文章编号:1000-5641(2014)05-0141-08

OceanBase 关系数据库架构

阳振坤
(阿里巴巴集团,北京 100020)

摘要: 传统关系数据库本质上是单机系统,通常采用昂贵的高端服务器和高端存储,难以应对互联网应用的高可扩展、高性能、高可用和低成本的挑战. OceanBase 是阿里巴巴研制的开源分布式无共享关系数据库,采用主流通用 PC 服务器,很好地满足了互联网对关系数据库的需求. OceanBase 已经用于淘宝、天猫和支付宝的多个生产系统. 本文介绍了 OceanBase 关系数据库系统的架构、目标和系统特点,特别分析了基于该系统架构的读写事务流程.

关键词: OceanBase; 关系数据库; 分布式系统; 事务; 互联网
中图分类号: TP31 文献标识码: A DOI:10.3969/j.issn.1000-5641.2014.05.012

The architecture of OceanBase relational database system

YANG Zhen-kun
(*Alibaba Inc.*, *Beijing* 100020, *China*)

Abstract: Traditional RDBMS is essentially a single-machine system and usually employs high-end server and high-reliable storage device due to performance and reliability issues which make it incompetent to serve today's Internet applications with high scalability, high performance, high available and low cost. OceanBase is an open-source, shared-nothing, distributed relational database system, which is developed from scratch at Alibaba. Built on top of computer clusters consisting of in-expensive commodity PC servers, OceanBase meets the requirement of modern internet applications quite well. OceanBase has been widely used in the production systems of Taobao, Tmall and Alipay for years.

Key words: relational database system; distributed system; transaction; Internet

0 引 言

在过去半个世纪左右的时间里,数据库系统从无到有,经历了层次数据库、网状数据库和关系数据库等几个发展阶段. 时至今日,主要数据库系统基本都是关系数据库,并广泛应用于银行、信用卡交易、商品销售、航空与铁路运输、电信、电力系统、教育、医疗与健康、电子商务等领域,成为了信息社会最关键的基础设施之一. 关系数据库系统也成为了最稳定可靠、最核心的系统之一.

然而,在互联网高速发展的今天,众多知名的互联网公司,例如 Google、Facebook、Am-

收稿日期:2014-06

作者简介:阳振坤,男,阿里巴巴集团技术专家,研究方向为分布式系统与数据库.
E-mail: zhengxiang@alibaba-inc.com.

azon、腾讯、阿里巴巴、百度、雅虎等,并没有在他们的业务系统中采用商业关系数据库(eBay 旗下的 PayPal 除外). 难道互联网公司不需要关系数据库吗?恰好相反,互联网公司不仅需要关系数据库,而且对关系数据库的伸缩性、高性能、高可用和低成本有更高的要求,传统的关系数据库难以满足这些需求.

阿里巴巴的淘宝和天猫是网上购物平台,在 2013 年 11 月 11 日(双 11)创造了单日 350.19 亿元人民币的交易额[1],同一天支付宝则完成了 1.88 亿笔支付,高峰时达到 79 万笔/min[2],都对传统关系数据库提出了很强的挑战. 过去 4 年时间里,阿里巴巴 OceanBase 团队研制了 OceanBase[3] 开源 分布式无共享关系数据库,以"OceanBase 数据库+主流 PC 服务器"取代"商业数据库+高端服务器+高端存储",广泛应用于淘宝、天猫和支付宝线上业务,极大地降低了软件和设备成本,良好的伸缩性和自动的故障恢复不仅很好地支撑了业务,而且大大地降低了运行维护的人力成本.

本文的结构如下:第 1 节介绍了互联网应用的特点,分析了互联网新型应用对传统数据库提出的挑战;第 2 节给出了 Oceanbase 数据库的系统架构、数据库事务等核心技术;第 3 节介绍了相关工作,对比分析了 Oceanbase 数据库的特点;第 4 节对全文进行了总结.

