数据库性能调优方法

阮祥炜[1]，童恒庆[2]
[1]武汉理工大学计算机科学技术系，武汉 (430070)
[2]武汉理工大学数学系，武汉 (430070)
E-mail：ruxiwei@163.com

摘　要：在当今高度信息化的社会，数据库在各个领域的使用越来越广泛，企业级应用系统的信息量和用户数迅猛增长，应用系统的对数据库提出了高性能的要求。然而，在企业级数据库应用系统的开发过程中，很多程序员仅关注系统功能的实现，在系统访问性能方面的考虑较少，要从大数据量的数据表种快速查询出相关数据已变得非常困难。基于这一问题，本文对提高大型数据库性能做出了一些探索和总结。

关键词：性能调优 数据库分区 RAID 临时表 索引 事务处理
中图分类号：TP392

1．引言

数据库调优可以使数据库应用运行得更快，它需要综合考虑各种复杂的因素。将数据均匀分布在磁盘上可以提高 I/O 利用率，提高数据的读写性能；适当程度的非规范化可以改善系统查询性能；建立索引和编写高效的 SQL 语句能有效避免低性能操作；通过锁的调优解决并发控制方面的性能问题。

数据库调优技术可以在不同的数据库系统中使用,它不必纠缠于复杂的公式和规则，然而它需要对程序的应用、数据库管理系统、查询处理、并发控制、操作系统以及硬件有广泛而深刻的理解。

2．计算机硬件调优

2.1 数据库对象的放置策略

利用数据库分区技术，均匀地把数据分布在系统的磁盘中，平衡 I/O 访问，避免 I/O 瓶颈：

（1）访问分散到不同的磁盘，即使用户数据尽可能跨越多个设备，多个 I/O 运转，避免 I/O 竞争，克服访问瓶颈；分别放置随机访问和连续访问数据。

（2）分离系统数据库 I/O 和应用数据库 I/O，把系统审计表和临时库表放在不忙的磁盘上。

（3）把事务日志放在单独的磁盘上，减少磁盘 I/O 开销，这还有利于在障碍后恢复，提高了系统的安全性。

（4）把频繁访问的“活性”表放在不同的磁盘上；把频繁用的表、频繁做 Join 的表分别放在单独的磁盘上，甚至把频繁访问的表的字段放在不同的磁盘上，把访问分散到不同的磁盘上，避免 I/O 争夺。

2.2 使用磁盘硬件优化数据库

RAID (独立磁盘冗余阵列)是由多个磁盘驱动器(一个阵列) 组成的磁盘系统。通过将磁盘阵列当作一个磁盘来对待，基于硬件的RAID允许用户管理多个磁盘。 使用基于硬件的RAID与基于操作系统的RAID相比较，基于硬件的RAID能够提供更佳的性能。 如果使用基于操作系统的RAID，那么它将占据其他系统需求的CPU周期；通过使用基于硬件的RAID，用户在不关闭系统的情况下能够替换发生故障的驱动器。

SQL Server 一般使用RAID 等级0、1 和5。

RAID 0 是传统的磁盘镜象，阵列中每一个磁盘都有一个或多个磁盘拷贝，它主要用来提供最高级的可靠性，使 RAID 0 成倍增加了写操作却可以并行处理多个读操作，从而提高了读操作的性能。RAID 1 是磁盘镜像或磁盘双工，能够为事务日志保证冗余性。RAID 5 是带奇偶的磁盘条带化，即将数据信息和校验信息分散到阵列的所有磁盘中，它可以消除一个

校验盘的瓶颈和单点失效问题， RAID 5 也会增加写操作，也可以并行处理一个读操作，还可以成倍地提高读操作的性能。

相比之下， RAID 5 增加的写操作比 RAID 0 增加的要少许多。 在实际应用中，用户的读操作要求远远多于写操作请求，而磁盘执行写操作的速度很快，以至于用户几乎感觉不到增加的时间，所以增加的写操作负担不会带来什么问题。 在性能较好的服务器中一般都会选择使用 RAID 5 的磁盘阵列卡来实现，对于性能相对差一些的服务器也可利用纯软件的方式来实现 RAID 5。

