MySQL/Oracle数据库优化总结（非常全面）

> ## MySQL数据库优化的八种方式(经典必看)
>
> ## 引言：
>
> 1. 关于数据库优化，网上有不少资料和方法，但是不少质量参差不齐，有些总结的不够到位，内容冗杂
> 2. 偶尔发现了这篇文章，总结得很经典，文章流量也很大，所以拿到自己的总结文集中，积累优质文章，提升个人能力，希望对大家今后开发中也有帮助

1、选取最适用的字段属性

MySQL可以很好的支持大数据量的存取，但是一般说来，数据库中的表越小，在它上面执行的查询也就会越快。因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

例如，在定义邮政编码这个字段时，如果将其设置为CHAR(255),显然给数据库增加了不必要的空间，甚至使用VARCHAR这种类型也是多余的，因为CHAR(6)就可以很好的完成任务了。同样的，如果可以的话，我们应该使用MEDIUMINT而不是BIGIN来定义整型字段。

另外一个提高效率的方法是在可能的情况下，应该尽量把字段设置为NOTNULL，这样在将来执行查询的时候，数据库不用去比较NULL值。
对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为ENUM类型。因为在MySQL中，ENUM类型被当作数值型数据来处理，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

2、使用连接（JOIN）来代替子查询(Sub-Queries)

MySQL从4.1开始支持SQL的子查询。这个技术可以使用SELECT语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。例如，我们要将客户基本信息表中没有任何订单的客户删除掉，就可以利用子查询先从销售信息表中将所有发出订单的客户ID取出来，然后将结果传递给主查询，如下所示：

DELETEFROMcustomerinfo

WHERECustomerIDNOTin(SELECTCustomerIDFROMsalesinfo)

使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的SQL操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接（JOIN）..替代。例如，假设我们要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

SELECT*FROMcustomerinfo

WHERECustomerIDNOTin(SELECTCustomerIDFROMsalesinfo)

如果使用连接（JOIN）..来完成这个查询工作，速度将会快很多。尤其是当salesinfo表中对CustomerID建有索引的话，性能将会更好，查询如下：

SELECT*FROMcustomerinfo

LEFTJOINsalesinfoONcustomerinfo.CustomerID=salesinfo.CustomerID

WHEREsalesinfo.CustomerIDISNULL

连接（JOIN）..之所以更有效率一些，是因为MySQL不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作。

3、使用联合(UNION)来代替手动创建的临时表

MySQL从4.0的版本开始支持union查询，它可以把需要使用临时表的两条或更多的select查询合并的一个查询中。在客户端的查询会话结束的时候，临时表会被自动删除，从而保证数据库整齐、高效。使用union来创建查询的时候，我们只需要用UNION作为关键字把多个select语句连接起来就可以了，要注意的是所有select语句中的字段数目要想同。下面的例子就演示了一个使用UNION的查询。

SELECTName,PhoneFROMclientUNION

SELECTName,BirthDateFROMauthorUNION

SELECTName,SupplierFROMproduct

4、事务

尽管我们可以使用子查询（Sub-Queries）、连接（JOIN）和联合（UNION）来创建各种各样的查询，但不是所有的数据库操作都可以只用一条或少数几条SQL语句就可以完成的。更多的时候是需要用到一系列的语句来完成某种工作。但是在这种情况下，当这个语句块中的某一条语句运行出错的时候，整个语句块的操作就会变得不确定起来。设想一下，要把某个数据同时插入两个相关联的表中，可能会出现这样的情况：第一个表中成功更新后，数据库突然出现意外状况，造成第二个表中的操作没有完成，这样，就会造成数据的不完整，甚至会破坏数据库中的数据。要避免这种情况，就应该使用事务，它的作用是：要么语句块中每条语句都操作成功，要么都失败。换句话说，就是可以保持数据库中数据的一致性和完整性。事物以BEGIN关键字开始，COMMIT关键字结束。

在这之间的一条SQL操作失败，那么，ROLLBACK命令就可以把数据库恢复到BEGIN开始之前的状态。

BEGIN;
INSERTINTOsalesinfoSETCustomerID=14;UPDATEinventorySETQuantity=11WHEREitem='bo