1 互联网时代的关系数据库

互联网公司的业务不仅涉及大量非结构化的数据,例如网站访问日志等,也有很多涉及结构化数据的业务,例如 Google 的搜索广告、淘宝/天猫的商品搜索广告的计费,淘宝/天猫、Amazon 和 eBay 等的网上和移动购物,支付宝的网上和移动支付等等,都是商务交易和金融交易,因此数据库的事务(Transaction)功能不可或缺. 事务需要满足 ACID 原则:原子性(Atomicy)、一致性(Consistency)、隔离性(Isolation)和持久性(Durability). 然而,绝大多数大型互联网公司(PayPal 除外)都没有在业务上使用商业数据库,因为互联网的应用场景跟传统业务的应用场景有了很大的不同. 这些不同主要体现在以下四个方面.

(1) 互联网应用需要数据库的并发处理能力是传统应用的几十倍甚至几百倍. 传统应用领域的数据库系统,通常只有少数人和设备并发操作数据库,例如银行的职员及 ATM 机、商场的收银员、飞机票火车票的售票员等,后台的数据库系统常常只有几百和几千的并发访问,几万的并发访问量很少见,几十万以上的并发几乎见不到. 在草根文化大行其道的互联网,比如网上购物和网上购票等,可能每个网民都是收银员或售货/售票员,对于后台数据库,几万和几十万的并发随时可见,特殊情况下,比如促销期间和春节期间,几百万的并发都很常见,2013 年阿里巴巴天猫双 11 大促(11 月 11 日),同时在线人数达到 1700 万,是整个香港人口的 2.5 倍[4].

(2) 互联网应用需要的数据库伸缩性也大大超越了传统应用. 传统业务的增长,通常都比较平稳,商场、银行网点、售票点的新建/扩建等,通常需要几个月甚至几年的周期;而互联网的促销,可能在一天之内增加/减少几倍、几十倍的访问量,例如 2013 年双 11 大促一天成交金额 350.19 亿元人民币,大约是全年日平均交易额的 8.5 倍[5]. 再比如,"愤怒的小鸟"游戏在 2012 年圣诞节一天下载量超过 800 万[6]. 这都要求数据库有几倍、几十倍的伸缩能力,并且这种伸缩常常要在若干天甚至若干小时之内完成.

(3) 互联网应用对数据库的性能价格比的要求也十分"苛刻". 在传统应用领域,数据库系统处在最核心的位置,必须有极高的稳定性和可靠性,为达到这一目的可能"不计成本"

（例如采用高度稳定可靠的服务器和存储设备，使用成熟的商业数据库软件等），成本高昂. 互联网应用需要的数据库并发能力常常是传统应用的几十倍、几百倍，如果性价比保持不变，则互联网公司需要传统应用几十倍、几百倍的数据库成本.

（4）互联网应用在数据库可用性上面临的挑战也跟传统应用有很大的不同. 不仅因为互联网应用的访问量大、数据库故障影响的人群数量更大，而且由于控制成本的原因，互联网公司的数据库使用廉价的主流 PC 服务器，其故障率也明显高于高端服务器和高可靠存储，因此面临的可用性挑战更大.

为了解决上述问题，针对互联网应用的高并发、强伸缩性、低成本、高可用等方面的挑战，设计实现了针对新型互联网应用的 Oceanbase 关系数据库系统.

2　OceanBase 架构

2.1　系统需求及目标

在设计和开发的过程中，OceanBase 定位于通用的数据库，并且需要满足互联网应用对数据库的伸缩性、高性能、高可用和低成本的要求.

基于这一总体需求，OceanBase 的设计目标如下.

（1）事务的 ACID 原则. 事务是关系数据库的核心技术. 事务的 ACID 原则是保障淘宝和天猫等商业交易业务和支付宝等金融交易业务的关键，因此 OceanBase 把事务作为首要的、不可妥协的目标. 此外，为了降低业务的学习以及迁移成本，OceanBase 并没有定义自己的客户端，而是使用了 MySQL 客户端.