3. 关系系统与应用程序调优

3.1 应用程序优化

从数据库设计者的角度来看，应用程序无非是实现对数据的增加、修改、删除、查询和体现数据的结构和关系。设计者在性能方面的考虑因素，总的出发点是：把数据库当作奢侈的资源看待，在确保功能的同时，尽可能少地动用数据库资源。包括如下原则：

（1）不访问或少访问数据库；

（2）简化对数据库的访问；

（3）使访问最优；

（4）对前期及后续的开发、部署、调整提出要求，以协助实现性能目标。

另外，不要直接执行完整的 SQL 语法，尽量通过存储过程来调用 SQL Server。客户与服务器连接时，建立连接池，让连接尽量得以重用，以避免时间与资源的损耗。非到不得已，不要使用游标结构，确实使用时，注意各种游标的特性。

3.2 基本表设计优化

在基于表驱动的信息管理系统中，基本表的设计规范是第三范式。第三范式的基本特征是非主键属性只依赖于主键属性。基于第三范式的数据库表设计具有很多优点：一是能消除冗余数据、节省磁盘存储空间；二是有良好的数据完整性限制（基于主外键的参照完整限制和基于主键的实体完整性限制），这使得数据容易维护、移植和更新；三是数据的可逆性好，在做连接查询或者合并表时不遗漏、不重复；四是消除了冗余数据（这里主要指冗余列），使得查询时每个数据页存储的数据行增多，这样就有效地减少了逻辑 I/O，同时也减少了物理 I/O；五是对大多数事务而言，运行性能好；六是物理设计的机动性较大，能满足日益增长的用户需求。

基于第三范式设计的库表虽然有其优越性，然而在实际应用中有时不利于系统运行性能的优化：例如需要部分数据时而要扫描整表，许多过程同时竞争同一数据，反复用相同行计算相同的结果，过程从多表获取数据时引发大量的连接操作，当数据来源于多表时的连接操作；这都消耗了磁盘 I/O 和 CPU 时间。特别需要提出的是，在遇到下述情形时，我们要对基本表进行扩展设计优化：许多过程要频繁访问一个表、子集数据访问、重复计算和冗余数据，有时用户要求一些过程优先或低的响应时间，为避免以上不利因素，我们通常根据访问的频繁程度对相关表进行分割处理、存储冗余数据、存储衍生列、合并相关表处理，这些都是克服这些不利因素和优化系统运行的有效途径。

（1）分割表

分割表可分为水平分割表和垂直分割表两种：水平分割是按照行将一个表分割为多个表，这可以提高每个表的查询速度，但是由于造成了多表连接，所以应该在同时查询或更新不同分割表中的列的情况比较少的情况下使用。垂直分割是对于一个列很多的表，若某些列的访问频率远远高于其它列，在不破坏第三范式的前提下将主键和这些列作为一个表，将主键和其它列作为另外一个表。一种是当多个过程频繁访问表的不同列时，可将表垂直分成几个表，减少磁盘 I/O。通过减少列的宽度，增加了每个数据页的行数，一次 I/O 就可以扫描更多的行，从而提高了访问每一个表的速度。垂直分割表可以达到最大化利用 Cache 的目的。分割表的缺点是要在插入或删除数据时要考虑数据的完整性，用存储过程维护。

（2）存储衍生数据

对一些要做大量重复性计算的过程而言，若重复计算过程得到的结果相同，或计算牵扯多行数据需额外的磁盘 I/O 开销，或计算复杂需要大量的 C P U 时间，就考虑存储计算结果：

若在一行或多行进行重复性计算，就在表内增加列存储结果，但若参与计算的列被更新时，必须要用触发器或存储过程更新这个新列。总之，存储冗余数据有利于加快访问速度，但违反了第三范式，这会增加维护数据完整性的代价，必须用触发器立即更新、或存储过程更新，以维护数据的完整性。

3.3 修改应用技术模式

引入“中间表”的概念，在实际单据未进入核心业务流程前，采用“中间表”的技术思路，就是在实际用户操作过程中，实际操作的是一个临时表，在进行数据某个阶段审核（进入下一个环节）后，将临时表的数据写入正式表，并且删除临时表的数据，这样始终保持用户操作表的固定的数据量而且控制增长，可以定期清除。

采用临时表技术首先需将要操作的数据集插入到临时表中，这会给系统带来额外的开销。这里假设临时表中的数据集远小于源数据表中的数据集，因此在进行数据连接操作或对数据集进行频繁读操作时，系统的性能会提高几倍甚至几十倍不等。

