MySQL优化

概述

MySQL优化分为三部分优化：

1. MySQL服务器和配置优化
2. 数据库设计和结构优化
3. **查询优化（重点）**

MySQL服务器和配置优化

1. 硬件方面
    - **配置较大的内存**
    - **选择高速磁盘系统**
    - 合理分布磁盘I/O
    - **选择多处理器系统**
    - **网络环境选择**
2. 操作系统内核
    - **使用64位操作系统，更好地使用大内存**
    - 设置noatimenodiratime
    - **优化内核参数**
    - 加大文件描述符限制
    - **文件系统选择 xfs**
3. MySQL参数配置

    MySQL的配置参数都在my.ini或则my.cnf文件中。一些对性能影响较大的参数如下所示，包括

    - I/O处理的常用参数；
    - 最大连接数设置；
    - 缓存使用参数的设置；
    - 慢日志的参数的设置；
    - InnoDB相关参数的设置等。

```
key_buffer_size
MySQL索引缓冲区的大小。如果是采用MyISAM的话要重点设置这个参数，根据
 (key_reads/key_read_requests) 判断

max_connections
服务器允许的最大连接数，尽量不要设置太大，因为设置太大的话容易导致内存溢出

wait_timeout
线程连接的超时时间，尽量不要设置很大，推荐10s

thread_concurrency
线程并发利用数量，(cpu+disk)*2，根据(os中显示的请求队列和tickets)判断
```

```
sort_buffer_size
排序缓冲区的大小。该值越大，进行排序的速度越快

read_rnd_buffer_size
当根据键进行分类操作时获得更快的--ORDER BY

join_buffer_size
join连接使用全表扫描连接的缓冲大小，根据select_full_join判断

read_buffer_size
全表扫描时为查询预留的缓冲大小，根据select_scan判断

tmp_table_size
临时内存表的设置，如果超过设置就会转化成磁盘表，根据参数(created_tmp_disk_tables)判断

innodb_buffer_pool_size
InnoDB表数据和索引的最大缓存。根据 (hit riatos和FILE I/O) 判断

innodb_log_file_size(默认5M)
InnoDB引擎的redo log文件，设置较大的值意味着较长的恢复时间

innodb_flush_log_at_trx_commit(默认1)
表示何时将缓冲区数据写入日志，并将日志文件写入磁盘。
1. 0表示每秒进行一次log写入cache，并flush log到磁盘；
2. 1表示在每次事务提交后执行log写入cache，并flush log到磁盘；
3. 2表示在每次事务提交后执行log写入cache，每秒执行一次flush log到磁盘；
```

数据库设计和结构优化

1. 数据表尽量符合范式**
   - [1] 1NF：表的列的具有原子性，不可再分解，即列的信息，不能分解成为更小的数据项；
   - [2] 2NF：表中的记录是唯一的，通常通过设计一个主键来实现；
   - [3] 3NF：表中不要有冗余数据，也就是说，如果某个字段能够被其它字段推导出来，就不应该存在；
2. **合理选择存储引擎**

   在开发中，经常使用的存储引擎有MyISAM/InnoDB/Memory。
   - MyISAM： 数据库并发不大，读多写少，sql 语句比较简单，对事务要求不高，比如 bbs 中的发帖表，回复表；
   - **InnoDB**：并发访问大，写操作比较多，有外键、事务等需求的应用，系统内存较大。比如订单表，账号表；
   - Memory：数据不需要入库，同时又频繁的查询和修改，速度极快；
3. **字段数据类型选择**

   字段数据类型的选择的一般原则：根据需求选择合适的字段类型，在满足需求的情况下字段类型尽可能小；只分配满足需求的最小字符数，不要太慷慨；原因：更小的字段类型更小的字符数占用更少的内存，占用更少的磁盘空间，占用更少的磁盘 IO 以及占用更少的带宽。
4. **CHAR 和 VARCHAR 的选取**

它们的区别在于：CHAR(M) 类型的数据列里，每个值都占用 M 个字节，如果某个长度小于 M，MySQL 就会在它的右边用空格字符补足；VARCHAR(M) 类型的数据列里，每个值只占用刚好够用的字节再加上一个用来记录其长度的字节（即总长度为 L+1 字节）。