（2）低成本. 低成本是 OceanBase 的重要目标. OceanBase 使用廉价 PC 服务器及其磁盘（主要是固态盘）代替了传统数据库使用的高端服务器和高端存储.

（3）高性能. 读写事务性能都是数据库的主要指标. OceanBase 需要为业务提供很高的读写事务性能. 与读事务可通过缓存（cache）提升性能相比，写事务性能的提升更加挑战，因此 OceanBase 在设计实现上偏重于写事务的性能，并消除了磁盘的随机写，以充分利用固态盘良好的随机读性能.

（4）在线伸缩. 在线的、按“天”甚至按“小时”级别的伸缩是互联网业务的关键需求. OceanBase 采用了分布式架构，可随时增加/减少服务器，并自动进行数据迁移和负载均衡.

（5）在线故障恢复. 传统数据库使用了高端服务器和高端存储，这些设备有相当高的可靠性，设备故障被认为是罕见的. OceanBase 使用了廉价 PC 服务器及其磁盘，硬件（特别是磁盘，包括固态盘）故障经常发生，因此 OceanBase 需要能够随时进行自动、快速的故障恢复，以免设备故障影响业务.

2.2　系统假设

与数据库的高性能、高并发相伴的常常是庞大数据量. 调研和分析发现，许多数据库中的数据量虽然很大，但一段时间（例如一天）内增删改的数据总量通常不大，每条记录的修改往往在几十到几百字节（通常明显低于单条记录的尺寸）. 例如在以下三个示例数据库中，一段时间内的数据增删改影响的记录数占整个数据库记录数的比例都很小.

（1）人口数据库. 人口数据库保存了每个人的姓名、身份证号、户口、婚姻、住址等信息，假如每人一条记录，全国就有 14 亿条记录. 但每天修改的记录并不多，例如人口出生（每天几万人）、死亡（每天几万人）、修改姓名、户口迁入迁出等，这些增删改记录数跟总记录数相

比,只占一个很小的比例.

(2) 交易数据库. 每个交易对应一条(或几条)记录,线上数据库通常保存一年或更长时间的交易记录,但每天新增的记录并不多(相当于 1 年的 1/365),每天修改的记录(例如支付宝的一笔交易由"买家已付款"修改为"卖家已发货"等),也只占几类总数的一个很小的比例.

(3) 账务数据库. 账务记录了每个用户的一个或多个账户信息. 当新建账户、注销账户、发生付款或收款时,需要修改对应的信息,尽管账务数据库可能有几千万条甚至几亿条记录,但每天修改的记录数,同样只占一个很小的比例.

设想一个系统一天有 10 亿笔写事务,每笔信息为 100-Byte,那么一天的信息量即为 10 亿 $\times$100=100 GB,这已经是当前单个 PC 服务器内存可以达到的容量. 因此尽管数据库记录总数可能很大,但一天内的增删改记录数,只占很小的比例,如图 1 所示.

