**《大数据管理》 李嘉鹏 U202115652 2023.10.31**

**第六章**

1.（1）并行处理系统理想情况下的加速比应该是如何？而一般实际情况往往出现什么样的加速比？简述该现象的理由。

**理想情况下加速比应该是线性增大的，而实际情况中往往会出现加速比亚线性增大的现象。这是因为随着资源数的增加（例如处理器的数量）并不会全部反映到系统性能的提升上，处理器之间的通讯与同步可能会导致效率的下降，处理器对全局资源的访问可能会出现竞争现象。并且，并非所有的处理器都会无时无刻保持工作状态，如果由于任务分配不均衡导致处理器处于等待状态，也会导致性能的下降。**

（2）比理想加速比更好的表现是何种加速比？该现象在什么情况下可能出现？

**比理想加速比更好的表现是超线性加速比。其最基本的解释是并行程序的工作量少于串行程序的工作量。**

**最通常的情况下，并行执行时所要访问的数据都驻留在每个处理器的cache中，而顺序执行时必须访问存储器系统中较慢的部分，因为此时数据无法全都驻留在单个cache中。**

**还有一种情况来自于算法方面，比如当要查找的元素已被发现导致搜索结束。当搜索以并行方式进行时，能以不同的次序有效地进行，意味着总的被搜索数据量将少于顺序搜索的情况。因此，并行执行将完成较少的工作。**

2.（1）对于3个关系的连接操作，R1⋈R2⋈R3，请给出其操作间并行处理的方案。

**操作间并行可以分为流水线并行和独立并行两种方式。其中流水线并行的处理方案为：**

**步骤1：RESULT1= R1⋈R2**

**步骤2：RESULT2= RESULT1⋈R3（在步骤1后执行）**

**独立并行的处理方案为：**

**将原始操作分解转换为：R1⋈R2和R3的并行执行，再将两个操作结果进行连接。**

（2）集差运算不适合那种操作间并行策略？请简述理由。

**集差运算不适合流水线并行策略，因为各步骤的运行时间可能不均匀，导致该方法能达到的并行加速效果有限。**

3.基于划分的并行散列连接算法又进一步分为阻塞式和非阻塞式两种，请简述两种方式的执行过程，并解释各自阻塞或者不阻塞的原因。

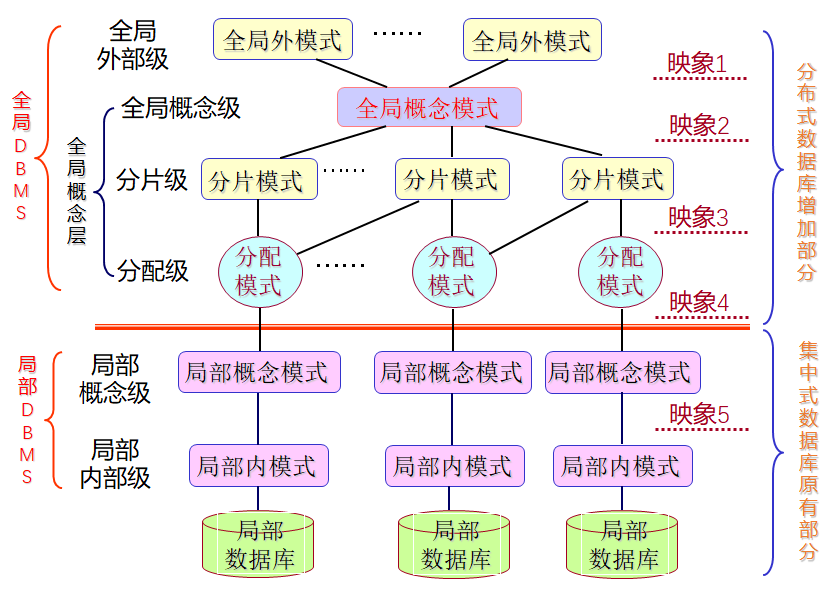
**基于划分的并行散列连接分为3个阶段（partition、build、probe）。其中，partition阶段将R表的数据划分到不同的分区上，build阶段在各个分区上构建哈希表，probe阶段先将S表中的元组映射到对应的分区上，再进行连接操作。**

**对于阻塞式算法，它在partition阶段多次扫描输入数据以生成分区，并通过多个步骤传递数据：（1）扫描R表并计算每个哈希键的元组数量的直方图在某个偏移量处的基数；（2）使用此直方图通过计算前缀和来确定输出偏移量；（3）再次扫描R并根据散列键进行分区。完成上述操作后才可以进入build阶段。**

