

数据库技术发展研究综述

侯博学, 陈 林

(上海第二工业大学 计算机与信息工程学院, 上海 201209)

摘 要: 数据库是一种用于存储、组织、检索和管理数据的系统, 其历史可以追溯到 20 世纪 60 年代中期, 当时第一个数据库管理系统(DBMS)诞生, 主要用于解决大型机器的资源共享问题。随着计算机技术的发展, 数据库技术也不断演进, 其应用领域已经涵盖了几乎所有行业, 并且在不断扩大。按照时序总结了数据库领域的常见技术和数据库种类, 并介绍了人工智能与数据库相结合的发展方向。

关键词: 数据库; 数据存储; 海量数据; 机器学习

DOI: 10.11907/rjdk.231396

中图分类号: TP311.1

文献标识码: A

开放科学(资源服务)标识码(OSID):

文章编号: 1672-7800(2024)006-0214-07



Research Overview of Database Technology Development

HOU Boxue, CHEN Lin

(College of Computer and Information Engineering, Shanghai Polytechnic University, Shanghai 201209, China)

Abstract: Database is a system used to store, organize, retrieve and manage data. Its history can be traced back to the mid-1960s, when the first database management system (DBMS) was born. It was mainly used to solve the resource sharing problems of large machines. With the development of computer technology, database technology has also continued to evolve. Today, database technology has covered almost all industries and continues to expand. This article summarizes common technologies and various types of databases adopted in the field of databases chronologically, and explores the current direction of integrating artificial intelligence with databases.

Key Words: database; data storage; massive data; machine learning

0 引言

在数据存储的发展过程中, 如何高效地管理查询数据始终是一个难题。随着时代的发展, 数据管理也由手工管理阶段过渡到计算机管理阶段, 特别是在当今大数据时代, 更是需要有高性能的数据库解决方案。数据库中的数据按照编码方式编码、组织, 具有冗余度小、易共享和独立性等特点。数据库在企业技术选型与应用中始终是核心组件。

数据库企业级应用产品诞生于 1970 年前后, 在 50 余年的发展过程中, 数据库领域的产品十分多样。关系模型数据库的发展以关系模型概念的提出和四大范式理论的论述作为开端。此阶段具有代表性的关系型数据库有 Sybase、Oracle、SQL Server、MySQL、PostgreSQL。就目前而

言, 大型开源关系型数据库应用较广的有 MySQL 和 PostgreSQL。这些开源的数据库产品推动了信息化时代的变革, 也给后续数据库的发展提供了思路。各大厂商对其进行定制化二次开发, 以满足特定的性能需求。Google 的三驾马车 (MapReduce、GFS、BigTable) 开启了非关系型数据库的时代, 该阶段数据量庞大、数据类型丰富, 非结构化数据占据主要部分。随着 GFS 与 BigTable 的问世, 大数据时代的数据存储有了解决方案^[1]。

目前, 单机存储在海量数据面前不堪一击, 众多数据库产品在发展过程中逐渐支持以集群方式部署, 例如: MySQL 主从架构——Redis 的插件化集群支持。当然, 部分数据库在设计时就是为海量数据而准备。Google 的主要业务是搜索引擎, 因此 Google 需要大量的爬虫以获取页面, 并将其存储至 Google File System (GFS) 中。为了应对不断增加的页面, GFS 在设计之初就是以多节点集群

收稿日期: 2023-05-12

作者简介: 侯博学 (1999-), 男, CCF 会员, 上海第二工业大学计算机与信息工程学院硕士研究生, 研究方向为计算机视觉、数据库、自动驾驶; 陈林 (1979-), 男, 博士, 上海第二工业大学计算机与信息工程学院副教授、硕士生导师, 研究方向为信息物理融合系统、深度学习。

方式部署。GFS是单主多从的架构,并且将控制与数据的分离,数据的Metadata存储在主节点,数据则存储在从节点。在没有缓存的情况下,一次读请求需要访问两次集群,第一次需要向 Master 请求数据的存储位置信息,第二次根据位置信息去请求对应的从节点,经过这两次操作就拿到了数据^[2]。GFS读请求示意图如图 1 所示。

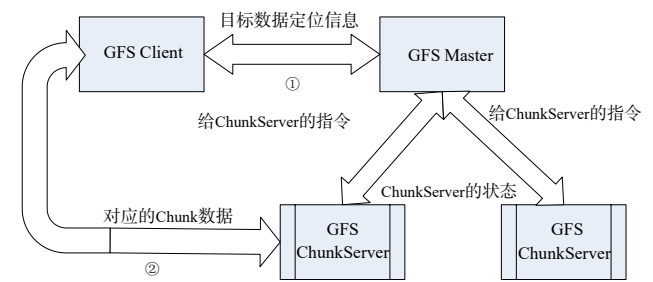


Fig. 1 Schematic diagram of GFS receiving read request
图 1 GFS 接收读请求示意图

业务系统处理数据分为联机事务处理(OLTP)与联机分析处理(OLAP)两类。在数据量大的企业中通常会维护不同类别的数据库以支持数据处理,一类是以 MySQL 为主的事务处理,另一类是以存储非结构化数据为主的 Click-house 等数据仓库。这些数据库为了加快数据提取分析速度,改变了关系型数据库通常的行存做法,将同一属性不同字段的一系列数据存到一起(列存),加快了对具体指定列属性数据的提取。