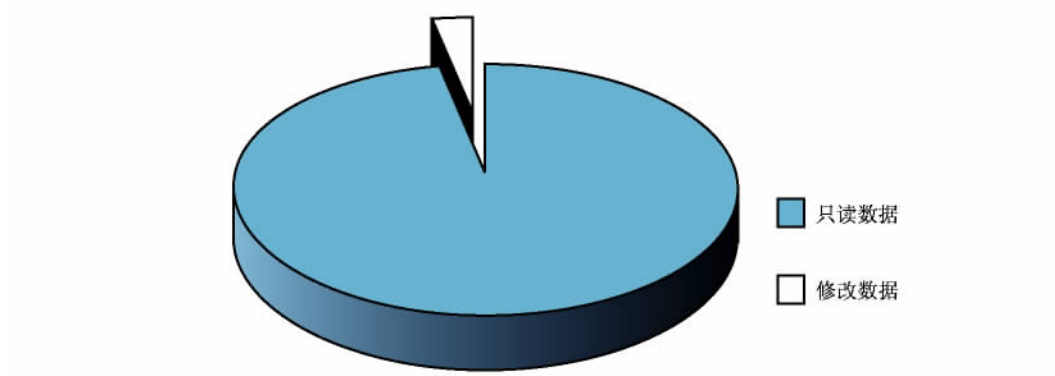

图 1 数据库单日修改数据比例示意图

Fig. 1 Schematic diagram of daily mutated data portion in a database

2.3 系统架构

基于第 2.2 节的分析,由于大多数数据库一天的增删改数据量相对于数据库的记录数比例较小,因此 OceanBase 以增量方式把当天增删改的数据保存在服务器内存中(Redo log 保存在磁盘),称为 MemTable;对应的基线数据(即数据库在 MemTable 开始时刻的快照)则分割后保存在磁盘(通常是固态盘)上,称为 Sstable,如图 2 所示.

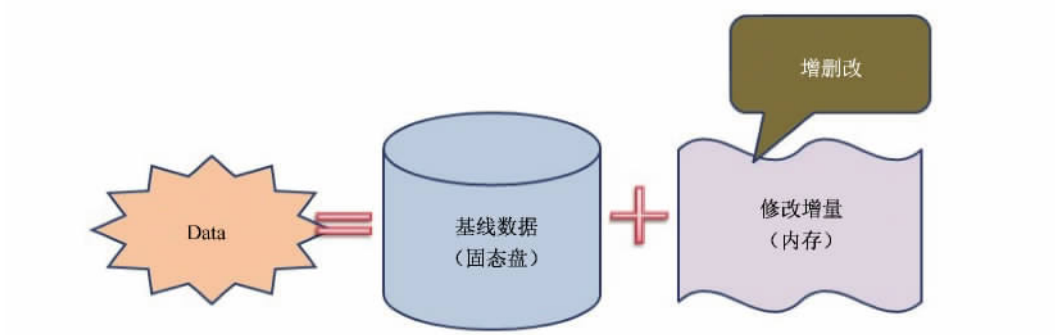

图 2 OceanBase 数据模型

Fig. 2 Data model of OceanBase

由于增删改数据 MemTable 通常较少，所以 MemTable 通常保存在服务器（称为 UpdateServer）的内存，以增量方式保存，多次修改之间以指针连接；基线数据 Sstable 通常保存在多台服务器（称为 ChunkServer）的磁盘，以数据页（page，例如 8KB）方式存储和访问. 图 3 所示是单机群 OceanBase 的架构图.

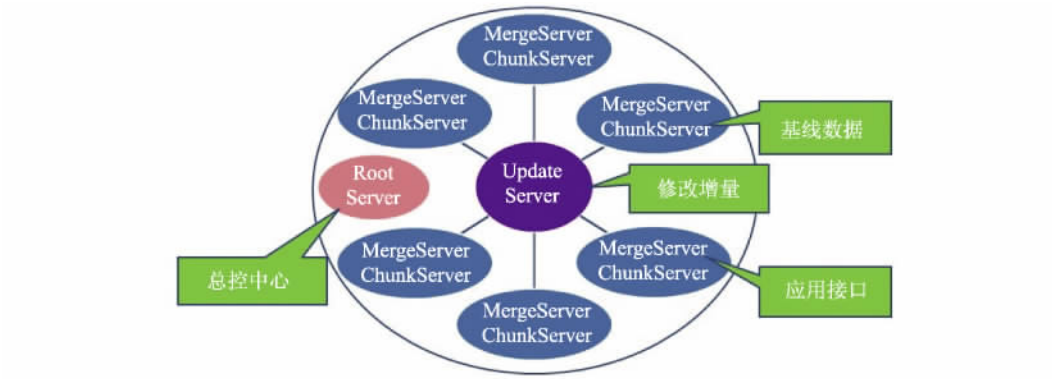

图 3　OceanBase 架构（单机群）

Fig. 3　OceanBase architecture (single cluster)

OceanBase 系统架构中，包括四类服务器：主控服务器 RootServer、基线数据服务器 ChunkServer、合并服务器 MergeServer，以及更新服务器 UpdateServer.