**对于非阻塞式算法，它在build阶段不会被partition阶段阻塞。因为build阶段只关心hash表的产生，对于partition阶段输入的数据是否完整并不关心。此方法只扫描一次输入关系并即时生成输出，通过另外的线程去读取输入产生的hash表等结果，从而不必要等到所有的线程都结束扫描再进入下一个阶段。**

4.（1）简述分布式数据库系统的模式结构，并说明分布式数据库的映像和数据独立性；

**模式结构如下图所示：**



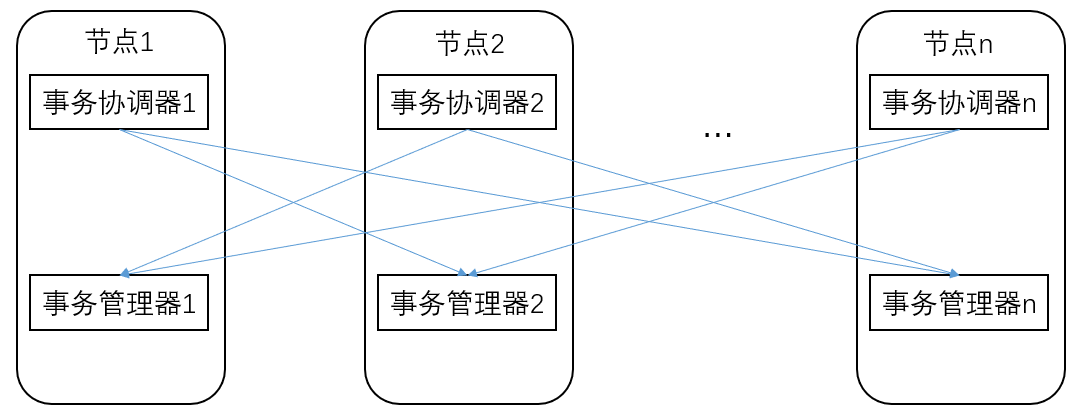
**分布式数据库系统包括局部DBMS和全局DBMS两个层面。对于局部数据库而言，每个局部数据库都拥有局部内模式与局部概念模式（分别是属于局部内部级和局部概念级），这构成了集中式数据库的原有部分。分布式数据库系统在局部概念级之上添加了分布式数据库部分，包括全局概念层（分配级——分配模式、分片级——分片模式、全局概念级——全局概念模式）和全局外部级——全局外模式。其中，来自多个不同局部数据库的分配模式构成分片模式，所有的分片模式构成全局概念模式，并由全局概念模式衍生出全局外模式。**



**分布式数据库的映像可以分为5层，数据独立性（分布透明性）包括分片透明性、位置透明性、局部数据模型透明性。其中分片透明性是指用户不必关心数据是如何分片的，它们对数据的操作在全局关系上进行，即如何分片对用户是透明的；位置透明性是指用户不必知道所操作的数据放在何处，即数据分配到哪些站点存储对用户是透明的；局部数据模型透明性是指用户不关心局部DBMS支持哪种数据模型、使用哪种数据操纵语言，数据模型和操纵语言的转换是由系统完成的。**

（2）简述分布式数据库的事务类型以及事务系统结构。

**分布式数据库的事务可分为两类：局部事务（local transaction）是指访问、更新局部分布式系统中的某个数据库就能完成的事务；全局事务（global transaction）是指需要访问、更新分布式数据库系统中的多个数据库才能完成的事务。**



