作者：andyao  
原文link: http://andyao.iteye.com/admin/show/144033  
转载请留名  
  
**1. 简介**   
在Web应用程序体系架构中，数据持久层（通常是一个关系数据库）是关键的核心部分，它对系统的性能有非常重要的影响。MySQL是目前使用最多的开源数据库，但是MySQL数据库的默认设置性能非常的差，仅仅是一个玩具数据库。因此在产品中使用MySQL数据库必须进行必要的优化。  
优化是一个复杂的任务，本文描述MySQL相关的数据库设计和查询优化，服务器端优化，存储引擎优化。  
  
**2. 数据库设计和查询优化**   
在MySQL Server性能调优中，首先要考虑的就是Database Schema设计，这一点是非常重要的。一个糟糕的Schema设计即使在性能调优的MySQL Server上运行，也会表现出很差的性能；和Schema相似，查询语句的设计也会影响MySQL的性能，应该避免写出低效的SQL查询。这一节将详细讨论这两方面的优化。  
  
2.1 Schema Design   
Schema的优化取决于将要运行什么样的query，不同的query会有不同的Schema优化方案。2.2节将介绍Query Design的优化。Schema设计同样受到预期数据集大小的影响。Schema设计时主要考虑：标准化，数据类型，索引。  
  
2.1.1 标准化   
  
标准化是在数据库中组织数据的过程。其中包括，根据设计规则创建表并在这些表间建立关系；通过取消冗余度与不一致相关性，该设计规则可以同时保护数据并提高数据的灵活性。通常数据库标准化是让数据库设计符合某一级别的范式，通常满足第三范式即可。也有第四范式（也称为 Boyce Codd范式，BCNF)）与第五范式存在，但是在实际设计中很少考虑。忽视这些规则可能使得数据库的设计不太完美，但这不应影响功能。  
标准化的特点：

1) 所有的“对象”都在它自己的table中，没有冗余。  
2) 数据库通常由E-R图生成。  
3) 简洁，更新属性通常只需要更新很少的记录。  
4) Join操作比较耗时。  
5) Select，sort优化措施比较少。  
6) 适用于OLTP应用。

非标准化的特点：

1) 在一张表中存储很多数据，数据冗余。  
2) 更新数据开销很大，更新一个属性可能会更新很多表，很多记录。  
3) 在删除数据是有可能丢失数据。  
4) Select，order有很多优化的选择。  
5) 适用于DSS应用。

标准化和非标准化都有各自的优缺点，通常在一个数据库设计中可以混合使用，一部分表格标准化，一部分表格保留一些冗余数据：

1) 对OLTP使用标准化，对DSS使用非标准化  
2) 使用物化视图。MySQL不直接支持该数据库特性，但是可以用MyISAM表代替。  
3) 冗余一些数据在表格中，例如将ref\_id和name存在同一张表中。但是要注意更新问题。  
4) 对于一些简单的对象，直接使用value作为建。例如IP address等  
5) Reference by PRIMARY/UNIQUE KEY。MySQL可以优化这种操作，例如：

java 代码

1. select city\_name
2. from city,state
3. where state\_id=state.id and state.code=‘CA’” converted to “select city\_name from city where state\_id=12

2.1.2 数据类型   
最基本的优化之一就是使表在磁盘上占据的空间尽可能小。这能带来性能非常大的提升，因为数据小，磁盘读入较快，并且在查询过程中表内容被处理所占用的内存更少。同时，在更小的列上建索引，索引也会占用更少的资源。  
可以使用下面的技术可以使表的性能更好并且使存储空间最小：

1) 使用正确合适的类型，不要将数字存储为字符串。  
2) 尽可能地使用最有效(最小)的数据类型。MySQL有很多节省磁盘空间和内存的专业化类型。  
3) 尽可能使用较小的整数类型使表更小。例如，MEDIUMINT经常比INT好一些，因为MEDIUMINT列使用的空间要少25%。  
4) 如果可能，声明列为NOT NULL。它使任何事情更快而且每列可以节省一位。注意如果在应用程序中确实需要NULL，应该毫无疑问使用它，只是避免 默认地在所有列上有它。  
5) 对于MyISAM表，如果没有任何变长列(VARCHAR、TEXT或BLOB列)，使用固定尺寸的记录格式。这比较快但是不幸地可能会浪费一些空间。即使你已经用CREATE选项让VARCHAR列ROW\_FORMAT=fixed，也可以提示想使用固定长度的行。  
6) 使用sample character set，例如latin1。尽量少使用utf-8，因为utf-8占用的空间是latin1的3倍。可以在不需要使用utf-8的字段上面使用latin1，例如mail，url等。