（1）主控服务器 RootServer. 该服务器是 OceanBase 的总控中心，负责 ChunkServer/MergeServer 的上线、下线管理以及 Sstable 的负载均衡.

（2）基线数据服务器 ChunkServer. ChunkServer 保存基线数据 Sstable 并提供读访问. 为了防止由于 ChunkServer 故障导致服务不可用甚至数据丢失，每个 Sstable 保存了多个副本并分布在不同的 ChunkServer 上.

（3）合并服务器 MergeServer. OceanBase 的应用接口，支持 JDBC/ODBC 协议，接收并解析用户的 SQL 请求，经过词法分析、语法分析、生成执行计划等一系列操作后，发送到相应的 ChunkServer.

（4）更新服务器 UpdateServer. 执行写事务、保存修改增量（到内存）、写 Redo log（到磁盘）并提供读服务. UpdateServer 通常也有多个副本.

由于 UpdateServer 内存是有限的，修改后的增量无法一直保存在内存中，因此 OceanBase 通常每天在某个时刻（例如后半夜业务低谷期）冻结当前 MemTable 并开启新的 MemTable；此后新的增删改写入新的 MemTable，然后系统在后台把冻结的 MemTable 与当前基线数据融合，生成新的基线数据，这个过程称为每日合并. 每日合并完成后，冻结的 MemTable 以及旧的 Sstable 即可释放，其占用的内存和磁盘空间也被回收.

为了防止 UpdateServer 故障或者机房故障导致服务不可用甚至数据丢失，OceanBase 通常部署多个机群（例如一主两备），主机群执行写事务并且至少同步到一个或多个备用机群（超过半数[7]）. 这样任何一个机群异常都不会导致服务不可用，从而保证了数据库的高可用性. 其架构如图 4 所示.

传统关系数据库通常采用主备库镜像，主库备库之间的网络异常以及备库异常都可能导致主库备库不同步；OceanBase 的多机群（≥3）方式使得不仅单个备机群异常不会影响业

务,而且使得,即使主机群突然故障,数据库系统也会在最多若干秒后自动恢复服务,不会造成任何数据丢失.

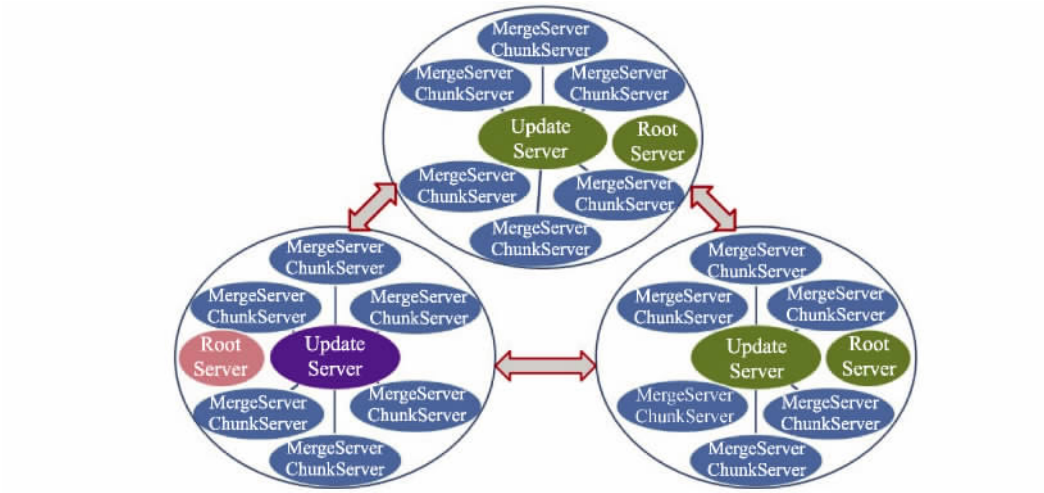

图 4 OceanBase 架构(三机群)

Fig. 4 OceanBase architecture (triple clusters)

2.4 读事务

对于用户的读取任务,MergeServer 接受 SQL 请求并解析生成 SQL 执行计划. MergeServer 确定事务中数据的基线数据在哪些 ChunkServer 上,然后通知这些 ChunkServer 执行对应的读请求. 由于基线数据与修改增量的分离,如图 2 所示,读事务需要融合基线数据和修改的更新数据,返回满足查询条件的数据.