**分布式事务系统结构如上图所示。其中每个节点都有各自独立的事务管理器，能保证节点内处理的事务具有ACID特性，节点内处理的事务可以是一个完整的局部事务，也可以是全局事务的一部分。而事务协调器负责协调节点间处理的不同事务。**

**具体来说，事务管理器负责多种功能：（1）维护日志，以便系统恢复；（2）参与并发控制机制，协调节点内同时运行的事务（在本地并发执行的事务）。**

**事务协调器负责协调节点发起的所有事务：（1）开启本地发起的事务；（2）将事务划分为多个子事务，并将这些子事务分配到合适的节点执行；（3）协调如何终止本地发起的事务，可能会使所有相关节点都提交或放弃该事务。**

5. 简述Aurora关系数据库多数协议下的读写规则。

**设数据有V个副本，则读操作至少会读Vr个副本，写操作至少会写Vw个副本，且满足：**

**（1）Vr + Vw > V，保证读操作一定能读到最新数据；**

**（2）Vw > V/2，保证不会出现写冲突。**

6. 简述谷歌的SpannerDB的数据模型，并说明它和BigTable数据模型的异同。

**Spanner定义了一种在一系列键值映射上的桶抽象，称为目录（directory），它是包含公共前缀的连续key的集合。一个目录是数据放置的基本单元，同一个目录的所有数据都具有相同的副本配置。一个Paxos组可以包含多个目录，多个频繁被一起访问的目录可以整合到一起，映射到一个tablet上。当数据在不同的Paxos组之间移动时，必须以目录为单位进行转移。**

**Spanner的另一种核心数据结构是表。表和关系数据库表类似，具备行、列和版本值。每个表有包含一个或多个列的主键，主键形成了行的名称，每个表都定义了从主键列到非主键列的映射。Spanner的模型属于半关系型模型，数据库根据客户端分割为一个表或者多个表的层次结构，类似于传统关系数据库中一对多关系。客户端应用使用INTERLEAVE IN语句在数据库模式中声明这个层次结构。该层次结构的上层的表是一个目录表，目录表中每行都具有键K，和子孙表中的所有以K开始(以字典顺序排序)的行一起，构成了一个目录。**

**Spanner与Bigtable的相同之处在于二者都基于键值对存储模型，将数据组织成行（row）和列（column）的形式，表中的数据都是根据行关键字按照字典顺序进行排序的。**

**二者的不同之处在于Spanner数据分布的基本单元是目录，而BigTable数据分布的基本单元是子表。除此之外，Spanner不再是类似BigTable的版本化key-value存储，而是“临时多版本”的数据库，数据存储在一个版本化的关系表里，存储的时间数据会根据其提交的时间打上时间戳，应用可以访问到较老的版本，老的版本也会被垃圾回收掉。**