二者的选取主要依据如下原则：

1.如果列数据项的大小一致或者相差不大，则使用 CHAR；

2.如果列数据项的大小差异相当大，则使用 VARCHAR；

3.对于 MyISAM 表，尽量使用 CHAR。 因为 VARCHAR 类型随着修改容易造成磁盘碎片，使用 CHAR 的缺点就是占用磁盘空间；

4.对于 InnoDB 表，从减少空间占用量和减少磁盘 I/O 的角度，使用 VARCHAR;5.表中只要存在一个 VARCHAR 类型的字段，那么所有的 CHAR 字段都会自动变成 VARCHAR 类型，因此建议定长和变长的数据分开。

5. 编码选择

   单字节：latin1

   多字节：utf8 (汉字占 3 个字节，英文字母占用一个字节)

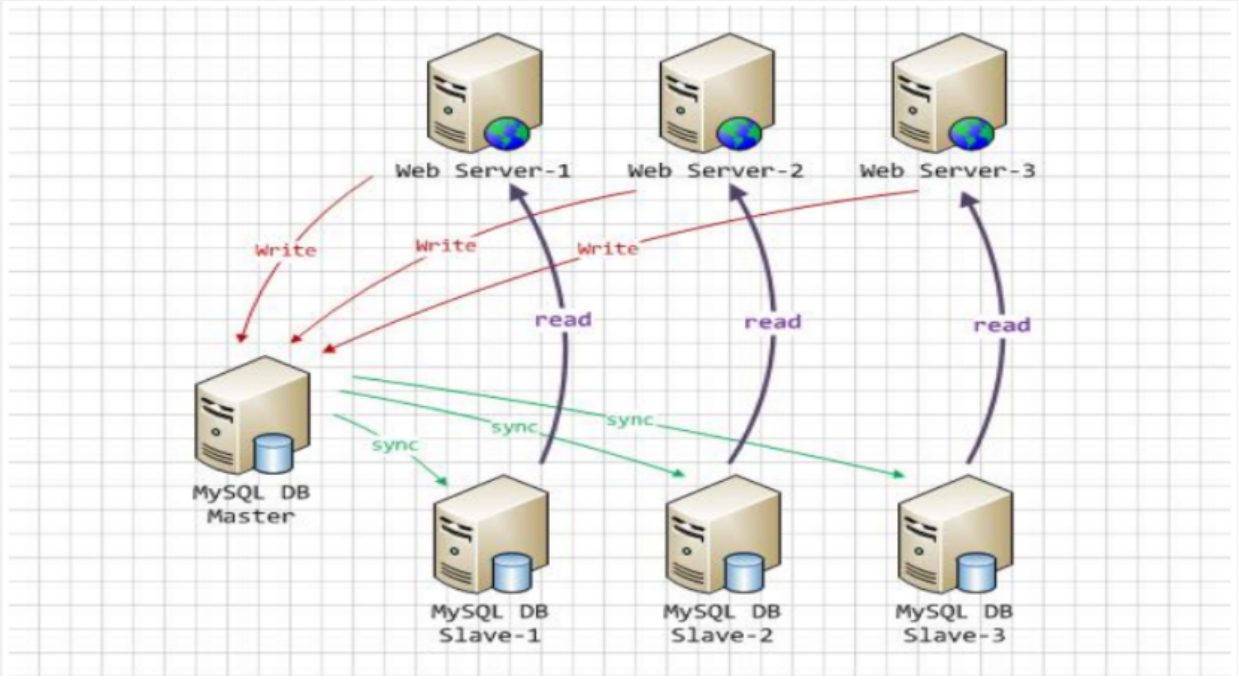
   注意：如果含有中文字符的话最好都统一采用 utf8 类型，避免乱码的情况发生。

6. **NULL OR NOT NULL**

   尽可能设置每个字段为 NOT NULL，除非有特殊的需求，原因如下：使用含有 NULL 列做索引的话会占用更多的磁盘空间，因为索引 NULL 列需要额外的空间来保存；进行比较的时候，程序会更复杂；含有 NULL 的列 SQL 难优化，如果是一个组合索引，那么这个 NULL 类型的字段会极大影响整个索引的效率。