一个特殊的情况是在每日合并期间,让所有 ChunkServer 同步完成每日合并或者同步启用新的 Sstable 可能非常复杂甚至无法完成(比如某个 ChunkServer 网络异常),因此某些 ChunkServer 的部分 Sstable 完成了与冻结的 MemTable 的融合,生成了新的 Sstable,其他 Sstable 尚未完成融合,有些查询可能用到旧的 Sstable,有些查询可能用到新的 Sstable,还有的查询可能同时用到旧的 Sstable 和新的 Sstable. 相同内容的 SQL 语句,到达的 MergeServer 不同或到达时间不同,使用新旧 Sstable 的情况可能有所不同,那么查询的结果是否一致呢?答案是肯定的. 因为查询使用旧基线数据时,会融合旧增量数据(即冻结的 MemTable)和新增量数据(即新的 MemTable),查询使用新基线数据时,则只融合新增量数据(新的 MemTable),因此得到的结果是一致的,如图 5 所示.

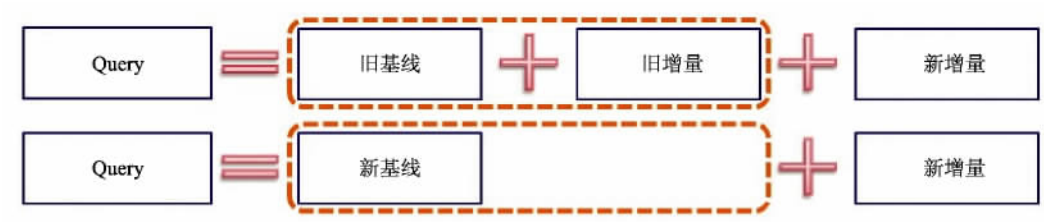

图 5 OceanBase 每日合并期间的查询

Fig. 5 OceanBase query during daily merging

尽管使用了与传统关系数据库相似的方式存储，Sstable 的随机读并没有成为 OceanBase 的瓶颈，这得益于 OceanBase 通常采用固态盘作为存储并且没有随机磁盘写，一个没有随机磁盘写的固态盘可以提供每秒几万次的随机读（一个机械盘每秒只能提供几百次的随机读）.

传统的基于磁盘的关系数据库，其读事务操作的基本步骤是先从磁盘中读出数据页（page），再从中取出需要的内容，然后根据用户的 SQL 进行操作得到结果. OceanBase 的读事务操作与它们类似，不同的是 OceanBase 从读出的数据页中取出需要的内容时，还得与对应的修改增量融合. 由于被修改数据所占的比例比较小、修改增量以及融合操作都在内存中，这个操作对性能的损耗很小. 与此同时，由于使用固态盘并且没有随机写，OceanBase 能够充分利用固态盘优异的随机读性能，因此能够获得很好的读性能.

2.5　写事务

MergeServer 解析 SQL 请求生成 SQL 执行计划后，如果不是只读事务，则向 ChunkServer 获取对应的基线数据，然后连同执行计划提交给主机群 UpdateServer 执行. 主机群 UpdateServer 执行写事务，生成 Redo log，同步给备机群并持久化到各自磁盘，如果成功者（包括自己）超过了半数，则在 MemTable 中提交该事务并应答客户.