**第七章**

1.（1）数据组织策略主要容易带来哪两方面的放大？引起这两方面的放大的主要原因各自是什么？

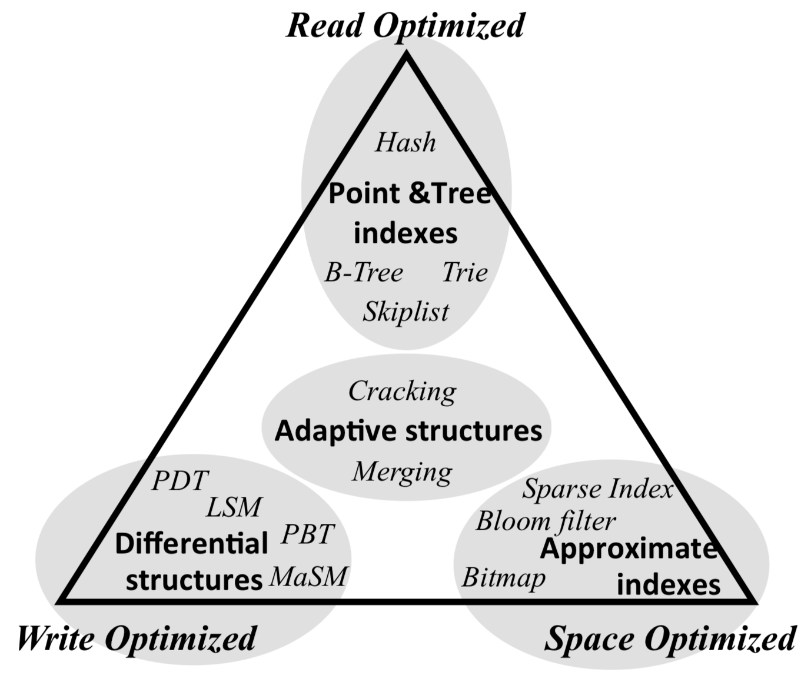
**主要会带来数据存储量的放大，以及数据访问量的放大。**

**引起数据存储量放大的主要原因是引入了额外数据，包括元数据、索引、日志、冗余等。**

**数据访问量放大包括读放大和写放大。引起数据访问量放大的主要原因是用户的数据访问大大增多，例如当系统中不存在索引时，用户需要用scan遍历的方式寻找目标数据。虽然索引可以明显减少额外的数据访问，但对索引的访问也会造成数据访问量的放大。**

（2）简述RUM（Read-Update-Memory）原理的含义。

**RUM原理主要是指在读放大、写放大、空间放大三个方向中，最多只能优化两个方向。例如hash表侧重于读优化，LSM树倾向于写优化，布隆过滤器倾向于空间优化（如下图所示）。**



2.（1）简述在常见的索引结构中的树、日志合并树（LSM树）是如何实现了写优化WOI策略的？其中树为何相对于B树的读性能下降了？

**WOI（Write-Optimized Index）是一种面向写优化而设计的索引结构。**

**对于树，它将B树的每个结点空间分为两部分，一部分按照原来的功能存储关键字和指向下层结点的指针；另一部分空间是缓冲区。树写入数据时，数据先有限缓存在当前结点的缓冲区中，采用追加写的方式，减少了原位更新，只有当缓冲区已满时才会将缓冲区的内容批量传递给下层结点。**

**对于日志合并树（LSM树），它结合了B树的结构，采用一种延迟更新、批量写入的日志内容的树形结构。写数据的步骤主要包括：（1）收到写请求，先把该条数据写到WAL Log用作故障恢复；（2）写完WAL Log后，把该条数据写入内存的SSTable中（删除对应墓碑标记，更新对应新记录一条的数据），也称Memtable（为了维持有序性，内存里可采用红黑树或者跳跃表相关的数据结构）；（3）当Memtable超过一定的大小后会在内存里冻结，变成不可变的Memtable，同时为了不阻塞写操作需要新生成一个Memtable继续提供服务；（4）把内存里面不可变的Memtable给dump到硬盘上的SSTable层中，此步骤也称为Minor Compaction；（5）当每层的磁盘上的SSTable的体积超过一定的大小或者个数，也会周期性进行合并，此步骤称为Major Compaction。该步骤会真正清除掉被标记删除的数据以回收空间。由于SSTable是有序的，可以采用merge sort进行高效合并。**

**由于树在缓冲区采取追加写的策略，因此会导致查找时开销增加，读性能相对于B树有所下降。**

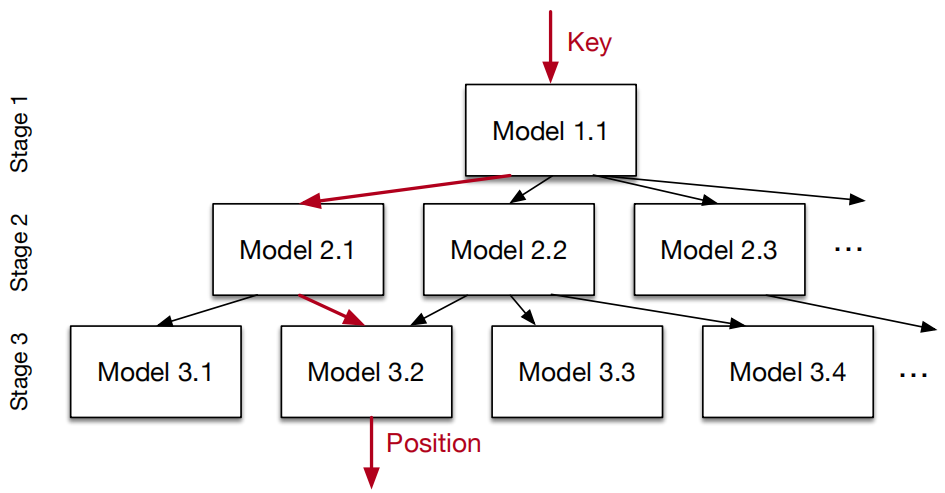
（2）写出LSM树的查询过程。

**①当收到读请求时先在内存里查询，如果查询到就返回；**

**②如果没有查询到就依次下沉，直到最坏情况下把所有的层查询一遍得到最终结果。**

3.（1）简述基于学习的索引RM-Index的设计思想；

**如下图所示，RM-Index是一种递归回归模型，它将整个预测过程划分成多个Stage，每一个Stage的Model基于Key作为Input，然后选择下一个Stage所对应的Model，依次递归，直到最终的一个Stage能够预测出Key的数据位置（在限定的误差范围内）。**



（2）写出RM-Index的训练过程。

**①固定整个RM-Index的结构，包括层数、每层Model数量等；**

**②用全部数据训练根节点，然后用根节点分类后的数据训练第二层模型，再用第二层分类后的数据训练第三层；**

**③对于第三层（叶节点），如果最大误差大于预设的阈值，就换成B树。**