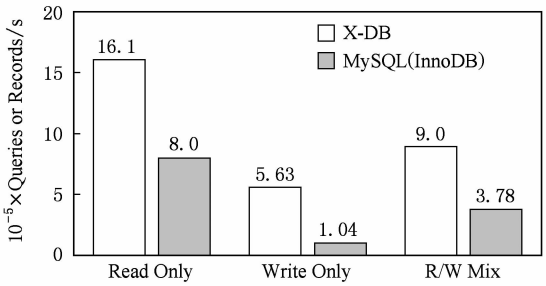


Fig. 3 Performance on Sysbench

图3 Sysbench测试结果

4.2 X-DB数据压缩

本节中我们使用X-DB存储引擎对QAT进行了测试.QAT是Intel的数据压缩技术,全称为QuickAssist Technology.测试过程为随机写入的同时开启数据压缩,观察硬件加速对数据压缩的性能提升.

测试使用64线程并发随机写入,key长度为8 B, value长度为240 B, 10张表按主键随机写入.写入数据为乱序,刷盘时压缩形成数据块.后台compaction线程将所有数据块读取出来解压并重新排序,然后压缩输出成全局有序的数据块.我们对比了使用QAT压缩和使用CPU压缩的性能(如图4所示).

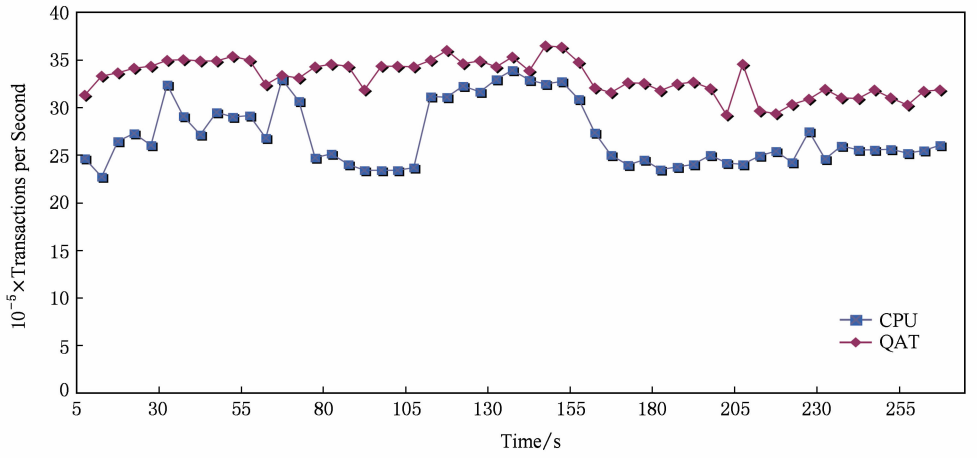


Fig. 4 compaction performance of QAT and CPU

图4 QAT和CPU压缩测试对比

可以看到,相对使用 CPU 压缩,QAT 能够获得平均 20%以上的性能提升.同时,使用 QAT 压缩时,服务器 CPU 平均使用率更低,写入性能更为平稳.基于 QAT 的测试结果,我们计划下一步将 compaction 任务放到 FPGA 上,目前该工作正在进行过程中.

5 总 结

本文介绍了阿里巴巴新一代数据系统 X-DB.阐述了 X-DB 的设计理念和关键模块的实现方法,特别针对软硬件协同设计进行了具体的分析和论述.实验结果表明,本文提出的设计思想以及软硬件一体化的设计对系统性能有较大提升,相对于 InnoDB,X-DB 存储引擎的读写有 2~5 倍的提升,使用 QAT 后,压缩速度有了 20%的提升.基于本文提出的设计思想,X-DB 将进一步融入新硬件特性,特别是在 NVM、多核(多 CPU)、RDMA 和 FPGA 方面进行更深入的探索,充分发挥新硬件特性,打造软硬一体协同工作的新一代数据库.目前该工作正在进行过程中.