与此同时,维护两套系统对于企业而言意味着成本的增加,于是 HTAP(Hybrid)的概念应运而生。HTAP 是创新统一支持 OLAP 和 OLTP 的数据存储方式。目前,HTAP 主要有两种实现方式:①主备库物理隔离,主库运行 OLTP 负载,备库运行 OLAP 负载,主备之间通过 Redo Log 进行数据同步;②将两种存储引擎进行统一设计,通过一套负载引擎实现 HTAP,通过区分 OLAP 与 OLTP 的请求所在资源组,对资源组进行逻辑隔离,例如 Oracle 的多租户隔离机制^[3]。

1 数据模型

数据模型是数据存储、数据组织、数据管理的关键,有层次模型、网状模型和关系模型 3 种类型。

(1)层次模型。由于树形结构具有很好的层次性,因此层次模型是用树状结构描述实体和它们之间的关系。在层次模型中,每一条记录都用节点表示,实体之间的联系则由树的节点之间的路径表示。每一个非叶子节点可以有多个子节点,但每一个子节点只允许有一个父节点。因此,层次模型采取树形结构表达,决定了其只能处理一对多的实体联系。

(2)网状模型。网状模型是图状结构,其内部实体可表达的关系比层次模型复杂得多,因此网状模型也更加灵

活。与层次模型不同的是,网状模型运行一个节点可以同时拥有多个父节点,该节点也可以有多个子节点。对于一般性的关系,网状模型更加适用^[4]。

(3)关系模型。关系模型是现代数据库管理的关系型数据模型,基于谓词逻辑与集合论,通过交、并、差、笛卡尔积等一系列集合操作得到数据。关系模型是第一个形式化的数据库模型,其中的基本关系块是元组,元组是属性的有序集合体。元组组成如表 1 所示^[5]。

Table 1 Tuple
表 1 元组

姓名	年龄	性别	电话号码	冗余字段
张三	23	女	1376725438	无

2 存储

数据库的主要功能之一是存储数据,数据如何存储、存储在哪些地方、如何减少数据操作时的读写放大问题等,都关系到数据库的执行效率。目前,数据库领域的存储主要有两大阵营:①基于非易失性存储;②基于持久化存储。

2.1 基于易失性存储的数据库

基于易失性存储的数据库是将数据主要存储在内存中,利用内存的访问速度优势提升数据库的查找、写入效率。但是,为了数据存储的可靠性,基于易失性存储的数据库都能实现数据恢复机制,比如 WAL(Write Ahead Log),以保证数据库高可用。

写前日志(WAL)是比较通用的实现方式,在真正的写入数据前,数据库会先将这次操作封装成一条日志,然后将这条日志以追加的形式写入本地文件,这样能够在很大程度上确保数据不丢失。并且,由于是顺序追加写,故访问磁盘的代价会小很多。典型的基于易失性存储的数据库有 Redis,它是一个 KV 存储数据库,其大部分操作都是关于内存,数据恢复部分会涉及到磁盘。Redis 内部实现了 RDB(Redis Database)和 AOF(Append Only File)两种协作式数据恢复机制^[6]。

2.2 基于持久化存储的数据库

基于持久化存储的数据库主要依赖于磁盘存储,以前更多的是将数据存储于机械磁盘(HDD),随着磁盘硬件的发展,数据目前更多存储于固态硬盘(SSD),甚至是硬盘厂商专门为大数据量厂商定制的磁盘。数据在磁盘上往往是分页存储,也即 DBMS 将磁盘空间逻辑抽象成磁盘页的形式,一页的存储大小为 16KB。在每一页中,根据存储的数据不同又可分为两种数据库:①存储多个元组(Tuple),也即关系数据库表中的一行(行存),将真实的数据存储在元组内;②Page 存储对数据的操作记录,当需要读取目标 Tuple 时,DBMS 会扫描出关于该 Tuple 的操作,然后重建出该 Tuple。Tuples 在 Page 中的逻辑表示如图 2 所示^[7]。