尽管有大量的随机访问，由于增删改的修改增量放置在内存，Redo log 是顺序写，所以 OceanBase 系统中没有随机写磁盘. 不仅如此，OceanBase 的 UpdateServer 服务器通常配置带电池或电容的 RAID 卡，这种 RAID 带有内存（例如 1GB）并且在服务器断电时能够保持其中的数据不丢失，当小块的 Redo log 写入磁盘时，其实只是写到了 RAID 卡的内存中，该 RAID 卡稍后批量把其内存中数据写到磁盘上，这样不仅缩短了日志刷盘的时间（大约 0.1 ms），而且降低了小块 Redo log 的写入对固态盘的性能和寿命的影响（因为固态盘的写入也是以页（page）为单位的，例如 4KB，并且写入前需要先擦除，在已有页上即使追加一个字节也需要把原有页读出来，与追加字节合并，然后写入一个新的页）. 每日合并期间把修改增量与旧基线数据合并生成新基线数据时，对磁盘的访问是顺序读和顺序写，当施加一定的流量控制后，这种顺序读和顺序写对磁盘的随机读的影响也是可控的.

传统的基于磁盘的关系数据库，其写事务操作的基本步骤是从磁盘中读出数据页（page），再从中取出需要的内容，然后根据用户的 SQL 进行修改，再把修改后的结果与原数据页融合生成新的数据页，写日志（Redo log、Undo log）并把新的数据页刷到磁盘，由于每次修改通常在几十到几百字节，而数据页的大小通常在几 KB（例如 8KB），这就出现了明显的写放大（8KB/100≈80X）. OceanBase 的写事务操作也是先从磁盘中读出数据页，再从中取出需要的内容，然后根据用户的 SQL 进行修改操作以生成增删改的增量，写日志（Redo log）并且把修改增量加入到 MemTable.

由于内存中修改增量没有也不需要做成数据页，因此 OceanBase 不仅省去了写新的数据页到磁盘的操作（随机写磁盘），也避免了传统数据库的写入放大（每日合并通常在系统低谷期间进行，对业务影响很小，更不会影响高峰期的业务）. 这使得 OceanBase 在写事务性能上有明显的优势.

3　OceanBase 的特点分析

目前，国内外的学术界和工业界在支撑互联网应用的数据库产品方面做了大量努力.

Google 的 Bigtable[8] 是一个基于 Google File System[9] 的分布式表格系统,只支持单行事务. Google 的 Spanner[10] 实现了跨越全球的分布式事务,但复杂 SQL 性能较低,且对时钟有很强的依赖. OceanBase 系统是在保证"强一致性"的前提下,追求系统的"最大可用性",实现面向互联网业务需求的强伸缩性、高性能、高可用和低成本的通用数据库.

根据数据存储方式的不同,我们可以把目前的数据库分为三类:外存型数据库、全内存数据库以及内外存混合型数据库.

(1) 外存型数据库. 这种类型的数据库的读写事务都基于外存(磁盘),内存作为缓存(Cache),数据的存储基于数据页(page),读和写都有一定的放大效应,写入放大和磁盘随机写性能限制了数据库的性能. 多数传统的关系数据库都属于这个类型.

(2) 全内存数据库. 这种类型的数据库数据全部在内存中(log 除外),没有数据页的概念,数据读写都在内存中进行(写日志除外),性能很高. VoltDB(http://voltdb.com/)和 MemSQL(http://www.memsql.com/)属于这个类型,并且都号称是全世界最快的内存数据库.

(3) 内外存混合型的数据库. 这种类型的数据库部分数据在外存(磁盘),部分数据在内存,在磁盘上的数据以数据页为单位存储,内存中的数据不使用数据页,内存也不仅仅作为缓存. OceanBase 属于这个类型,其基线数据以数据页方式保存在磁盘上,而增删改的增量则在内存中,性能介于磁盘型的数据库与全内存数据库之间.

从存储方式来看,OceanBase 属于内外存混合型数据库;与内存数据库相比,OceanBase 在读写事务性能上的劣势不大,但在经济性上优势十分明显,具体在以下几方面.