事务的另一个重要作用是当多个用户同时使用相同的数据源时，它可以利用锁定数据库的方法来为用户提供一种安全的访问方式，这样可以保证用户的操作不被其它的用户所干扰。

5、锁定表

尽管事务是维护数据库完整性的一个非常好的方法，但却因为它的独占性，有时会影响数据库的性能，尤其是在很大的应用系统中。由于在事务执行的过程中，数据库将会被锁定，因此其它的用户请求只能暂时等待直到该事务结束。如果一个数据库系统只有少数几个用户来使用，事务造成的影响不会成为一个太大的问题；但假设有成千上万的用户同时访问一个数据库系统，例如访问一个电子商务网站，就会产生比较严重的响应延迟。

其实，有些情况下我们可以通过锁定表的方法来获得更好的性能。下面的例子就用锁定表的方法来完成前面一个例子中事务的功能。

LOCKTABLEinventoryWRITESELECTQuantityFROMinventoryWHEREItem='book';

...

UPDATEinventorySETQuantity=11WHEREItem='book';UNLOCKTABLES

这里，我们用一个select语句取出初始数据，通过一些计算，用update语句将新值更新到表中。包含有WRITE关键字的LOCKTABLE语句可以保证在UNLOCKTABLES命令被执行之前，不会有其它的访问来对inventory进行插入、更新或者删除的操作。

6、使用外键

锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键。

例如，外键可以保证每一条销售记录都指向某一个存在的客户。在这里，外键可以把customerinfo表中的CustomerID映射到salesinfo表中CustomerID，任何一条没有合法CustomerID的记录都不会被更新或插入到salesinfo中。

1. CREATETABLEcustomerinfo(
   CustomerIDINTNOTNULL,PRIMARYKEY(CustomerID))TYPE=INNODB;

```
2.
3. CREATETABLEsalesinfo( SalesIDINTNOTNULL,CustomerIDINTNOTNULL,
4.
5. PRIMARYKEY(CustomerID,SalesID),
6.
7. FOREIGNKEY(CustomerID)REFERENCEScustomerinfo(CustomerID)ONDELETECASCADE)
```

注意例子中的参数“ONDELETECASCADE”。该参数保证当customerinfo表中的一条客户记录被删除的时候，salesinfo表中所有与该客户相关的记录也会被自动删除。如果要在MySQL中使用外键，一定要记住在创建表的时候将表的类型定义为事务安全表InnoDB类型。该类型不是MySQL表的默认类型。定义的方法是在CREATETABLE语句中加上TYPE=INNODB。如例中所示。

7、使用索引

索引是提高数据库性能的常用方法，它可以令数据库服务器以比没有索引快得多的速度检索特定的行，尤其是在查询语句当中包含有MAX(),MIN()和ORDERBY这些命令的时候，性能提高更为明显。

那该对哪些字段建立索引呢?

一般说来，索引应建立在那些将用于JOIN,WHERE判断和ORDERBY排序的字段上。尽量不要对数据库中某个含有大量重复的值的字段建立索引。对于一个ENUM类型的字段来说，出现大量重复值是很有可能的情况

例如customerinfo中的“province”..字段，在这样的字段上建立索引将不会有什么帮助；相反，还有可能降低数据库的性能。我们在创建表的时候可以同时创建合适的索引，也可以使用ALTERTABLE或CREATEINDEX在以后创建索引。此外，MySQL从版本3.23.23开始支持全文索引和搜索。全文索引在MySQL中是一个FULLTEXT类型索引，但仅能用于MyISAM类型的表。对于一个大的数据库，将数据装载到一个没有FULLTEXT索引的表中，然后再使用ALTERTABLE或CREATEINDEX创建索引，将是非常快的。但如果将数据装载到一个已经有FULLTEXT索引的表中，执行过程将会非常慢。

8、优化的查询语句

绝大多数情况下，使用索引可以提高查询的速度，但如果SQL语句使用不恰当的话，索引将无法发挥它应有的作用。

下面是应该注意的几个方面。

- 首先，最好是在相同类型的字段间进行比较的操作。

> 在MySQL3.23版之前，这甚至是一个必须的条件。例如不能将一个建有索引的INT字段和BIGINT字段进行比较；但是作为特殊的情况，在CHAR类型的字段和VARCHAR类型字段的字段大小相同的时候，可以将它们进行比较。