致谢 感谢阿里巴巴集团张瑞和叶正盛对本文的指导!

参 考 文 献

[1] Codd E F. A relational model of data for large shared data banks [J]. Communications of the ACM, 1970, 13(6): 377-387

[2] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data [C] //Proc of the 7th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX, 2006: 15-15

[3] Apache. Apache Hbase [EB/OL]. [2017-11-15]. https://hbase.apache.org/

[4] Apache. Apache Cassandra [EB/OL]. [2017-03-01]. http://cassandra.apache.org/

[5] Sivasubramanian S. Amazon dynamoDB: A seamlessly scalable non-relational database service [C] //Proc of the 2012 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2012: 729-730

[6] MongoDB. MongoDB [EB/OL]. [2017-11-15]. https://www.mongodb.com/

[7] Harizopoulos S, Abadi D J, Madden S, et al. OLTP through the looking glass, and what we found there [C] //Proc of the 2008 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2008: 981-992

[8] Johnson R, Pandis I, Hardavellas N, et al. Shore-MT: A scalable storage manager for the multicore era [C] //Proc of the 12th Int Conf on Extending Database Technology: Advances in Database Technology. New York: ACM, 2009: 24-35

[9] Yu Xiangyao, Bezerra G, Pavlo A, et al. Staring into the abyss: An evaluation of concurrency control with one thousand cores [J]. Proceedings of the VLDB Endowment, 2014, 8(3): 209-220

[10] Jung H, Han H, Fekete A D, et al. A scalable lock manager for multicores [C] //Proc of ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2013: 73-84

[11] Porobic D, Pandis I, Branco M, et al. OLTP on hardware islands [J]. Proceedings of the VLDB Endowment, 2012, 5(11): 1447-1458

[12] Tu S, Zheng W, Kohler E, et al. Speedy transactions in multicore in-memory databases [C] //Proc of the 24th ACM Symp on Operating System. New York: ACM, 2013: 18-32

[13] Leis V, Boncz P, Kemper A, et al. Morsel-driven parallelism: A NUMA-aware query evaluation framework for the many-core age [C] //Proc of ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2014: 743-754

[14] Lang H, Leis V, Albutiu M, et al. Massively parallel NUMA-aware Hash joins [G] //LNCS 8921: Proc of IMDM 2013/2014. Berlin: Springer, 2015: 3-14

[15] Larson P A, Blanas S, Diaconu C, et al. High-performance concurrency control mechanisms for main-memory databases [J]. Proceedings of the VLDB, 2011, 5(4): 298-309

[16] Ren K, Thomson A, Abadi D J. Lightweight locking for main memory database systems [J]. Proceedings of the VLDB, 2012, 6(2): 145-156

[17] Arulraj J, Pavlo A, Dulloor S. Let's talk about storage & recovery methods for non-volatile memory database systems [C] //Proc of the 2015 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2015: 707-722

[18] Arulraj J, Perron M, Pavlo A. Write-behind logging [J]. Proceedings of the VLDB Endowment, 2016, 10(4): 337-348

[19] Arulraj J, Pavlo A. How to build a non-volatile memory database management system [C] //Proc of the 2017 ACM Int Conf on Management of Data. New York: ACM, 2017: 1753-1758

[20] Chen Shimin, Jin Qin, Persistent B+-trees in non-volatile main memory [J]. Proceedings of the VLDB Endowment, 2015, 8(7): 786-797

[21] Kalia A, Kaminsky M, Andersen D. Using RDMA efficiently for key-value services [C] //Proc of ACM SIGCOMM 2014. New York: ACM, 2014: 17-22

[22] Huang J, Ouyang X, Jose J, et al. High-performance design of HBase with RDMA over InfiniBand [C] //Proc of the 26th IEEE Int Parallel and Distributed Processing Symp. Los Alamitos, CA: IEEE Computer Society, 2012: 774-785

[23] Lu X, Islam N S, Rahman M, et al. High-performance design of Hadoop RPC with RDMA over InfiniBand [C] // Proc of the 42nd Int Conf on Parallel Processing. Los Alamitos, CA: IEEE Computer Society, 2013: 641-650