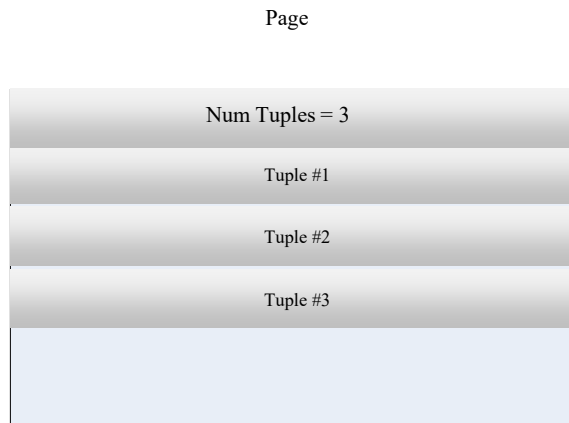


Fig. 2 Logical representation of Tuples in Page

图2 Tuples在Page中的逻辑表示图

同时,存在两种维护磁盘页的方式。第一种是利用链表的思想,空闲页之间由 Free Page List 链在一起,已存储的磁盘页由 Data Page List 链在一起,而两者的起始地址由 Header Page 维护,这样通过 Header Page 可以遍历寻找到所有的 Page,查找时间复杂度为 $O(n)$ 。磁盘页链式维护如图 3 所示。第二种是采用以空间换时间的方式,DBMS 维护字典页以记录所有页的情况,当需要查找时,只需访问该字典页,便可得到数据页的地址,这样访问时间复杂度就被降低。磁盘页字典式维护如图 4 所示。

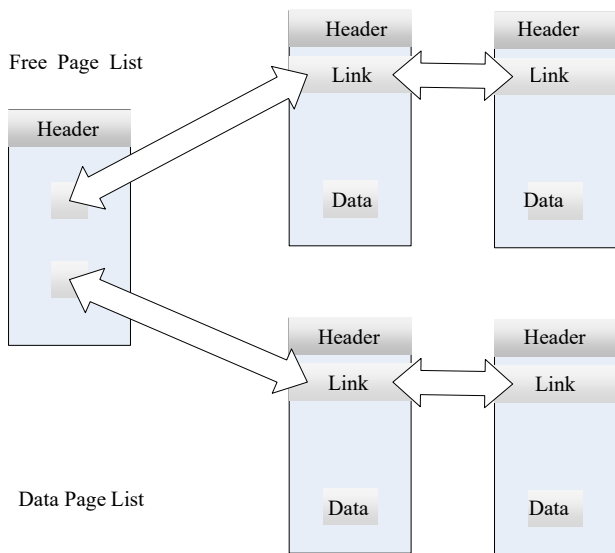


Fig. 3 Schematic diagram of disk page chain maintenance

图3 磁盘页链式维护示意图

在基于持久化存储的数据库中,数据库通常也会使用内存作为热点数据的缓存,以加速对数据的存取。比如,MySQL 的可插拔存储引擎 InnoDB 会开启 Buffer Pool 以减少对磁盘的访问次数。Buffer Pool 是由 DBMS 维护的一段内存空间,DBMS 会将频繁访问的磁盘页缓存在 Buffer Pool 中,因为 Buffer Pool 的容量有限,其会采取一定的淘汰策略以剔除冷数据页。淘汰策略的好坏决定了 Buffer Pool 的利

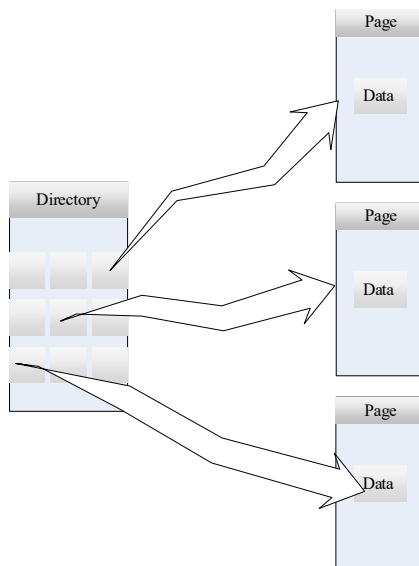


Fig. 4 Schematic diagram of disk page dictionary maintenance

图4 磁盘字典式维护示意图

用效率,选取的淘汰策略要尽可能地降低内存与磁盘之间的数据移动次数,以减少 DBMS 性能的抖动^[8]。

3 并发

随着数据量的增大与用户访问量的增加,串行式的访问已不能满足数据库访问者对速度的要求,因而在设计数据库时需考虑并发的存在。在关系型数据库中,同一时刻可能会发生多个事务,多个事务亦可能涉及相同的目标数据,这对数据库的一致性是一个挑战。因此,多线程并发对资源的互斥访问是有待解决的一大问题,这在数据一致性要求高的产品中尤为重要,比如支持事务的数据库的 ACID 特性。原子性(Atomicity)指事务中的操作要么都成功,要么都失败,不能出现部分成功的状态;一致性(Consistency)指一个事务必须使数据库从一个状态切换到另一个状态,这意味着数据库必须维护完整性约束,使得数据库在事务完成前后保证一致性与正确;隔离性(Isolation)指并发执行的事务之间不能互相干扰,这也意味着一个事务内部使用的数据对别的事务而言说是不可见的,常见的隔离级别如表 2 所示。