- 其次，在建有索引的字段上尽量不要使用函数进行操作。

> 例如，在一个DATE类型的字段上使用YEAE()函数时，将会使索引不能发挥应有的作用。所以，下面的两个查询虽然返回的结果一样，但后者要比前者快得多。

- 第三，在搜索字符型字段时，我们有时会使用LIKE关键字和通配符，这种做法虽然简单，但却也是以牺牲系统性能为代价的。

例如下面的查询将会比较表中的每一条记录。

```
1.
2. SELECT*FROMbooks
3.
4. WHEREnamelike"MySQL%"
```

但是如果换用下面的查询，返回的结果一样，但速度就要快上很多：

```
1.
2. SELECT*FROMbooks
3.
4. WHEREname > ="MySQL"andname < "MySQM"
```

最后，应该注意避免在查询中让MySQL进行自动类型转换，因为转换过程也会使索引变得不起作用。

优化Mysql数据库的8个方法

本文通过8个方法优化Mysql数据库：创建索引、复合索引、索引不会包含有NULL值的列、使用短索引、排序的索引问题、like语句操作、不要在列上进行运算、不使用NOT IN和<>操作

1、创建索引

对于查询占主要的应用来说，索引显得尤为重要。很多时候性能问题很简单的就是因为我们忘了添加索引而造成的，或者说没有添加更为有效的索引导致。如果不加索引的话，那么查找任何哪怕只是一条特定的数据都会进行一次全表扫描，如果一张表的数据量很大而符合条件的结果又很少，那么不加索引会引起致命的性能下降。但是也不是什么情况都非得建索引不可，比如性别可能就只有两个值，建索引不仅没什么优势，还会影响到更新速度，这被称为过度索引。

2、复合索引

比如有一条语句是这样的：select * from users where area='beijing' and age=22;

如果我们是在area和age上分别创建单个索引的话，由于mysql查询每次只能使用一个索引，所以虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在area、age两列上创建复合索引的话将带来更高的效率。如果我们创建了(area, age, salary)的复合索引，那么其实相当于创建了(area,age,salary)、(area,age)、(area)三个索引，这被称为最佳左前缀特性。因此我们在创建复合索引时应该将最常用作限制条件的列放在最左边，依次递减。

3、索引不会包含有NULL值的列

只要列中包含有NULL值都将不会被包含在索引中，复合索引中只要有一列含有NULL值，那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为NULL。

4、使用短索引

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的 列，如果在前10 个或20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

5、排序的索引问题

mysql查询只使用一个索引，因此如果where子句中已经使用了索引的话，那么order by中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

6、like语句操作

一般情况下不鼓励使用like操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。

7、不要在列上进行运算

select * from users where YEAR(adddate)<2007;

将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成

select * from users where adddate< ‘2007-01-01';

8、不使用NOT IN和<>操作

NOT IN和<>操作都不会使用索引将进行全表扫描。NOT IN可以NOT EXISTS代替，id<>3则可使用id>3 or id<3来代替。

数据库SQL优化大总结之 百万级数据库优化方案

网上关于SQL优化的教程很多，但是比较杂乱。近日有空整理了一下，写出来跟大家分享一下，其中有错误和不足的地方，还请大家纠正补充。

这篇文章我花费了大量的时间查找资料、修改、排版，希望大家阅读之后，感觉好的话推荐给更多的人，让更多的人看到、纠正以及补充。

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。

可以在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。

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)
```

用下面的语句替换：

```
select num from a where exists(select 1 from b where num=a.num)
```

6.下面的查询也将导致全表扫描：

```
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
```

.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：

```
select id from t where substring(name,1,3) = ’abc’       --name以abc开头的id
select id from t where datediff(day,createdate,’2005-11-30’) = 0    --‘2005-11-30’    --生成的id
```

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。

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。

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)

}
```