并非所有情况都适宜用临时表技术。一般来说，下面两种情况适宜采用临时表技术进行处理：

（1）对数据量较大的表进行连接操作，并且连接操作的结果是一个小结果集。

（2）对数据量较大的表进行频繁访问，访问的范围比较固定且比较集中。

合理使用临时表技术，有助于提高应用系统对大数据表的实时处理的性能。

4．数据库索引优化

索引是建立在表上的一种数据组织，它能提高访问表中一条或多条记录的特定查询效率。利用索引优化系统性能是显而易见的，对所有常用于查询中的 Where 子句的列和所有用于排序的列创建索引，可以避免整表扫描或访问，在不改变表的物理结构的情况下，直接访问特定的数据列，这样可以减少数据存取时间；利用索引可以优化或排除耗时的分类操作，把数据分散到不同的页面上，这样就分散了插入的数据；主键自动建立了唯一索引，因此唯一索引也能确保数据的唯一性（即实体完整性）。总之，索引可以加快查询速度、减少 I/O 操作、消除磁盘排序。

优化索引可以避免扫描整个表，减少因查询造成的开销。一般说来建立索引要注意以下几点：

（1）检查被索引的列或组合索引的首列是否出现在 PL/SQL 语句的 WHERE 子句中，这是“执行计划”能用到相关索引的必要条件。比较一下列中唯一键的数量和表中记录的行数，就可以判断该列的可选择性。如果该列的“唯一键的数量/表中记录行数”的比值越接近于 1，则该列的可选择行越高。在可选择性高的列上进行查询，返回的数据就较少，比较适合索引查询。相反，比如性别列上只有两个值，可选择行就很小，不适合索引查询。因此，在查询中经常作为条件表达式且不同值较多的列上建立索引，不同值较少的列上不要建立索引。

（2）索引的创建也是需要代价的，对于删除、某些更新、插入操作，对于每个索引都要进行相应的删除、更新、插入操作。从而导致删除、某些更新、插入操作的效率变低。 因此频繁进行删除、插入操作的表不要建立过多的索引。

（3）查询经常用到的列上建立非聚簇索引，在频繁进行范围查询、排序、分组的列上建立聚簇索引。

（4）对于不存在重复值的列，创建唯一索引优于创建非唯一索引。

（5）当数据库表更新大数据后， 删除并重新建立索引来提高查询速度。

（6）当对一个表的 update 操作远远多于 select 操作时， 不应创建索引。

（7）如果索引列是函数的参数，则索引在查询时用不上，该列也不适合索引。

（8）Hash Join （HJ）由于须做 HASH 运算，索引的存在对数据查询速度几乎没有影响。

（9）在主键上建立索引，尤其当经常用它作为连接的时候；在经常用于连接而又未指定为外键的列上建立索引。

(10)经常同时存取多列，且每列都含有重复值，可以考虑建立复合索引来覆盖一个或一组查询，并且把查询引用最频繁的列作为前导列。

（11）尽使用较窄的索引，这样数据页每页上能因存放较多的索引行而减少操作。
（12）并行查询将不会用到索引。
（13）索引中存储值不能为全空。
（14）查询中较少用到的列、数据量较大的列均不应建立索引。

5．SQL 语句优化

在完成了系统设计、索引设计等工作以后， 就要考虑在使用过程中对语句的设计了。影响数据库应用程序性能的一个重要因素是 SQL 语句，按其影响严重程度，依次可分为：无谓的 SQL，拙劣的 SQL，复杂的 SQL。

无谓的 SQL：它们对数据库的访问，并不存在技术、技能上的问题，但却不是必要的，超出了实际业务需求。其结果是浪费了宝贵的主机资源、占用了网络流量，降低了系统性能。

拙劣的 SQL：它们对数据库的访问并不是多余的，所体现的业务逻辑或结果是正确的，但是“写法”不够好，导致数据库处理起来不够优化。

复杂的 SQL：数据库中多表（或视图）关联，条件复杂、冗长，计算复杂，使用冷僻的 SQL 技术等。

其中，无谓的 SQL 和拙劣的 SQL 属于开发技能方面的问题；复杂的 SQL 属于设计技能方面的问题，设计到数据库的结构。

在使用结构化查询语言来执行查询时，推荐以下举措：

（1）择运算应尽可能先做，并在对同一个表进行多个选择运算时， 选择影响较大的语句放在前面；较弱的选择条件写在后面， 这样就可以先根据较严格的条件得出数据较小的信息， 再在这些信息中根据后面较弱的条件得到满足条件的信息。

