07 概要：什么是存储引擎，为什么需要了解它？

经过第一个模块的学习，相信你已经知道了什么是分布式数据库，对分布式数据库的核心知识有了比较全面和深入的了解了。

这一讲是第二模块存储引擎的概要，主要目的是为你解释什么是存储引擎，以及它在分布式数据库中起到什么样的作用。

数据库的一个首要目标是可靠并高效地管理数据，以供人们使用。进而不同的应用可以使用相同的数据库来共享它们的数据。数据库的出现使人们放弃了为每个独立的应用开发数据存储的想法，同时，随着数据库广泛的使用，其处理能力飞速发展，演进出如现代的分布式数据库这般惊人的能力。

那么，为了支撑抽象的多种场景。一般的数据库都会采用多模块或多子系统的架构来构建数据库，从而方便数据库项目团队依据现实的场景来组合不同的子模块，进而构造出一众丰富的数据库产品。

而存储引擎就是这一众模块中极为重要的一环，下面我们开始解释它在整个数据库架构中的定位和意义。

存储引擎的定位

这个世界上，没有针对数据库设计的一定之规。每个数据库都是根据它所要解决的问题，并结合其他因素慢慢发展成如今的模样的。所以数据库子模块的分化也没有一个广泛接受的标准，且有些模块之间的边界也是很模糊的。特别是需要优化数据库性能时，原有被设计为独立存在的模块很可能会融合以提高数据库整体性能。

这里，我总结出了一个**比较典型的分布式数据库的架构和模块组合标准**。虽然不能完全代表所有分布式数据库，但是可以帮助你理解模块的组成方式。这里需要注意，我给出的模型是基于客户端/服务器，也就是 C/S 模式的，因为这是大部分分布式数据库的架构模式。

1. 传输层：它是接受客户端请求的一层。用来处理网络协议。同时，在分布式数据库中，它还承担着节点间互相通信的职责。
2. 查询层：请求从传输层被发送到查询层。在查询层，协议被进行解析，如 SQL 解析；后进行验证与分析；最后结合访问控制来决定该请求是否要被执行。解析完成后，请求被发送到查询优化器，在这里根据预制的规则，数据分布并结合数据库内部的统计，会生成该请求的执行计划。执行计划一般是树状的，包含一系列相关的操作，用于从数据库中查询到请求希望获取的数据。
3. 执行层：执行计划被发送到执行层去运行。执行层一般包含本地运行单元与远程运行单元。根据执行计划，调用不同的单元，而后将结果合并返回到传输层。

细心的你可能会注意到，这里只有查询层，那么数据是怎么写入的？这对于不同的数据库，答案会非常不同。有的数据库会放在传输层，由于协议简单，就不需要额外处理，直接发送到执行层；而有些写入很复杂，会交给查询层进行处理。

以上就是数据库领域中比较常见的模块划分方式。你可能有这样的疑问：那么存储引擎在哪里呢？

执行层本地运行单元其实就是存储引擎。它一般包含如下一些功能：

1. 事务管理器：用来调度事务并保证数据库的内部一致性（这与模块一中讨论的分布式一致性是不同的）；
2. 锁管理：保证操作共享对象时候的一致性，包括事务、修改数据库参数都会使用到它；
3. 存储结构：包含各种物理存储层，描述了数据与索引是如何组织在磁盘上的；
4. 内存结构：主要包含缓存与缓冲管理，数据一般是批量输入磁盘的，写入之前会使用内存去缓存数据；
5. 提交日志：当数据库崩溃后，可以使用提交日志恢复系统的一致性状态。

以上就是存储引擎比较重要的几个功能，其核心就是提供数据读写功能，故一般设计存储引擎时，会提供对其写入路径与读取路径的描述。

好了，现在你清楚了存储引擎的定位和主要结构，那么存储引擎的种类也是很多的，下面我通过一些关键特性，来介绍几种典型的存储引擎。

内存与磁盘

存储引擎中最重要的部分就是磁盘与内存两个结构。而根据数据在它们之中挑选一种作为主要的存储，数据库可以被分为内存型数据库与磁盘型数据库。由此可见存储引擎的一个功能，就是可以被作为数据库类型划分的依据，可见引擎的重要性。

内存型存储是把数据主要存储在内存里，其目的很明显，就是加快数据读写性能。分布式数据库一个重要的门类就是内存型数据库，包括 Redis、NuoDB 和 MySQL Cluster 等。当然其缺点也很明显，那就是内存的成本较高，且容量有限。而分布式的架构能有效地扩充该类数据库的容量，这也是内存数据库主要是分布式数据库的原因。