好了，到这里就写完了。我知道还有很多没有写到的，还请大家补充。后面有空会介绍一些SQL优化工具给大家。让我们一起学习，一起进步吧！

运维角度浅谈MySQL数据库优化

一个成熟的数据库架构并不是一开始设计就具备高可用、高伸缩等特性的，它是随着用户量的增加，基础架构才逐渐完善。这篇博文主要谈MySQL数据库发展周期中所面临的问题及优化方案，暂且抛开前端应用不说，大致分为以下五个阶段：

1、数据库表设计

项目立项后，开发部根据产品部需求开发项目，开发工程师工作其中一部分就是对表结构设计。对于数据库来说，这点很重要，如果设计不当，会直接影响访问速度和用户体验。影响的因素很多，比如慢查询、低效的查询语句、没有适当建立索引、数据库堵塞（死锁）等。当然，有测试工程师

的团队，会做压力测试，找bug。对于没有测试工程师的团队来说，大多数开发工程师初期不会太多考虑数据库设计是否合理，而是尽快完成功能实现和交付，等项目有一定访问量后，隐藏的问题就会暴露，这时再去修改就不是这么容易的事了。

2、数据库部署

该运维工程师出场了，项目初期访问量不会很大，所以单台部署足以应对在1500左右的QPS（每秒查询率）。考虑到高可用性，可采用MySQL主从复制+Keepalived做双击热备，常见集群软件有Keepalived、Heartbeat。

双机热备博文：http://lizhenliang.blog.51cto.com/7876557/1362313

3、数据库性能优化

如果将MySQL部署到普通的X86服务器上，在不经过任何优化情况下，MySQL理论值正常可以处理2000左右QPS，经过优化后，有可能会提升到2500左右QPS，否则，访问量当达到1500左右并发连接时，数据库处理性能就会变慢，而且硬件资源还很富裕，这时就该考虑软件问题了。那么怎样让数据库最大化发挥性能呢？一方面可以单台运行多个MySQL实例让服务器性能发挥到最大化，另一方面是对数据库进行优化，往往操作系统和数据库默认配置都比较保守，会对数据库发挥有一定限制，可对这些配置进行适当的调整，尽可能的处理更多连接数。

具体优化有以下三个层面：

3.1 数据库配置优化

MySQL常用有两种存储引擎，一个是MyISAM，不支持事务处理，读性能处理快，表级别锁。另一个是InnoDB，支持事务处理（ACID），设计目标是为处理大容量数据发挥最大化性能，行级别锁。

表锁：开销小，锁定粒度大，发生死锁概率高，相对并发也低。

行锁：开销大，锁定粒度小，发生死锁概率低，相对并发也高。

为什么会出现表锁和行锁呢？主要是为了保证数据的完整性，举个例子，一个用户在操作一张表，其他用户也想操作这张表，那么就要等第一个用户操作完，其他用户才能操作，表锁和行锁就是这个作用。否则多个用户同时操作一张表，肯定会数据产生冲突或者异常。

根据以上看来，使用InnoDB存储引擎是最好的选择，也是MySQL5.5以后版本中默认存储引擎。每个存储引擎相关联参数比较多，以下列出主要影响数据库性能的参数。

公共参数默认值：

```
1 max_connections = 151
2 #同时处理最大连接数，推荐设置最大连接数是上限连接数的80%左右
3 sort_buffer_size = 2M
```

```
#查询排序时缓冲区大小，只对order by和group by起作用，可增大此值为16M
open_files_limit = 1024
#打开文件数限制，如果show global status like 'open_files'查看的值等于或者大于open_files_limit值时，程序会无法连接数据库或卡死
```

MyISAM参数默认值：

```
key_buffer_size = 16M
#索引缓存区大小，一般设置物理内存的30-40%
read_buffer_size = 128K
#读操作缓冲区大小，推荐设置16M或32M
query_cache_type = ON
#打开查询缓存功能
query_cache_limit = 1M
#查询缓存限制，只有1M以下查询结果才会被缓存，以免结果数据较大把缓存池覆盖
query_cache_size = 16M
#查看缓冲区大小，用于缓存SELECT查询结果，下一次有同样SELECT查询将直接从缓存池返回结果，可适当成倍增加此值
```