2.1.3 索引   
所有MySQL列类型可以被索引。对相关列使用索引是提高SELECT操作性能的最佳途径。使用索引应该注意以下几点：

1) MySQL只会使用前缀，例如key(a, b) …where b=5 将使用不到索引。  
2) 要选择性的使用索引。在变化很少的列上使用索引并不是很好，例如性别列。  
3) 在Unique列上定义Unique index。  
4) 避免建立使用不到的索引。  
5) 在Btree index中（InnoDB使用Btree），可以在需要排序的列上建立索引。  
6) 避免重复的索引。  
7) 避免在已有索引的前缀上建立索引。例如：如果存在index（a，b）则去掉index（a）。  
8) 控制单个索引的长度。使用key（name（8））在数据的前面几个字符建立索引。  
9) 越是短的键值越好，最好使用integer。  
10) 在查询中要使用到索引（使用explain查看），可以减少读磁盘的次数，加速读取数据。  
11) 相近的键值比随机好。Auto\_increment就比uuid好。  
12) Optimize table可以压缩和排序index，注意不要频繁运行。  
13) Analyze table可以更新数据。

2.2 Designing queries   
查询语句的优化是一个Case by case的问题，不同的sql有不同的优化方案，在这里我只列出一些通用的技巧。

1) 在有index的情况下，尽量保证查询使用了正确的index。可以使用EXPLAIN select …查看结果，分析查询。  
2) 查询时使用匹配的类型。例如select \* from a where id=5， 如果这里id是字符类型，同时有index，这条查询则使用不到index，会做全表扫描，速度会很慢。正确的应该是 … where id=”5” ，加上引号表明类型是字符。  
3) 使用--log-slow-queries –long-query-time=2查看查询比较慢的语句。然后使用explain分析查询，做出优化。

**3. 服务器端优化**3.1 MySQL安装   
MySQL有很多发行版本，最好使用MySQL AB发布的二进制版本。也可以下载源代码进行编译安装，但是编译器和类库的一些bug可能会使编译完成的MySQL存在潜在的问题。  
如果安装MySQL的服务器使用的是Intel公司的处理器，可以使用intel c++编译的版本，在Linux World2005的一篇PPT中提到，使用intel C++编译器编译的MySQL查询速度比正常版本快30%左右。Intel c++编译版本可以在MySQL官方网站下载。  
  
3.2 服务器设置优化   
MySQL默认的设置性能很差，所以要做一些参数的调整。这一节介绍一些通用的参数调整，不涉及具体的存储引擎（主要指MyISAM，InnoDB，相关优化在4中介绍）。

--character-set：如果是单一语言使用简单的character set例如latin1。尽量少用Utf-8，utf-8占用空间较多。  
--memlock：锁定MySQL只能运行在内存中，避免swapping，但是如果内存不够时有可能出现错误。  
--max\_allowed\_packet：要足够大，以适应比较大的SQL查询，对性能没有太大影响，主要是避免出现packet错误。  
--max\_connections：server允许的最大连接。太大的话会出现out of memory。  
--table\_cache：MySQL在同一时间保持打开的table的数量。打开table开销比较大。一般设置为512。  
--query\_cache\_size： 用于缓存查询的内存大小。  
--datadir：mysql存放数据的根目录，和安装文件分开在不同的磁盘可以提高一点性能。

**4. 存储引擎优化**   
MySQL支持不同的存储引擎，主要使用的有MyISAM和InnoDB。  
  
4.1 MyISAM   
MyISAM管理非事务表。它提供高速存储和检索，以及全文搜索能力。MyISAM在所有MySQL配置里被支持，它是默认的存储引擎，除非配置MySQL默认使用另外一个引擎。  
  
4.1.1 MyISAM特性   
4.1.1.1 MyISAM Properties

1) 不支持事务，宕机会破坏表  
2) 使用较小的内存和磁盘空间  
3) 基于表的锁，并发更新数据会出现严重性能问题  
4) MySQL只缓存Index，数据由OS缓存