磁盘存储相对传统，它存储主要数据，而内存主要作为缓冲来使写入批量化。磁盘存储的好处是，存储性价比较高，这主要得益于磁盘甚至是磁带的单位存储价格相比内存非常低廉。但是与内存型数据库相比，磁盘型数据库的性能比较低。不过，随着近年 SSD 磁盘的普及，这种趋势得到了有效的改善。

这两种存储引擎的差别还体现在功能实现的难度上。内存型数据库相对简单，因为写入和释放随机的内存空间是相对比较容易的；而磁盘型数据库需要处理诸如数据引用、文件序列化、碎片整理等复杂的操作，实现难度很高。

从目前的分布式数据库发展来看，磁盘型存储引擎还是占据绝对统治地位的。除了性价比因素外，内存型数据库要保证不丢失数据的代价是很高昂的，因为掉电往往就意味着数据的丢失。虽然可以使用不间断电源来保证，但是需要复杂的运维管理来保证数据库稳定运行。

然而近年来，随着 NVM（Non-Volatile Memory，非易失性内存）等技术的引入。这种情况开始出现了一些变化，此种存储具有 DRAM 内存的性能，同时能保证掉电后数据不丢失。且最重要的是读写模式类似于内存，方便应用去实现功能。有了它的加持，未来内存型数据库还将有比较大的发展。

除了硬件加持，内存型数据库也可以通过结构设计来保证数据不丢失。最常用的手段就是使用数据备份+提交日志的模式。数据库为了不影响写入读取性能，可以异步地备份数据。同时在每次写入数据之前要先写入提交日志，也就是说提交日志的写入成功才被认为是数据写入成功。

当数据库节点崩溃恢复后，将备份拿出来，计算出该备份与最新日志之间的差距，然后在该备份上重放这些操作。这样就保证数据库恢复出了最新的数据。

除了内存和磁盘的取舍，存储引擎还关心数据的组合模式，现在让我们看看两种常见的组合方式：行式与列式。

行式存储与列式存储

数据一般是以表格的形式存储在数据库中的，所以所有数据都有行与列的概念。但这只是一个逻辑概念，我们将要介绍的所谓“行式”和“列式”体现的其实是物理概念。

行式存储会把每行的所有列存储在一起，从而形成数据文件。当需要把整行数据读取出来时，这种数据组织形式是比较合理且高效的。但是如果要读取多行中的某个列，这种模式的代价就很昂贵了，因为一些不需要的数据也会被读取出来。

而列式存储与之相反，不同行的同一列数据会被就近存储在一个数据文件中。同时除了存储数据本身外，还需要存储该数据属于哪行。而行式存储由于列的顺序是固定的，不需要存储额外的信息来关联列与值之间的关系。

列式存储非常适合处理分析聚合类型的任务，如计算数据趋势、平均值，等等。因为这些数据一般需要加载一列的所有行，而不关心的列数据不会被读取，从而获得了更高的性能。

我们会发现 OLTP 数据库倾向于使用行式存储，而 OLAP 数据库更倾向于列式存储，正是这两种存储的物理特性导致了这种倾向性。而 HATP 数据库也是融合了两种存储模式的一种产物。

当然这里我们要区分 HBase 和 BigTable 所说的宽列存储与列存储在本质上是不同的。宽列存储放在其中的数据的列首先被聚合到了列簇上，列簇被放在不同的文件中；而列簇中的数据其实是按行进行组织的。

选择行模式与列模式除了以上的区分外，一些其他特性也需要考虑。在现代计算机的 CPU 中，向量指令集可以一次处理很多类型相同的数据，这正是列式存储的特点。同时，将相同类型数据就近存储，还可以使用压缩算法大大减少磁盘空间的占用。

当然，选择这两种存储模式最重要的因素还是访问模式。如果数据主要是按照行进行读取，比如交易场景、资料管理场景等，那么行式存储应是首选。如果需要经常查询所有数据做聚合，或者进行范围扫描，那么列式存储就很值得一试。