(1) 读事务性能. 内存数据库完全避免了磁盘 I/O(写日志除外)并且也没有数据页导致的读放大. 然而,由于固态盘的随机读响应时间(0.1 ms)仅为机械盘(3～5 ms)的几十分之一,并且由于 80/20 法则,OceanBase 可以用相对较小的内存代价(比如全量数据的 20%)缓存少部分比较热的数据,再加上 query cache,因此读事务并不会成为 OceanBase 的性能瓶颈.

(2) 写事务性能. OceanBase 的写事务也是内存操作,与全内存数据库类似,其中的读操作,由于 SSD 优异的随机读性能以及缓存(80/20 法则),因此 OceanBase 与全内存数据库的写性能的差距不大.

(3) 经济性. 80/20 法则表明,一段时间(例如小时或者天)内数据库中的数据只有小部分会被频繁访问,大部分数据被很少访问或者根本没有被访问. 把这些很少或没有被访问的数据与频繁访问的数据同等对待并一样占用昂贵的内存资源,显然是不经济的. 由于固态盘容量大大高于内存容量,同等尺寸的数据量,内存数据库需要的服务器数量也比内外存混合型的 OceanBase 数据库需要的服务器数量多得多.

4 结 论

针对互联网应用对关系数据库的高可扩展、高性能、高可用和低成本的需求,本文阐述了传统关系数据库的不足,给出了 OceanBase 关系数据库的架构,对 OceanBase 的读写事务事务及其性能进行了分析,并对比分析了 OceanBase 与传统关系数据库以及全内存数据库的优劣.

(下转第 163 页)

[3] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4.
[4] RODEH O. B-trees, shadowing, and clones[J]. ACM Transactions on Storage (TOS), 2008, 3(4): 2.
[5] MICHAEL M M. Hazard pointers: Safe memory reclamation for lock-free objects[J]. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(6): 491-504.
[6] Oracle. Data Concurrency and Consistency[EB/OL]. [2014-07-31]. http://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm#i17841.
[7] Oracle. Transaction Set Consistency [EB/OL]. [2014-07-31]. http://docs.oracle.com/cd/B28359_01/server.111/b28318/consist.htm#CNCPT1322.
[8] **何登成**. InnoDB **事务/锁/多版本 实现分析**[EB/OL]. [2014-07-31]. http://hedengcheng.com/?p=286.
[9] LEWIS J. Transactions and Consistency[M]//Oracle Core. New York: Apress, 2011: 25-58.
[10] JOHNSON R, PANDIS I, STOICA R, et al. Aether: a scalable approach to logging[J]. Proceedings of the VLDB Endowment, 2010, 3(1): 681-692.

(责任编辑　赵　伟)

(**上接第** 148 **页**)

[参　考　文　献]

[1] **天猫微博**[EB/OL]. http://weibo.com/1768198384/AiigJrzYT?mod=weibotime.
[2] **支付宝微博**[EB/OL]. http://weibo.com/1627897870/AiiuiseVH?mod=weibotime.
[3] OceanBase **开源**[EB/OL]. http://alibaba.github.io/oceanbase/.
[4] **天猫微博**. http://weibo.com/1768198384/Aie2CyONt?mod=weibotime#_rnd1404271771131.
[5] **阿里巴巴招股书**[EB/OL]. 2014-06-17. http://www.sec.gov/Archives/edgar/data/1577552/000119312514236860/d709111df1a.htm.
[6] Angry Birds Racks Up 8 Million Downloads in One Day[EB/OL]. http://www.forbes.com/sites/davidthier/2013/01/04/angry-birds-racks-up-8-million-downloads-in-one-day/.
[7] LAMPORT, L. The part-time parliament[J]. ACM TOCS, 1998, 16(2): 133-169.
[8] CHANG F, DEAN J, GHEMAWAT S, et al. Bigtable: A distributed storage system for structured data[J]. OSDI, 2006: 205-218.
[9] GHEMAWAT S, GOBIOFF H, LEUNG, et al. The Google file system[R]. ACM SOSP, 2003: 29-43.
[10] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally-distributed database[C]. OSDI, 2012: 251-264.

(责任编辑　李　艺)