InnoDB参数默认值：

```
innodb_buffer_pool_size = 128M
#索引和数据缓冲区大小，一般设置物理内存的60%-70%
innodb_buffer_pool_instances = 1
#缓冲池实例个数，推荐设置4个或8个
innodb_flush_log_at_trx_commit = 1
#关键参数，0代表大约每秒写入到日志并同步到磁盘，数据库故障会丢失1秒左右事务数据。1为每执行一条SQL后写入到日志并同步到磁盘，I/O开销大，执行完SQL要等待日志读写，效率低。2代表只把日志写入到系统缓存区，再每秒同步到磁盘，效率很高，如果服务器故障，才会丢失事务数据。对数据安全性要求不是很高的推荐设置2，性能高，修改后效果明显。
innodb_file_per_table = OFF
#默认是共享表空间，共享表空间idbdata文件不断增大，影响一定的I/O性能。推荐开启独立表空间模式，每个表的索引和数据都存在自己独立的表空间中，可以实现单表在不同数据库中移动。
innodb_log_buffer_size = 8M
#日志缓冲区大小，由于日志最长每秒钟刷新一次，所以一般不用超过16M
```

3.2 系统内核优化

大多数MySQL都部署在linux系统上，所以操作系统的一些参数也会影响到MySQL性能，以下对linux内核进行适当优化。

```
net.ipv4.tcp_fin_timeout = 30
#TIME_WAIT超时时间，默认是60s
net.ipv4.tcp_tw_reuse = 1
#1表示开启复用，允许TIME_WAIT socket重新用于新的TCP连接，0表示关闭
```

```
net.ipv4.tcp_tw_recycle = 1
#1表示开启TIME_WAIT socket快速回收，0表示关闭
net.ipv4.tcp_max_tw_buckets = 4096
#系统保持TIME_WAIT socket最大数量，如果超出这个数，系统将随机清除一些TIME_WAIT并打印警告信息
net.ipv4.tcp_max_syn_backlog = 4096
#进入SYN队列最大长度，加大队列长度可容纳更多的等待连接
```

在linux系统中，如果进程打开的文件句柄数量超过系统默认值1024，就会提示“too many files open”信息，所以要调整打开文件句柄限制。

```
# vi /etc/security/limits.conf  #加入以下配置，*代表所有用户，也可以指定用户，重启系统生效
* soft nofile 65535
* hard nofile 65535
# ulimit -SHn 65535  #立刻生效
```

3.3 硬件配置

加大物理内存，提高文件系统性能。linux内核会从内存中分配出缓存区（系统缓存和数据缓存）来存放热数据，通过文件系统延迟写入机制，等满足条件时（如缓存区大小到达一定百分比或者执行sync命令）才会同步到磁盘。也就是说物理内存越大，分配缓存区越大，缓存数据越多。当然，服务器故障会丢失一定的缓存数据。

SSD硬盘代替SAS硬盘，将RAID级别调整为RAID1+0，相对于RAID1和RAID5有更好的读写性能（IOPS），毕竟数据库的压力主要来自磁盘I/O方面。

4、数据库架构扩展

随着业务量越来越大，单台数据库服务器性能已无法满足业务需求，该考虑加机器了，该做集群了~~~。主要思想是分解单台数据库负载，突破磁盘I/O性能，热数据存放缓存中，降低磁盘I/O访问频率。

4.1 主从复制与读写分离

因为生产环境中，数据库大多都是读操作，所以部署一主多从架构，主数据库负责写操作，并做双击热备，多台从数据库做负载均衡，负责读操作，主流的负载均衡器有LVS、HAProxy、Nginx。

怎么来实现读写分离呢？大多数企业是在代码层面实现读写分离，效率比较高。另一个种方式通过代理程序实现读写分离，企业中应用较少，常见代理程序有MySQL Proxy、Amoeba。在这样数据库集群架构中，大大增加数据库高并发能力，解决单台性能瓶颈问题。如果从数据库一台从库能处理2000 QPS，那么5台就能处理1w QPS，数据库横向扩展性也很容易。

有时，面对大量写操作的应用时，单台写性能达不到业务需求。如果做双主，就会遇到数据库数据不一致现象，产生这个原因是在应用程序不同的用户会有可能操作两台数据库，同时的更新操作造