以上就是常见的数据的组合模式，那么组合好的数据如何存储在物理设备上呢？下面让我们探讨一下数据文件和索引文件两种常用的存放数据的物理原件。

数据文件与索引文件

上文介绍了内存与磁盘之间的取舍，从中可看到磁盘其实更为重要的，因为数据库是提供数据持久化存储的服务。故我们开始介绍磁盘上最为重要的两类文件：数据文件和索引文件。

数据文件和索引文件如名字所示，分别保存原始数据与检索数据用的索引数据。

但是随着时间的推移，两者的区分也不是那么泾渭分明了。其中以 IOT（索引组织表）模式为代表的数据文件在数据库，特别是分布式数据库中占据越来越重的位置。一种将两者进行融合的趋势已经变得势不可挡。

数据文件最传统的形式为堆组织表（Heap-Organized Table），数据的放置没有一个特别的顺序，一般是按照写入的先后顺序排布。这种数据文件需要一定额外的索引帮助来查找数据。

另外有两种数据表形式自带了一定的索引数据能力，即哈希组织表（Hash-Organized Table）和索引组织表（Index-Organized Table）。前者是将数据通过哈希函数分散到一组数据桶内，桶内的数据一般是按照一定规则进行排序，以提高查询效率；而后者一般采用索引文件的形式来存储数据，以 B+树为例，数据被存储在叶子节点上，这样做的目的是减少检索数据时读取磁盘的次数，同时对范围扫描支持友好。

索引文件的分类模式一般为主键索引与二级索引两类。前者是建立在主键上的，它可能是一个字段或多个字段组成。而其他类型的索引都被称为二级索引。主键索引与数据是一对一关系，而二级索引很有可能是一对多的关系，即多个索引条目指向一条数据。

这里按照索引与数据之间结合的程度，我们又可以把索引分为聚簇索引和非聚簇索引。前者如哈希组织表和索引组织表那样，数据的分布与索引分布是有关联的，它们被“聚”在一起，这样的查询效率很好。而后者最常见的例子就是针对这两种数据文件的二级索引，因为二级索引要索引的列不是主键，故索引与数据是分割的，查询时需要进行多次磁盘读取。但是对于写入，聚簇索引可能需要进行唯一判断，性能会比简单构建的非聚簇索引低效。

最后一点需要说明的是，二级索引需要保存指向最终数据的“引用”。从实现层面上，这个引用可以是数据的实际位置，也可以是数据的主键。前者的好处是查询效率高，而写入需要更新所有索引，故性能相对较低。而后者就恰好相反，查询需要通过主键索引进行映射，效率稍低，但写入性能很稳定，如 MySQL 就是选用后者作为其索引模式。

面向分布式的存储引擎特点

以上内容为存储引擎的一些核心内容。那分布式数据库相比传统单机数据库，在存储引擎的架构上有什么不同呢？我总结了以下几点。

内存型数据库会倾向于选择分布式模式来进行构建。原因也是显而易见的，由于单机内存容量相比磁盘来说是很小的，故需要构建分布式数据库来满足业务所需要的容量。

列式存储也与分布式数据库存在天然的联系。你可以去研究一下，很多列式相关的开源项目都与 Hadoop 等平台有关系的。原因是针对 OLAP 的分析数据库，一个非常大的应用场景就是要分析所有数据。

而列式存储可以被认为是这种模式的一种优化，实现该模式的必要条件是要有分布式系统，因为一台机器的处理能力是有瓶颈的。如果希望处理超大规模数据，那么将数据分散到多个节点就成为必要的方式。所以说，列模式是由分析性分布式的优化需求所流行起来的。

至于宽列存储更是分布式数据库场景下才会采用的模式。

数据文件的组织形式，分布式数据库几乎不会使用堆组织表。因为该形式过于随意，无法有效地分散数据。不知道学习过数据分片那一讲的时候你有没有注意到，另外两种组织表的名字与两种分片算法是有着天然联系的。

哈希组织表数据经过哈希函数散列到不同的桶，这些桶可以被分散到不同节点。而索引组织表一般叶子节点是按一定顺序排列的，这与范围分片又有着某种契合的关系。所以分布式数据库一般都会采用这两种模式作为其存储引擎，甚至一些分布式数据库直接将数据文件当作索引使用。

总结

好了，关于存储引擎我就介绍到这了。这一讲我们首先展示了数据库的整体架构，并点出了存储引擎所在的位置；而后分别讨论了存储引擎中几组概念的对比，并在最后说明了分布式数据库在引擎层面的选择及其原因。

当然，本讲只是一篇概述。存储引擎中其他重要的概念，我会在本模块随后的几讲中为你详细介绍。

上一页 下一页