Table 2 Normal transaction isolation

表2 常见的隔离级别

级别	解释	实现
读未提交	允许脏读(读另一个事务未提交的数据)	/
读已提交	只允许读到已提交的数据,不保证可重复读	行锁
可重复读	只允许读到已提交的数据,同一事务的两次读取之间,不允许其他事务对其进行更新	快照 MVCC
可串行化	保证串行处理	2PL+意向锁

持久性(Durability)指一个事务如果提交了,那么它产生的效果就是持久的,而且不能随意更改。数据库如果要满足以上 ACID 4个特性,就需要引入并发控制协议。并发控制协议分为乐观与悲观两种。

3.1 悲观并发控制协议

悲观并发控制协议是在事务一开始就将用到的所有数据加锁,第一时间阻止并发问题的出现,但是由于其会对数据进行大量加解锁操作,这也就意味着事务处理性能低下,因而其更多的应用于写、读写事务频率较高的场景,不适合读事务较多的场景。常见的悲观并发控制协议有两阶段锁(Two Phase Locking, 简称为 2PL)。为防止无序的加解锁可能导致的死锁情况发生,2PL 将数据库的事务处理划分为两个阶段,锁增加阶段和锁收缩阶段。在锁增加阶段只允许事务进行加锁(Lock)进行操作,不允许进行解锁(Unlock)操作,在锁收缩阶段只允许事务进行解锁(Unlock)操作,不允许进行加锁(Lock)操作。2PL 已经通过严格的数学证明其能够正确地处理并发事务。2PL 加解锁阶段如图 5 所示^[9]。

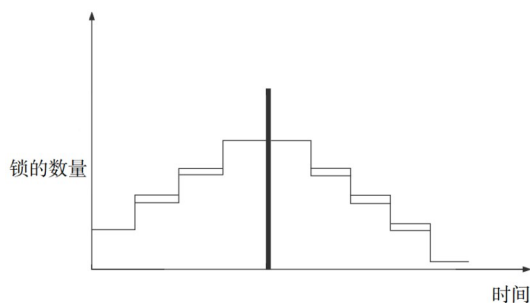


Fig. 5 Schematic diagram of two-phase lock quantity and time

图5 两阶段锁数量与时间示意图

3.2 乐观并发控制协议

乐观并发控制协议假定多个事务之间并不会操作相同数据,因而允许事务之间穿插并行访问。乐观并发控制协议有3个阶段:①读阶段,追踪读写请求并将它们存储在其私有空间内;②验证阶段,当一个事务准备提交时,DBMS 检测其是否与其它事务有冲突;③写(提交)阶段,如果通过验证阶段,DBMS 将该事务保存至私有空间的修改应用于数据库,如果其没有通过验证,该事务会被中断并且回滚。因此,乐观并发控制协议更适合于只读事务频繁出现的场景,这样带来的回滚开销非常小。乐观的并发控制协议有基于时间戳排序的并发控制协议(Timestamp Ordering Concurrency Control)、多版本并发控制(Multi-Version Concurrency Control 简称为 MVCC)。

基于时间戳排序协议,使用时间戳为事务进行定序,从而使事务之间达到串行执行序列。每一个事务到来时,DBMS 会为该事务指派一个唯一单调递增的时间戳。如果事务*i*被分配到的时间戳小于事务*j*被分配到的时间戳,则 DBMS 需要保证事务*i*需要先于事务*j*被执行。

基于多版本并发控制协议进行事务处理的数据库会

为单个逻辑对象维护多个物理版本。对于写事务,其会直接创建一个新版本;对于读事务,DBMS 会为该事务返回距离其开始最近版本的数据,也即其能看到的最新数据。此协议中发起写事务的请求不会阻塞读者,读者也不会阻塞写者。对于不同版本的维护,又细分为3种存储方式。第一种是 Append-Only 存储,DBMS 会为该元组维护一个主表,该元组及后续更新所生成的新元组都会存储在该主表中,元组每发生一次更新,都会在该元组后面追加一个新版本的元组。第二种是 Time-Travel 存储,在这种存储模式下,DBMS 维护了两张表,一张是主表,一张是时间旅行表(Time-Travel Table),主表内存储了针对元组的最新更新,时间旅行表内存储了该元组的历史版本。第三种是 Delta 存储方式,在 Delta 存储方式中,DBMS 维护了主表和 Delta Storage Segment。与 Time-Travel 不同的是,Delta Storage Segment 中记录的并不是更新后的元组,其记录的是对该元组施加操作,这种方式对写操作十分友好,而对于读操作,DBMS 需要读取目标元组过往被施加的所有操作,然后重放这些操作,从而得到最新的元组。