（2）应避免使用相关子查询。把子查询转换成联结来实现。对于主查询的每一条记录子查询都要执行一次， 嵌套的层次越多效率越低。避免对子句使用数学运算符。即不要对数据表的属性列进行操作。SQL 概念上将位于 WHERE 子句中的相关子查询，处理成获取参数并且返回一个单独的值或值的集合的函数。因为子查询要对应位于外层查询的每一个元组进行单独的计算。从而导致大量的随机磁盘 I/O 操作。所以在实际应用中若可以用连接代替的子查询， 则用连接实现。例如，有以下相关子查询语句：

```
SELECT ProductName FROM Products WHERE EXISTS
(SELECT * FROM OrderDetails
WHERE  Discount >= 25 AND  Products.ProductID= OrderDetails.ProjectID)；
```

用连接查询实现如下：

```
SELECT  ProductName  FROM  Products， OrderDetails
WHERE Discount >= 25 AND Products.ProductID= OrderDetails.ProjectID
```

（3）字段提取按照“ 需多少， 提多少” 的原则， 避免“SELECT *”。“SELECT *”需要数据库返回相应表的所有列信息， 这对于一个列较多的表无疑是一项费时的操作。

(4)避免使用!=(或<>)、IS NULL 或 IS NOT NULL、IN、NOT IN 等这样的操作符，避免在 WHERE 子句中使用非聚合表达式。这些操作符会使系统无法使用索引， 而只能直接搜索表中的数据。例如， SELECT id， name FROM employee WHERE id!=B%

优化器将无法通过索引来确定将要命中的行数， 因此需要搜索该表的所有行。

(5)避免使用 OR， 用 UNION 代替。OR 语句的执行原理并不是利用列上的索引根据每个语句分别查找再将结果求并集， 而是先取出满足每个 OR 子句的行， 存入临时数据库的工作表中， 再建立唯一索引以去掉重复行， 最后从这个临时表中计算结果。这样使用可能造成索引失效， 导致顺序扫描整个表， 大大降低查询效率。

（6）在执行连接前对关系作适当的预处理， 预处理的方法有两种， 在连接属性上建立索引和对关系进行排序。

（7）将一个大的查询拆成多步执行查询。

（8）如果应用程序使用循环,可考虑在查询内放入循环。

6．事务处理调优

数据库的日常运行过程中可能面临多个用户同时对数据库的并发操作带来的数据不一致的问题，如：丢失更新、脏读和不可重复读等。并发控制的主要方法是封锁，锁就是在一

段时间内禁止用户做某些操作以避免产生数据不一致。

数据库应用程序将其工作分成若干个事务进行处理。当一个事务执行时，它访问数据库并执行一些本地计算。开发人员可以假设每一个事务都会被隔离地执行---没有任何并发动作。因为隔离的概念提供了透明性，这种对事务处理方式的保证有时被称为原子性保证。但是，如果把应用程序中的事务序列作为一个整体来看，则并没有上面所说的那种保证。在一个应用程序执行的两个事务之间，可能会执行另外一个应用程序的事务，而且第二个应用程序的执行可能修改了第一个应用程序中的两个事务（或其中的一个）需要访问的数据项。因此，事务的长度对保证正确性有着重要影响。

尽管将事务切分成较小粒度可以提高执行效率，但会因此破坏执行的正确性。这种性能和正确性之间的矛盾充斥并发控制的整个调优过程。考虑事务的性能我们要考虑到：事务使用的锁的个数（在所有其他条件相同的情况下，使用的锁个数越少，性能越好）；锁的类型（读锁对性能更有利）；事务持有锁的时间长短（持有时间越短，性能越好）。关于锁的调优有以下建议：

（1）使用特殊的系统程序来处理长的读操作。对于一个只读的事务 R 来说，它“看到”的数据库的状态一直是事务 R 开始时的状态。只读查询可以不需要封锁开销，在不造成阻塞和死锁的情况下，只读的查询可以与其他对同一数据进行更新的较小的事务并行地执行。