4.1.1.2 Typical MyISAM usages

1) 日志系统  
2) 只读或者绝大部分是读操作的应用  
3) 全表扫描  
4) 批量导入数据  
5) 没有事务的低并发读/写

4.1.2 MyISAM优化要点

1) 声明列为NOT NULL，可以减少磁盘存储。  
2) 使用optimize table做碎片整理，回收空闲空间。注意仅仅在非常大的数据变化后运行。  
3) Deleting/updating/adding大量数据的时候禁止使用index。使用ALTER TABLE t DISABLE KEYS。  
4) 设置myisam\_max\_[extra]\_sort\_file\_size足够大，可以显著提高repair table的速度。

4.1.3 MyISAM Table Locks

1) 避免并发insert，update。  
2) 可以使用insert delayed，但是有可能丢失数据。  
3) 优化查询语句。  
4) 水平分区。  
5) 垂直分区。  
6) 如果都不起作用，使用InnoDB。

4.1.4 MyISAM Key Cache

1) 设置key\_buffer\_size variable。MyISAN最主要的cache设置，用于缓存MyISAM表格的index数据，该参数只对MyISAM有影响。通常在只使用MyISAM的Server中设置25-33%的内存大小。  
2) 可以使用几个不同的Key Caches（对一些hot data）。

a) SET GLOBAL test.key\_buffer\_size=512\*1024;  
b) CACHE INDEX t1.i1, t2.i1, t3 IN test;

2) Preload index到Cache中可以提高查询速度。因为preloading index是顺序的，所以非常快。

a) LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES；

4.2 InnoDB  
InnoDB给MySQL提供了具有提交，回滚和崩溃恢复能力的事务安全（ACID兼容）存储引擎。InnoDB提供row level lock，并且也在SELECT语句提供一个Oracle风格一致的非锁定读。这些特色增加了多用户部署和性能。没有在InnoDB中扩大锁定的需要，因为在InnoDB中row level lock适合非常小的空间。InnoDB也支持FOREIGN KEY约束。在SQL查询中，你可以自由地将InnoDB类型的表与其它MySQL的表的类型混合起来，甚至在同一个查询中也可以混合。  
InnoDB是为在处理巨大数据量时获得最大性能而设计的。它的CPU使用效率非常高。  
InnoDB存储引擎已经完全与MySQL服务器整合，InnoDB存储引擎为在内存中缓存数据和索引而维持它自己的缓冲池。 InnoDB存储它的表＆索引在一个表空间中，表空间可以包含数个文件（或原始磁盘分区）。这与MyISAM表不同，比如在MyISAM表中每个表被存在分离的文件中。InnoDB 表可以是任何大小，即使在文件尺寸被限制为2GB的操作系统上。  
许多需要高性能的大型数据库站点上使用了InnoDB引擎。著名的Internet新闻站点Slashdot.org运行在InnoDB上。 Mytrix, Inc.在InnoDB上存储超过1TB的数据，还有一些其它站点在InnoDB上处理平均每秒800次插入/更新的负荷。  
4.2.1 InnoDB特性   
4.2.1.1 InnoDB Properties

1) 支持事务，ACID，外键。  
2) Row level locks。   
3) 支持不同的隔离级别。  
4) 和MyISAM相比需要较多的内存和磁盘空间。  
5) 没有键压缩。  
6) 数据和索引都缓存在内存hash表中。

4.2.1.2 InnoDB Good For

1) 需要事务的应用。  
2) 高并发的应用。  
3) 自动恢复。  
4) 较快速的基于主键的操作。

4.2.2 InnoDB优化要点

1) 尽量使用short，integer的主键。  
2) Load/Insert数据时按主键顺序。如果数据没有按主键排序，先排序然后再进行数据库操作。  
3) 在Load数据是为设置SET UNIQUE\_CHECKS=0，SET FOREIGN\_KEY\_CHECKS=0，可以避免外键和唯一性约束检查的开销。  
4) 使用prefix keys。因为InnoDB没有key压缩功能。