成两台数据库数据库数据发生冲突或者不一致。在单库时MySQL利用存储引擎机制表锁和行锁来保证数据完整性，怎样在多台主库时解决这个问题呢？有一套基于perl语言开发的主从复制管理工具，叫MySQL-MMM（Master-Master replication managerfor Mysql，Mysql主主复制管理器），这个工具最大的优点是在同一时间只提供一台数据库写操作，有效保证数据一致性。

主从复制博文：http://lizhenliang.blog.51cto.com/7876557/1290431

读写分离博文：http://lizhenliang.blog.51cto.com/7876557/1305083

MySQL-MMM博文：http://lizhenliang.blog.51cto.com/7876557/1354576

4.2 增加缓存

给数据库增加缓存系统，把热数据缓存到内存中，如果缓存中有要请求的数据就不再去数据库中返回结果，提高读性能。缓存实现有本地缓存和分布式缓存，本地缓存是将数据缓存到本地服务器内存中或者文件中。分布式缓存可以缓存海量数据，扩展性好，主流的分布式缓存系统有memcached、redis，memcached性能稳定，数据缓存在内存中，速度很快，QPS可达8w左右。如果想数据持久化就选择用redis，性能不低于memcached。

工作过程：

4.3 分库

分库是根据业务不同把相关的表切分到不同的数据库中，比如web、bbs、blog等库。如果业务量很大，还可将切分后的库做主从架构，进一步避免单个库压力过大。

4.4 分表

数据量的日剧增加，数据库中某个表有几百万条数据，导致查询和插入耗时太长，怎么能解决单表压力呢？你就该考虑是否把这个表拆分成多个小表，来减轻单个表的压力，提高处理效率，此方式称为分表。

分表技术比较麻烦，要修改程序代码里的SQL语句，还要手动去创建其他表，也可以用merge存储引擎实现分表，相对简单许多。分表后，程序是对一个总表进行操作，这个总表不存放数据，只有一些分表的关系，以及更新数据的方式，总表会根据不同的查询，将压力分到不同的小表上，因此提高并发能力和磁盘I/O性能。

分表分为垂直拆分和水平拆分：

垂直拆分：把原来的一个很多字段的表拆分多个表，解决表的宽度问题。你可以把不常用的字段单独放到一个表中，也可以把大字段独立放一个表中，或者把关联密切的字段放一个表中。

水平拆分：把原来一个表拆分成多个表，每个表的结构都一样，解决单表数据量大的问题。

4.5 分区

分区就是把一张表的数据根据表结构中的字段（如range、list、hash等）分成多个区块，这些区块可以在一个磁盘上，也可以在不同的磁盘上，分区后，表面上还是一张表，但数据散列在多个位置，这样一来，多块硬盘同时处理不同的请求，从而提高磁盘I/O读写性能，实现比较简单。

注：增加缓存、分库、分表和分区主要由程序猿来实现。

5、数据库维护

数据库维护是运维工程师或者DBA主要工作，包括性能监控、性能分析、性能调优、数据库备份和恢复等。

5.1 性能状态关键指标

QPS，Queries Per Second：每秒查询数，一台数据库每秒能够处理的查询次数

TPS，Transactions Per Second：每秒处理事务数

通过show status查看运行状态，会有300多条状态信息记录，其中有几个值帮可以我们计算出QPS和TPS，如下：

Uptime：服务器已经运行的实际，单位秒

Questions：已经发送给数据库查询数

Com_select：查询次数，实际操作数据库的

Com_insert：插入次数

Com_delete：删除次数

Com_update：更新次数

Com_commit：事务次数

Com_rollback：回滚次数

那么，计算方法来了，基于Questions计算出QPS：

```
1  mysql> show global status like 'Questions';
2  mysql> show global status like 'Uptime';
```

QPS = Questions / Uptime

基于Com_commit和Com_rollback计算出TPS：

```
mysql> show global status like 'Com_commit';
mysql> show global status like 'Com_rollback';
mysql> show global status like 'Uptime';
```

TPS = (Com_commit + Com_rollback) / Uptime

另一计算方式：基于Com_select、Com_insert、Com_delete、Com_update计算出QPS

```
mysql> show global status where Variable_name in('com_select','com_insert','com_delete','com_update');
```