（2）消除不必要的封锁。只有一个事务执行时，或所有事务都是只读事务时，用户应利用配置选项减少锁的个数，从而减小锁管理模块的内存开销和执行封锁操作的处理时间开销。

（3）根据事务的内容将事务切分成较小的事务。事务所要求的锁越多，它需要等待其他事务释放某个锁的可能就越大。事务 T 执行的时间越长，被 T 阻塞的事务等待的时间可能就越长。因此，在可能发生阻塞的情况下，利用较短的事务较好。

（4）在应用程序允许的情况下，适当降低隔离级别。

（5）选择适当的封锁粒度。页级封锁阻止并发事务访问或修改该页面上所有记录，表级封锁阻止并发事务访问或修改表内所有的页面；记录级封锁（行级锁）比页级封锁粒度好，页级封锁比表级封锁粒度好。长事务（指要访问表内几乎所有页面的事务）应该尽可能使用表级封锁来防止死锁，而短事务应该使用记录级封锁来提高并发度。

（6）只在数据库很少被访问时才修改有关数据定义的数据（系统目录或元数据）。每个能够编译、添加或删除表、添加或删除索引、改变属性定义的事务都必须访问目录数据，因此，目录很容易成为热点，也因而成为瓶颈。

（7）减少访问热点（大量事务访问和更新的数据）。只有在更新某热点的事务完成滞后，其他的事务才能获得这个热点上的锁，因此热点可能成为瓶颈。

（8）死锁检测周期的调优。

以上每个建议都可以独立于其他建议来运用，但是在调优时必须检测是否能体现合适的隔离性保证。

7．总结

数据库性能优化的基本原则就是通过尽可能少的磁盘访问获得所需要的数据。本文从计算机硬件、关系系统与应用程序、数据库索引、SQL 语句、事务处理几个比较共性的方面分析了数据库性能优化的问题，提出了若干数据库性能优化的策略。当然实现优化的方法还有很多，要根据具体情况而定。对于不同的应用情况,我们应该具体情况具体分析，各方面优化措施综合运用，以使数据库性能得到提高。数据库应用系统的性能是一项全民工程，开发团队的所有人都有责任为性能做贡献，树立性能意识，使之成为日常工作的习惯而不是单独成为某一阶段的工作，要未雨绸缪，不要寄希望于某一个环节的工作。

参考文献

[1] Dennis Shasha, Philippe Bonnet 著．数据库调优原理与技术．孟小峰、李站怀等译，北京：电子工业出版社，2004 年．

[2] EDWARD WHALEN 著．SQL Server2000 性能调优技术指南．武欣、何畅、罗云峰译，北京：机械工业出版社，2005 年．

[3] 陈庆芳，方芳．优化数据库的设计方法．通信与广播电视，2006 年，第 2 期：26-34．

[4] 任玉辉，王成良．企业级应用系统中的数据库性能优化策略．计算机应用，2008 年，第 3 期：7-14．

[5] 沙有闯．浅析关系数据库性能优化技术．中国西部科技，2008 年，第 33 期：30-32．
[6] 年纬．数据库应用系统性能优化的几个策略．计算机与信息技术，经验与交流：85-86．
[7] 王书海，张婧．基于 SQL Server 应用系统的大量数据实时处理技术．实验室研究与探索，2008 年，第 3 期：16-18．
[8] 胡百敬．SQL Server 性能调教．北京：电子工业出版社，2008 年．

The Methods Of Database Performance Tuning

Ruan Xiangwei[1]，Tong Hengqing[2]
1 Department of Computer Science, Wuhan University of Technology, Wuhan, PRC, (430070)
2 Department of Mathematics, Department of Mathematics, Wuhan, PRC, (430070)

Abstract

In today's highly information-oriented society, the database is getting more and more widespread in each field. The increase of information in enterprise application, as well as the greatly growth in the number of users, has now required a high performance to the database. However, in the process of developing the enterprise application, lots of programmers pay more attention to the realization than the performance. It’s difficult to retrieval related data from a mass data table. Considering this problem, we propose some new ideas and give a summarization on the tuning of database performance.

Keywords: *Performance Tuning, Database partition, RAID, Temporary table, index, Transaction Processing*