[24] Mitchell C, Geng Y, Li J. Using one-sided RDMA reads to build a fast, cpu-efficient key-value store [C] //Proc of the 2013 USENIX Conf on Annual Technical Conf. Berkeley, CA: USENIX, 2013: 103-114

[25] Kallman R, Kimura H, Natkins J, et al. H-Store: A high-performance, distributed main memory transaction processing system [J]. Proceedings of the VLDB Endowment, 2008, 1(2): 1496-1499

[26] VoltDB. VoltDB [EB/OL]. [2017-03-01]. http://voltdb.com

[27] Neumann T, Mühlbauer T, Kemper A. Fast serializable multi-version concurrency control for main-memory database systems [C] //Proc of ACM SIGMOD Int Conf on Management. New York: ACM, 2015: 677-689

[28] Neumann T. Efficiently compiling efficient query plans for modern hardware [J]. Proceedings of the VLDB Endowment, 2012, 4(9): 539-550

[29] Franz F, Sang K C, Jürgen P, et al. SAP HANA database: Data management for modern business applications [J]. ACM SIGMOD Record, 2011, 40(4): 45-51

[30] Diaconu C, Freedman C, Ismert E, et al. Hekaton: SQL Server's memory-optimized OLTP engine [C] //Proc of the 2013 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2013: 1243-1254

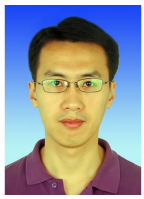
[31] Bacon D F, Bales N, Bruno N, et al. Spanner: Becoming a SQL system [C] //Proc of the 2017 ACM Int Conf on Management of Data. New York: ACM, 2017: 331-343

[32] O'Neil P, Cheng E, Gawlick, D, et al. The Log-structured Merge-tree (LSM-tree)[J]. Acta Informatica, 1996, 33(4): 351-385

[33] Mao Yandong, Kohler E, Morris R, et al. Cache craftiness for fast multicore key-value storage [C] //Proce of the 7th ACM European Conf on Computer Systems. New York: ACM, 2012: 183-196

[34] Levandoski J J, Lomet D B, Sengupta S. The Bw-Tree: A B-tree for new hardware platforms [C] //Proc of the 2013 IEEE Int Conf on Data Engineering (ICDE 2013). Los Alamitos, CA: IEEE Computer Society, 2013: 302-313

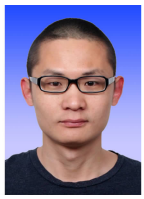
[35] GitHub. Sysbench [EB/OL]. [2017-10-08]. https://github.com/akopytov/sysbench



Zhang Tieying, born in 1982. Received his PhD degree from Chinese Academy of Sciences. Staff engineer in Alibaba Group. His main research interests include database system and distributed system.

Huang Gui, born in 1981. Received his bachelor degree from Taiyuan University of Technology. Database kernel developer in Alibaba Group. His main research interests is distributed database.

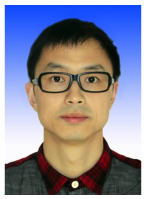
Zhang Yingqiang, born in 1987. Received his master degree from Zhejiang University. Database kernel engineer in Alibaba Group. His main research interests include database system and consensus algorithms.



Wang Jianying, born in 1986. Received his master degree from Zhejiang University. Database kernel engineer in Alibaba Group. His main research interests include relational database and distributed system.

Hu Wei, born in 1987. Received his master degree of computer science from Zhejiang University. Database kernel engineer in Alibaba Group. His main research interests include relational database and NewSQL systems.

Zhao Diankui, born in 1981. Received his master degree of computer science from Dalian University of Technology. Database kernel engineer in Alibaba Group. His main research interests include relational database and MPP database.



He Dengcheng, born in 1983. Received his master degree from Zhejiang University. Director of Alibaba Database Kernel. His main research interests include building self-maintained distributed database system.