4 日志与恢复

日志对于数据库的正常运行至关重要,在数据库宕机恢复时会用到日志,比如 Redis 中的 RDB 与 AOF、MySQL 中的 Redo Log 等。在 Aurora 数据库中,会将日志与数据一起在数据中心之间移动,对于 Aurora 而言,“日志就是数据库”,可见日志在数据库中的地位。这样能够让数据库从崩溃中恢复到之前的状态,对于数据的一致性、事务的原子性而言都非常重要。错误可能发生在事务内部,也可能是系统出错或者更严重的存储介质出错。对于最严重的磁盘出错,反而有最简便的处理方式^[10]。

4.1 日志

为了处理事务执行过程中可能出现的错误,数据库一般需要维护 Undo 与 Redo 两种日志。Undo 日志的作用是让数据库能够回到事务执行之前的状态,其重要性主要体现在两个方面:①数据库恢复;②事务回滚。Redo 日志体现了数据库的持久性,如果数据库突然断电导致已经提交的事务还没有写到磁盘而丢失,则数据库在恢复运行后需要通过某种机制将数据库已经承诺提交的数据重新写入磁盘(重做)。Undo 与 Redo 的实现都依赖于 Buffer Pool 管理系统,Buffer Pool 有不同的写盘策略,包含是否 STEAL、是否 FORCE,其对错误的处理也不同。使用 STEAL 策略意味着数据库允许在事务还有没提交时就将其写入磁盘,对于这种操作,当数据库需恢复时要将写盘的操作 Undo 回去。使用 NO-FORCE 策略意味着在事务提交之后不需要将数据也落盘。但是,这也意味着系统在崩溃后,会造成已经提交事务的数据丢失,此时需要利用 Redo 日志将它的操作进行重放。从以上策略多样性可以看出,数据库

设计并没有要求其在某个时间点将数据写入磁盘,这给了数据库很大程度的灵活性,但是为了数据恢复,数据库需要实现对应的恢复机制。

4.2 恢复

数据库为了实现恢复数据的高效性,一般存在Shadow Paging与WAL两种容错机制。Shadow Paging实现方式是在数据库中维护两份副本:①Master副本,其包含了所有已提交事务施加的更改;②Shadow副本,其包含了还没有提交事务内的更改。当一个事务发起提交操作时,DBMS会原子性地让Shadow副本变为Master副本,此时Buffer Pool的策略是NO-STEAL + FORCE。容错机制WAL的全称为预写日志(Write Ahead Log),其主要思想是每次将数据变化写入磁盘前都需要先将这条修改操作持久化。由于WAL是以追加写的方式进行,因此产生的额外写入开销不会太大。与此同时,数据库也会使用WAL Buffer来缓存多条修改语句的WAL Log,以将多条WAL日志进行组提交,从而减少对磁盘的IO次数。

对于一款成熟的数据库产品而言,错误的发生是偶然的。为了防止写预写日志(WAL)时发生异常,对Log的追加写一般会使用O_DIRECT函数以绕过内核空间直接写硬盘(为了防止数据还在内核缓存区时断电),这一步要尽可能快。将Page flush进磁盘时才会调用fsync函数,因此日志的长度与数据库平稳运行时间呈正相关。当数据库在平稳运行很长一段时间后突然发生错误,后续恢复数据库状态时,从“远古时期”开始执行日志并不是一个明智的选择。因此,数据库会定时在日志打上检查点(Checkpoints)标记,检查点的状态由之前状态转移而来,因而数据库决定恢复时仅需从日志里最近的检查点执行即可。

5 查询

查询是数据库技术的最基本功能之一,调查结果显示,网站打开时长超过3 s就会有57%的用户离开^[11],因而如何以更快的速度\更高的准确率将结果呈现给用户,是数据库领域一直努力的方向。为了加速数据查询,数据库提供了索引建立机制,并且会默认给区分度高的键建立索引(比如主键)。索引可以看作是存储了原表中一些属性的一个子集。DBMS在建立索引时会有两个取舍:①索引也需要占据空间,因此使用索引是使用空间换取时间的策略;②DBMS需要维护索引的更新,在每一次数据被修改后,DBMS需要将其索引进行修改以保证数据的正确性。并且为了平衡读写请求,根据数据库的应用场景不同,索引的设置也不同,MySQL的InnoDB、MyISAM引擎使用B+ Tree作为索引以加速数据查询。但同时,在它们的叶子节点存储的数据有所不同,InnoDB的B+ Tree主键索引存储的是真实数据,而MyISAM的B+ Tree主键索引的叶子节点存储的是指向数据的指针。除建立主键索引外,DBA也会

针对一些查询频率较高的字段,根据个人经验建立索引,并且使用对查询较为友好的SQL语句,比如遵循最左前缀匹配原则。为了加速查询,除建立索引外,考虑到对数据的访问具有空间局部性,DBMS还会开启prefetch(预读)以将数据的邻居一并读入,这在一定程度上缓解了机械磁盘的性能差距^[12]。