7. **读写分离**

   开启 MySQL 复制，实现读写分离、负载均衡（nginx、lvs），将读的负载分摊到多个从服务器上，提高服务器的处理能力。



集群和分布式理解：

集群是指多台服务器提供同一种服务，避免单点失效，可靠性高。

分布式是指将一个应用拆分成多个服务，每个服务单独提供自己服务，提高并发能力。

8. **分表和分区**

当表存储了百万级乃至千万级条记录时，在查询和插入的时候耗时太长，性能低下，如果涉及联合查询的情况，性能会更加糟糕。分表和分区的目的就是减少数据库的负担，提高数据库的效率。

分表可以分成水平拆分和垂直拆分两种

水平拆分就是将一张大表拆分成多张小表，每张小表的结构与原表相同，只是记录条数要少；

t_order 100万条数据

t_order_01 10万条数据，0-99999

t_order_02 10万条数据，100000-199999

…

t_order_10 10万条数据，900000-999999

垂直拆分就是将大表中的某些字段（使用频率低）取出来，放在另一张表中。

id	name	sex	age	address
1	abc	1	25	广东深圳
2	asd	0	22	广东东莞

id	name	sex	age
1	abc	1	25
2	asd	0	22

id	address
1	广东深圳
2	广东东莞

分区是指在同一个表分区，分为水平分区和垂直分区两种。

水平分区有几种模式如下：

1.Range：定义范围来分

2.Hash：定义Hash来分

3.Key：Hash的一种延伸

4.List（预定义列表）：自己定义几个值来分

5.Composite（复合模式）：对1234组合使用

垂直分区：一般将大text和BLOB列分到另一个区

```
CREATE TABLE part_table
(
c1 int default NULL,
c2 varchar(30) default NULL,
c3 date default NULL
) engine=myisam
PARTITION BY RANGE (year(c3))
(
PARTITION p0 VALUES LESS THAN (1995),
PARTITION p1 VALUES LESS THAN (1996) , PARTITION p2 VALUES LESS THAN (1997) ,
PARTITION p3 VALUES LESS THAN (1998) , PARTITION p4 VALUES LESS THAN (1999) ,
PARTITION p5 VALUES LESS THAN (2000) , PARTITION p6 VALUES LESS THAN (2001) ,
PARTITION p7 VALUES LESS THAN (2002) , PARTITION p8 VALUES LESS THAN (2003) ,
PARTITION p9 VALUES LESS THAN (2004) , PARTITION p10 VALUES LESS THAN (2010),
PARTITION p11 VALUES LESS THAN MAXVALUE
);
```

查询优化

1. **对查询进行优化，要尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。**
2. **应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描**，

   如：select id from t where num is null

   - 最好不要给数据库留NULL，尽可能的使用 NOT NULL填充数据库
   - 备注、描述、评论之类的可以设置为 NULL，其他的最好不要使用NULL
   - 不要以为 NULL 不需要空间，比如：char(100) 型，在字段建立时，空间就固定了， 不管是否插入值（NULL也包含在内），都是占用 100个字符的空间的，如果是varchar这样的变长字段， null 不占用空间
   - 在num上设置默认值0，确保表中num列没有null值，然后这样查询：select id from t where num = 0
3. **应尽量避免在 where 子句中使用 != 或 <> 操作符，否则将引擎放弃使用索引而进行全表扫描。**

   比如：sex要么是1要么是0，当查询sex = 0的数据

   这种写法不用索引：select * from table where sex != 1;

   这种用了索引：select * from table where sex = 0;
4. **应尽量避免在 where 子句中使用 or 来连接条件，如果一个字段有索引，一个字段没有索引，将导致引擎放弃使用索引而进行全表扫描**，

   如：select id from t where num=10 or Name = 'admin'

   可以这样查询：select id from t where num = 10 union all select id from t where Name = 'admin'
5. **in 和 not in 也要慎用，否则会导致全表扫描**，

   如：select id from t where num in(1,2,3)

   对于连续的数值，能用 between 就不要用 in 了：

   select id from t where num between 1 and 3

   很多时候用 exists 代替 in 是一个好的选择：

   select num from a where num in(select num from b)（a表数据多而b的数据少，in的效率高）

用下面的语句替换：

select num from a where exists(select 1 from b where num=a.num)（a表数据少而b的数据多，exists的效率高）