4.2.3 InnoDB服务器端设定

innodb\_buffer\_pool\_size：这是InnoDB最重要的设置，对InnoDB性能有决定性的影响。默认的设置只有8M，所以默认的数据库设置下面InnoDB性能很差。在只有InnoDB存储引擎的数据库服务器上面，可以设置60-80%的内存。更精确一点，在内存容量允许的情况下面设置比InnoDB tablespaces大10%的内存大小。  
  
innodb\_data\_file\_path：指定表数据和索引存储的空间，可以是一个或者多个文件。最后一个数据文件必须是自动扩充的，也只有最后一个文件允许自动扩充。这样，当空间用完后，自动扩充数据文件就会自动增长（以8MB为单位）以容纳额外的数据。例如： innodb\_data\_file\_path=/disk1/ibdata1:900M;/disk2/ibdata2:50M:autoextend两个数据文件放在不同的磁盘上。数据首先放在ibdata1中，当达到900M以后，数据就放在ibdata2中。一旦达到50MB，ibdata2将以8MB为单位自动增长。如果磁盘满了，需要在另外的磁盘上面增加一个数据文件。

innodb\_autoextend\_increment: 默认是8M, 如果一次insert数据量比较多的话, 可以适当增加.  
  
innodb\_data\_home\_dir：放置表空间数据的目录，默认在mysql的数据目录，设置到和MySQL安装文件不同的分区可以提高性能。  
  
innodb\_log\_file\_size：该参数决定了recovery speed。太大的话recovery就会比较慢，太小了影响查询性能，一般取256M可以兼顾性能和recovery的速度  
。  
innodb\_log\_buffer\_size：磁盘速度是很慢的，直接将log写道磁盘会影响InnoDB的性能，该参数设定了log buffer的大小，一般4M。如果有大的blob操作，可以适当增大。  
  
innodb\_flush\_logs\_at\_trx\_commit=2： 该参数设定了事务提交时内存中log信息的处理。

1) =1时，在每个事务提交时，日志缓冲被写到日志文件，对日志文件做到磁盘操作的刷新。Truly ACID。速度慢。  
2) =2时，在每个事务提交时，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新。只有操作系统崩溃或掉电才会删除最后一秒的事务，不然不会丢失事务。  
3) =0时， 日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新。任何mysqld进程的崩溃会删除崩溃前最后一秒的事务

innodb\_file\_per\_table：可以存储每个InnoDB表和它的索引在它自己的文件中。  
  
transaction-isolation=READ-COMITTED: 如果应用程序可以运行在READ-COMMITED隔离级别，做此设定会有一定的性能提升。  
  
innodb\_flush\_method： 设置InnoDB同步IO的方式：

1) Default – 使用fsync（）。  
2) O\_SYNC 以sync模式打开文件，通常比较慢。  
3) O\_DIRECT，在Linux上使用Direct IO。可以显著提高速度，特别是在RAID系统上。避免额外的数据复制和double buffering（mysql buffering 和OS buffering）。

innodb\_thread\_concurrency： InnoDB kernel最大的线程数。

1) 最少设置为(num\_disks+num\_cpus)\*2。  
2) 可以通过设置成1000来禁止这个限制

**5. 缓存**   
缓存有很多种，为应用程序加上适当的缓存策略会显著提高应用程序的性能。由于应用缓存是一个比较大的话题，所以这一部分还需要进一步调研。   
  
**6. Reference**   
1) http://www.mysqlperformanceblog.com/  
2) Advanced MySQL Performance Optimization, Peter Zaitsev, Tobias Asplund, MySQL Users Conference 2005  
3) Improving MySQL Server Performance with Intel C++ Compiler，Peter Zaitsev，Linux World 2005  
4) MySQL Performance Optimization, Peter Zaitsev, Percona Ltd, OPEN SOURCE DATABASE CONFERENCE 2006  
5) MySQL Server Settings Tuning, Peter Zaitsev, co-founder, Percona Ltd, 2007  
6) MySQL Reference Manual