在非关系型数据库中,同样会引入加速查询的数据结构。在高性能键值对存储数据库Pebbles DB中使用Bloom Filter以提升查询性能。Bloom Filter是一种通过多个哈希函数快速判断查询目标是否存在的一种数据结构。由于其设计上的特性,Bloom Filter存在假阳性(False Positive)的问题,随着Hash函数的增多,Bloom Filter判断的正确率会上升,同时计算量也会增大^[13]。在Pebbles DB中,写入的数据会首先被存储在内存中的MemTable里,当MemTable中写入的数据达到指定大小Options.write_buffer_size后,则自动转换为只读的Immutable Memtable,同时生成新的MemTable以存储新的数据。后台进程会进行紧凑(Compact)操作以将Immutable Memtable存储在磁盘内生成Sstable,随着数据写入量的增加,磁盘内的Sstable也逐渐增加,查找操作消耗的平均时长显著上升,因此Pebbles DB引入了Bloom Filter以加速数据查找过程,当Bloom Filter判定该数据不可能存在于该Sstable时,就跳过对该Sstable的查询,从而减少了磁盘IO次数^[14]。

在另一种键值对数据库Redis中,同样使用到了高级数据结构以优化数据的读性能。Redis中的有序集合zset(Sorted Set)底层基于跳表(Skip List)实现,跳表是一种随机化的数据结构,其实质是一种可以进行二分查找的有序链表,通过在原有最底层有序链表上方冗余多级索引的方式加速查询过程^[15]。跳表的逻辑表示如图6所示。

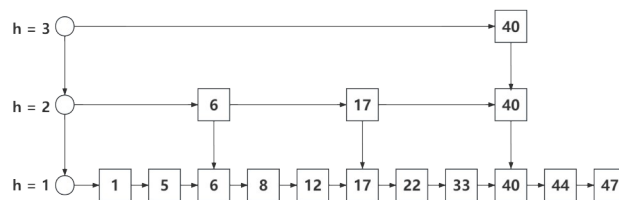


Fig. 6 Logic diagram of Skip-List

图6 跳表的逻辑示意图

6 分布式数据库

随着信息化的发展,单体数据库已经不能满足海量数据的存储需求,因而催生出了一大批分布式数据库。在分布式数据库中,如何管理多个节点、如何处理跨节点的分布式事务、如何平衡节点的负载以及数据在节点间的迁移,都需要考虑并加以解决。

在分布式系统中,最重要的是节点之间的共识问题。共识意味着多个节点的实质是隐藏的,从系统外部看,它们是一个整体。下文主要介绍一种易于理解和工业实现

的共识算法——Raft,同时提及其它开源组件中的共识算法实现。

6.1 共识算法

Raft 共识算法是一种分布式一致性算法,用来解决在分布式系统中多个节点之间如何达成一致的问题。其基本思想是将节点分为多个角色(如领导者(Leader)、追随者(Candidate)和候选人(Follower)等),并通过发送心跳实现共识。Raft 算法的工作过程大致如下:①当一个节点启动时,它首先作为一个候选人发起选举;②当候选人收到其它节点的选票时,它会计算收到的选票数,如果超过半数,则成为 Leader;③Leader 节点会不断发送心跳给其它节点,以确保它们依然认可它的领导地位;④追随者会收到领导人的心跳,并根据领导人的日志更新自己的日志。如果追随者在一段时间内没有收到领导人的心跳,则会认为 Leader 节点已经宕机,其会重新发起选举。

Raft 共识算法与其他一致性算法(如 Paxos)相比,具有更简单的语义和更好的实现。在具体工业实现时,可以在系统中加入观察者和无投票权成员以分流系统中的读请求^[16]。

分布式键值数据库 TiKV 在简单 Raft 算法的基础上进行了性能优化:①将多次修改进行缓存,然后将它们进行组提交;②Leader 并行地发送日志给 Followers;③当一个 log 被集群中的大多数节点提交后,使用另一个线程异步地应用这个日志^[17]。

6.2 分布式事务

分布式事务是指跨多个节点运行的事务,具有原子性、一致性、隔离性和持久性四大特性,可以确保事务在分布式系统中的正确执行。它是分布式系统中的一项重要技术,在大型电子商务系统、分布式数据库系统等领域都有广泛应用。Google 的分布式系统 Spanner 在处理跨全球的事务时采用了优化后的两阶段提交(2PC)技术^[18]。两阶段提交(Two-Phase Commit)是一种在分布式事务中用以确保事务在多个节点上正确执行的技术。它通过将事务的提交过程分为两个阶段以保证事务的正确性。朴素的 2PC 流程工作大致如下:

(1)预提交阶段(Prepare Phase)。在这个阶段,事务协调者(coordinator)会向所有参与者(participant)发送一个请求,要求它们准备提交事务。参与者会检查事务是否能够成功提交,并向协调者返回一个结果。如果所有参与者都能够成功提交事务,协调者会进入下一个阶段。

(2)提交阶段(Commit Phase)。协调者会向所有参与者发送一个提交命令,要求它们执行事务。参与者会按照命令执行事务,并向协调者返回执行结果。如果所有参与者都成功执行了事务,协调者会将事务提交成功,否则会将事务回滚。两阶段提交的工作流程可以保证事务在分布式系统中的正确执行,并且能够应对各种网络故障和节点故障。但是,其缺点也很明显,两阶段提交的工作流程

需要额外的通信和协调,会增加系统的开销和延迟。此外,如果在提交阶段有参与者发生故障,整个事务都会失败,回滚事务也会带来额外的开销。因此,Spanner 将优化的重心放在此处。Spanner 采用 Paxos 共识算法以解决单点故障,它将几个副本组成一个 Paxos 组,使得 3 个副本中任意一个崩溃都不会影响整个系统的可用性。这样增加了系统的通信开销,因为节点之间需要达成共识,但不会像之前那样每个节点都需要与协调者通信。两阶段提交示意图如图 7 所示。

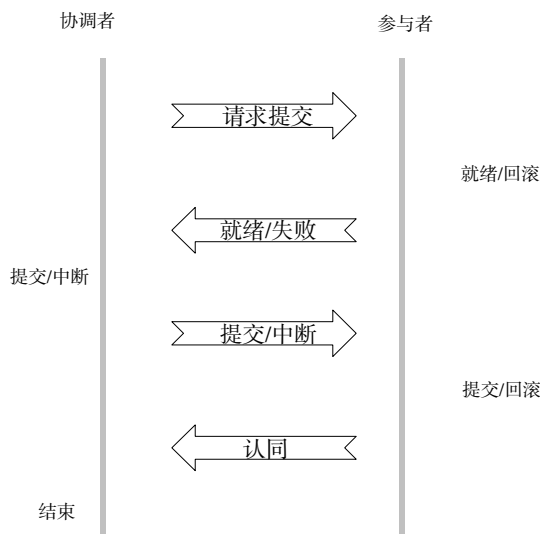


Fig. 7 Schematic diagram of two-phase commit

图7 两阶段提交示意图

7 机器学习与数据库

机器学习和数据库结合是一个新兴的领域,它涉及到将机器学习技术应用于数据库领域,以提高数据库的性能、可用性和智能化程度。在该领域中,机器学习技术主要应用于数据库的优化、自动维护和自我管理等方面。例如,机器学习技术可以通过自然语言处理和机器学习等手段,提高数据库查询效率,并能够自动生成优化建议。此外,机器学习技术还可以帮助数据库进行自动化管理,比如自动检测和修复数据库中的错误,以提高数据库的可用性。当前,机器学习与数据库结合领域正处于快速发展阶段,许多新的技术和产品都在不断推出^[19]。例如, Otter Tune,它是一个能够对数据库进行自动调优的系统,数据库的参数对数据库的性能有着重要影响。例如,在 MySQL 的 InnoDB 引擎中, innodb_buffer_pool_size 和 innodb_log_file_size 等参数对于数据库的性能非常关键^[20],并且正确调整参数可以显著提高数据库的性能。然而,调优这些参数却非常困难,数据库中的参数个数已经远远超过人类能够手动调优的范围。通过“show variables”参数查看 MySQL 的参数,其已有 500 多个, PostgreSQL 的参数个数也有 200 多个。对于数据库管理员而言,手动调优这些参数

非常困难,而 Otter Tune 通过机器学习实现了一种自动调参系统以解决上述问题^[21]。

除参数对数据库性能影响巨大外,索引也是决定查询性能的关键。现有技术的索引调优器依靠查询优化器的成本估算以搜索出最优索引配置。由于优化器的估算存在局限性,故在大部分情况下,为改善查询被估算执行成本(例如 CPU 时间)的索引,实际上会导致性能变差。这类错误是在生产系统中建立自动索引的主要障碍。AI Meets AI 团队观察到,在索引调优过程中,比较对应于不同索引配置的同—查询的执行成本是关键步骤。AI Meets AI 团队的关键观点是,将这一步骤表述为机器学习中的分类任务可以显著提高准确度,而不是使用优化器的估算。AI Meets AI 团队对该分类问题的设计空间进行了研究,进一步展示了如何将该分类器与现有技术的索引调优器相结合,即人工智能(Artificial Intellect, AI)如何支持自动索引(Auto Index, AI)。AI Meets AI 团队使用业界标准基准测试和大量真实客户工作负载进行的评估表明,当模型用于索引调优时,几乎消除了所有的查询执行成本。总之, AI Meets AI 团队的研究表明,通过将两个方案执行成本的步骤比较表述为机器学习中的分类任务,可以显著提高索引调优准确度,并减少查询执行成本退化情况。