6. **下面的查询也将导致全表扫描（不能前置百分号）**：

   select id from t where name like'%abc%'若要提高效率，可以考虑全文检索。

7. 如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。

   如下面语句将进行全表扫描：select id from t where num = @num可以改为强制查询使用索引：select id from t with(index(索引名)) where num = @num

8. **应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。**

   如：select id from t where num/2 = 100

   应改为:select id from t where num = 100*2

9. 应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。

   如：--name以abc开头的id

   select id from t where substring(name,1,3) = 'abc'--'2005-11-30'

   --生成的id

   select id from t where datediff(day,createdate,'2005-11-30') = 0

   应改为:select id from t where name like 'abc%'

   select id from t where createdate >= '2005-11-30' and createdate < '2005-12-1'

10. 不要在 where 子句中的"="左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

11. 在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

12. 不要写一些没有意义的查询，如需要生成一个空表结构：select col1,col2 into #t from t where 1=0这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：create table #t(...)

13. Update 语句，如果只更改1、2个字段，不要Update全部字段，否则频繁调用会引起明显的性能消耗，同时带来大量日志。

14. **对于多张大数据量（这里几百条就算大了）的表JOIN，要先分页再JOIN，否则逻辑读会很高，性能很差。**

15. select count(*) from table；这样不带任何条件的count会引起全表扫描，并且没有任何业务意义，是一定要杜绝的。如果要在 InnoDB下使用，建议另外弄一张统计表，采用 MyISAM，定期做统计。一般的对统计的数据不会要求太精准的情况下适用。

16. 索引并不是越多越好，索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。

17. 应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

18. 尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连 接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

19. 尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。
20. **任何地方都不要使用 select * from t ，用具体的字段列表代替“*”，不要返回用不到的任何字段。**
21. 尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。
22. 避免频繁创建和删除临时表，以减少系统表资源的消耗。临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件， 最好使用导出表。
23. 在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。
24. 如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。
25. 尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。
26. 使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。
27. 与临时表一样，游标并不是不可使用。对小型数据集使用 FAST_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时 间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。
28. 在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE_IN_PROC 消息。
29. **尽量避免大事务操作，提高系统并发能力**
30. **尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。**

慢日志

```
show variables like 'slow_query_log';
set global slow_query_log_file = 'D:/mysql/log/slow.log'; (windows无效)
set global log_queries_not_using_indexes = on;
set global long_query_time = 1;
set global slow_query_log=on;

执行sql的主机信息: # User@Host: root[root] @ localhost [::1]  Id:     3
sql执行信息: # Query_time: 0.012032  Lock_time: 0.011532 Rows_sent: 2  Rows_examined: 2
sql执行时间: SET timestamp=1511101457;
sql执行内容: select * from store limit 10;
```

案例

实际案例分析拆分大的 DELETE 或INSERT 语句，批量提交SQL语句。

如果你需要在一个在线的网站上去执行一个大的 DELETE 或 INSERT 查询，你需要非常小心，要避免你的操作让你的整个网站停止相应。因为这两个操作是会锁表的，表一锁住了，别的操作都进不来了。

Apache 会有很多的子进程或线程。所以，其工作起来相当有效率，而我们的服务器也不希望有太多的子进程，线程和数据库链接，这是极大的占服务器资源的事情，尤其是内存。

如果你把你的表锁上一段时间，比如30秒钟，那么对于一个有很高访问量的站点来说，这30秒所积累的访问进程/线程，数据库链接，打开的文件数，可能不仅仅会让你的WEB服务崩溃，还可能会让你的整台服务器马上挂了。

所以，如果你有一个大的处理，你一定把其拆分，使用 LIMIT oracle(rownum),sqlserver(top)条件是一个好的方法。下面是一个mysql示例：

```
while(1){

     //每次只做1000条
     mysql_query("delete from logs where log_date <= '2012-11-01' limit 1000");

     if(mysql_affected_rows() == 0){

         //删除完成，退出!
         break;
    }

//每次暂停一段时间，释放表让其他进程/线程访问。
usleep(50000)

}
```