等待1秒再执行，获取间隔差值，第二次每个变量值减去第一次对应的变量值，就是QPS

TPS计算方法：

```
mysql> show global status where Variable_name in('com_insert','com_delete','com_update');
```

计算TPS，就不算查询操作了，计算出插入、删除、更新四个值即可。

经网友对这两个计算方式的测试得出，当数据库中myisam表比较多时，使用Questions计算比较准确。当数据库中innodb表比较多时，则以Com_*计算比较准确。

5.2 开启慢查询日志

MySQL开启慢查询日志，分析出哪条SQL语句比较慢，使用set设置变量，重启服务失效，可以在my.cnf添加参数永久生效。

```
mysql> set global slow-query-log=on  #开启慢查询功能
mysql> set global slow_query_log_file='/var/log/mysql/mysql-slow.log'；#指定慢查询日志文件位置
mysql> set global log_queries_not_using_indexes=on;  #记录没有使用索引的查询
mysql> set global long_query_time=1;  #只记录处理时间1s以上的慢查询
```

分析慢查询日志，可以使用MySQL自带的mysqldumpslow工具，分析的日志较为简单。

mysqldumpslow -t 3 /var/log/mysql/mysql-slow.log #查看最慢的前三个查询

也可以使用percona公司的pt-query-digest工具，日志分析功能全面，可分析slow log、binlog、general log。

分析慢查询日志：pt-query-digest /var/log/mysql/mysql-slow.log

分析binlog日志：mysqlbinlog mysql-bin.000001 >mysql-bin.000001.sql

pt-query-digest --type=binlog mysql-bin.000001.sql

分析普通日志：pt-query-digest --type=genlog localhost.log

5.3 数据库备份

备份数据库是最基本的工作，也是最重要的，否则后果很严重，你懂得！但由于数据库比较大，上百G，往往备份都很耗费时间，所以就该选择一个效率高的备份策略，对于数据量大的数据库，一般都采用增量备份。常用的备份工具有mysqldump、mysqlhotcopy、xtrabackup等，mysqldump比较适用于小的数据库，因为是逻辑备份，所以备份和恢复耗时都比较长。mysqlhotcopy和xtrabackup是物理备份，备份和恢复速度快，不影响数据库服务情况下进行热拷贝，建议使用xtrabackup，支持增量备份。

Xtrabackup备份工具使用博文：http://lizhenliang.blog.51cto.com/7876557/1612800

5.4 数据库修复

有时候MySQL服务器突然断电、异常关闭，会导致表损坏，无法读取表数据。这时就可以用到MySQL自带的两个工具进行修复，myisamchk和mysqlcheck。

myisamchk：只能修复myisam表，需要停止数据库

常用参数：

-f --force　强制修复，覆盖老的临时文件，一般不使用

-r --recover 恢复模式

-q --quik　快速恢复

-a --analyze 分析表

-o --safe-recover 老的恢复模式，如果-r无法修复，可以使用此参数试试

-F --fast　只检查没有正常关闭的表

快速修复weibo数据库:

cd /var/lib/mysql/weibo

myisamchk -r -q *.MYI

mysqlcheck：myisam和innodb表都可以用，不需要停止数据库，如修复单个表，可在数据库后面添加表名，以空格分割

常用参数：

-a --all-databases 检查所有的库

-r --repair 修复表

-c --check 检查表，默认选项

-a --analyze 分析表

-o --optimize 优化表

-q --quik 最快检查或修复表

-F --fast 只检查没有正常关闭的表

快速修复weibo数据库:

mysqlcheck -r -q -uroot -p123 weibo

5.5 另外，查看CPU和I/O性能方法

#查看CPU性能

#参数-P是显示CPU数，ALL为所有，也可以只显示第几颗CPU

#查看I/O性能

#参数-m是以M单位显示，默认K

#%util：当达到100%时，说明I/O很忙。

#await：请求在队列中等待时间，直接影响read时间。

I/O极限：IOPS（r/s+w/s）,一般RAID0/10在1200左右。（IOPS，每秒进行读写（I/O）操作次数）

I/O带宽：在顺序读写模式下SAS硬盘理论值在300M/s左右，SSD硬盘理论值在600M/s左右。