8 结语

数据库技术是信息管理的重要工具,在各行各业得到广泛应用。随着互联网的发展,数据的规模和复杂度迅速增长,对数据库技术提出了更新的挑战。在信息大数据处理的新时代,数据库技术需要更加高效、更多可扩展、更加智能化,以满足新的需求。未来的数据库技术更倾向于与人工智能、机器学习等技术相结合,实现智能化的数据管理和分析。此外,随着物联网技术的发展,数据库技术将广泛应用于物联网领域,例如 Apache IoTDB 等时序数据库。数据库技术可以帮助行业组织、企业和个人更好地管理、分析物联网设备生成的海量数据。

参考文献:

- [1] ALI A, NAEEM S, ANAM S, et al. A state of art survey for big data processing and nosql database architecture[J]. International Journal of Computing and Digital Systems, 2023, 14(1): 297–309.
- [2] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system[C]// Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003: 29–43.
- [3] GUO B, YU J, YANG D, et al. Energy-efficient database systems: a systematic survey[J]. ACM Computing Surveys, 2022, 55(6): 1–53.
- [4] SILBERSCHATZ A, KORTH H F, SUDARSHAN S. Database system concepts[M]. 3rd Edition. New York: McGraw-Hill Book Company, 1997.
- [5] ALAMI A E, BAHAJ M, KHOUDIFI Y. Supply of a key value database re-dis in-memory by data from a relational database [C]// 2018 19th IEEE Mediterranean Electrotechnical Conference (MELECON), 2018: 46–51.
- [6] WENG S, WANG Q S, QU L, et al. Lauca: a workload duplicator for benchmarking transactional database performance[J]. IEEE Transactions on Knowledge and Data Engineering, 2024(early access): 1–14.
- [7] RASIKAN D, SANG H S. A secure two phase locking protocol[C]// Proceedings of 12th Symposium on Reliable Distributed Systems, 1993: 126–135.
- [8] LI Q, LYU M, XU L, et al. Fast recovery for large disk enclosures based on RAID2. 0: algorithms and evaluation[J]. Journal of Parallel and Distributed Computing, 2024, 188: 104854.
- [9] SILITONGA D, ROHMAYANTI S A A, ARIPI N Z, et al. Edge computing in e-commerce business: economic impacts and advantages of scalable information systems[J]. ICST Transactions on Scalable Information Systems, 2023, 11(1): 1–11.
- [10] MADAVI C, PATEL C, KATE V, et al. Mysql select query optimization using self-join [EB/OL]. <https://stackoverflow.com/questions/16846659/sql-query-using-self-join/16846711>.
- [11] RAJU P, KADEKODI R, CHIDAMBARAM V, et al. Pebblesdb: building key-value stores using fragmented log-structured merge trees [C]// Proceedings of the 26th Symposium on Operating Systems Principles, 2017: 497–514.
- [12] PUGH W. Skip lists: a probabilistic alternative to balanced trees [J]. Communications of the ACM, 1990, 33(6): 668–676.
- [13] ONGARO D, OUSTERHOUT J. In search of an understandable consensus algorithm [C]// 2014 USENIX Annual Technical Conference (Usenix ATC 14), 2014: 305–319.
- [14] HUANG D, LIU Q, CUI Q, et al. TiDB: a Raft-based HTAP database [J]. Proceedings of the VLDB Endowment, 2020, 13(12): 3072–3084.
- [15] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database [J]. ACM Transactions on Computer Systems, 2013, 31(3): 1–22.
- [16] CAI B, LIU Y, ZHANG C, et al. HUNTER: an online cloud database hybrid tuning system for personalized requirements [C]// Proceedings of the 2022 International Conference on Management of Data, 2022: 646–659.
- [17] QIAO Y. Design of database systems with DRAM-only heterogeneous memory architecture [C]// 2020 IEEE 36th International Conference on Data Engineering, 2020: 2054–2058.
- [18] VAN AKEN D, PAVLO A, GORDON G J, et al. Automatic database management system tuning through large-scale machine learning [C]// Proceedings of the 2017 ACM International Conference on Management of Data, 2017: 1009–1024.
- [19] DING B, DAS S, MARCUS R, et al. AI meets AI: leveraging query executions to improve index recommendations [C]// Proceedings of the 2019 International Conference on Management of Data, 2019: 1241–1258.
- [20] ZHANG H, ZENG X, ZHANG S, et al. Scalable online interval join on modern multicore processors in OpenMLDB [C]// 39th International Conference on Data Engineering, 2023: 3031–3042.
- [21] WANG C, HUANG X, QIAO J, et al. Apache IoTDB: time-series database for internet of things [J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2901–2904.

(责任编辑: 孙娟)