全栈必备：MySQL性能调优

2016/11/01 · IT技术 · MySQL, 全栈, 数据库

对于全栈而言，数据库技能不可或缺，关系型数据库或者nosql，内存型数据库或者偏磁盘存储的数据库，对象存储的数据库或者图数据库……林林总总,但是第一必备技能还应该是MySQL。从LAMP的兴起，到Mariadb的出现，甚至PG的到来，熟练的MySQL技能都是大有用武之地的。

MySQL数据库技术的方方面面也是很多，这里只涉及必备的性能调优，推崇从下向上的性能调优，主要包括运行环境，配置参数，SQL性能，和系统架构设计调优。

运行环境调优

这里是Linux的天下，MySQL 运行环境的调优往往和Linux的内核调优一并完成。当然了，对云服务 RDS 也有一定的参考作用。

调整Linux默认的IO调度算法.

IO调度器的总体目标是希望让磁头能够总是往一个方向移动,移动到底了再往反方向走,这恰恰就是现实生活中的电梯模型,所以IO调度器也被叫做电梯 (elevator)，而相应的算法也就被叫做电梯算法.而Linux中IO调度的电梯算法有好几种,一个叫做as(Anticipatory),一个叫做 cfq(Complete Fairness Queueing),一个叫做deadline,还有一个叫做noop(No Operation).

IO对数据库的影响较大，linux默认的IO调度算法为cfq,需要修改为deadline,如果是SSD或者PCIe-SSD设备,需要修改为noop,可以使用下面两种修改方式。

```Shell
echo “deadline” > /sys/block/sda/queue/scheduler
```

2、修改/etc/grub.conf,永久生效。
修改/etc/grub.conf配置文件,在kernel那行增加一个配置,例如:

```Shell
elevator=deadline
```

主要关注elevator这个参数,设置内核的话需要重启系统才能生效。

禁用numa特性

新一代架构的NUMA不适合跑数据库,NUMA是为了内存利用率的提高,但反而可能导致一CPU的内存尚有剩余,另外一个却不够用了,发生swap的问题,因此一般建议关闭或修改NUMA的调度。

1、修改/etc/grub.conf关闭NUMA,重启后生效。

```Shell
numa=off
```

2、修改/etc/init.d/mysql或mysqld_safe脚本,设置启动mysqld进程时的NUMA调度机制,如 `numactl –interleave=all`。

修改swappiness设置

swappiness是linux的一个内核参数,用来控制物理内存交换出去的策略.它允许一个百分比的值,最小的为0,最大的为100,改值默认是60.这个设置值到底有什么影响呢?

vm.swappiness设置为0表示尽量少使用swap,100表示尽量将inactive的内存页交换到swap里或者释放cache。inactive内存的意思是程序映射着，但是”长时间”不用的内存。我们可以利用vmstat查看系统里面有多少inactive的内存。

```Shell
# vmstat -a 1
```

这个值推荐设置为1,设置方法如下,在/etc/sysctl.conf文件中增加一行。

```Shell
vm.swappiness = 1
```

扩大文件描述符

这个是经常修改的参数,高并发的程序都会修改.

1、动态修改,重启失效,只能使用root,并且当前session有效。

2、修改配置文件,永久生效。
在/etc/security/limits.conf配置文件中增加

```Shell
* hard nofile 51200

* soft nofile 51200
```

面向session的进程文件描述符的修改稍有不同，在云上的修改也略有差异，可以参见一样的“open too many files”

优化文件系统挂载参数。

对于文件系统，如无特殊要求，最好采用ext4.

文件系统挂载参数是在/etc/fstab文件中修改,重启时候生效。
noatime表示不记录访问时间,nodiratime不记录目录的访问时间。
barrier=0,表示关闭barrier功能.

barrier的主要目的是为了保证磁盘写数据的安全性,但是会降低性能。如果有BBU之类的电池备份电源保证控制卡不瞬间掉电,那么这个功能就可以放心大胆的关闭。

配置参数调优

my.cnf中的配置参数调优取决于业务，负载或硬件，在慢内存和快磁盘、高并发和写密集型负载情况下，都需要特殊的调整。

基本配置

query_cache_size
query cache是一个众所周知的瓶颈，甚至在并发并不多时也如此。 最 好是一开始就停用，设置query_cache_size = 0，并利用其他方法加速查询：优化索引、增加拷贝分散负载或者启用额外的缓存（比如memcache或redis） 。如果已经启用了query cache并且还没有发现任何问题，query cache可能有用。如果想停用它，那就得小心了。

innodb_buffer_pool_size

缓冲池是数据和索引缓存的地方：这个值越大越好，这能保证你在大多数的读取操作时使用的是内存而不是硬盘。典型的值是5–6GB(8GB内存)，20–25GB(32GB内存)，100–120GB(128GB内存)。

innodb_log_file_size

redo日志被用于确保写操作快速而可靠并且在崩溃时恢复。从MySQL 5.5之后，崩溃恢复的性能的到了很大提升，可以同时拥有较高的写入性能和崩溃恢复性能。在MySQL 5.6里可以被提高到4GB以上。如果应用程序需要频繁的写入数据，可以一开始就把它这是成4G。

max_connection值被设高了（例如1000或更高）之后一个主要缺陷是当服务器运行1000个或更高的活动事务时会变的没有响应。在应用程序里使用连接池或者在MySQL里使用进程池有助于解决这一问题。

back_log

要求 mysql 能有的连接数量。当主要mysql线程在一个很短时间内得到非常多的连接请求，这就起作用，然后主线程花些时间检查连接并且启动一个新线程。back_log指明在mysql暂时停止回答新请求之前的短时间内多少个请求可以被存在堆栈中。只有如果期望在一个短时间内有很多连接，需要增加它，换句话说，该值对到来的tcp/ip连接的侦听队列的大小。

Innodb配置

innodb_file_per_table

这项设置告知InnoDB是否需要将所有表的数据和索引存放在共享表空间里（innodb_file_per_table = OFF）或者为每张表的数据单独放在一个.ibd文件（innodb_file_per_table = ON）。每张表一个文件允许你在drop、truncate或者rebuild表时回收磁盘空间。这对于一些高级特性也是有必要的，比如数据压缩。但是它不会带来任何性能收益。MySQL 5.6中，这个属性默认值是ON。

innodb_flush_log_at_trx_commit

默认值为1，表示InnoDB完全支持ACID特性。当关注点是数据安全的时候这个值是最合适的，比如在一个主节点上。但是对于磁盘（读写）速度较慢的系统，它会带来很巨大的开销，因为每次将改变flush到redo日志都需要额外的fsyncs。如果值为0速度就更快了，但在系统崩溃时可能丢失一些数据, 所以一遍只适用于备份节点。

innodb_flush_method

这项配置决定了数据和日志写入硬盘的方式。一般来说，如果你有硬件RAID控制器，并且其独立缓存采用write-back机制，并有着电池断电保护，那么应该设置配置为O_DIRECT；否则，大多数情况下应将其设为fdatasync（默认值）。sysbench是一个可以帮助你决定这个选项的好工具。

innodb_log_buffer_size

这项配置决定了为尚未执行的事务分配的缓存。但是如果事务中包含有二进制大对象或者大文本字段的话，看Innodb_log_waits状态变量，如果它不是0，增加innodb_log_buffer_size。

其他配置

log_bin

如果数据库服务器充当主节点的备份节点，那么开启二进制日志是必须的。就算只有一个服务器，如果你想做基于时间点的数据恢复，这也是很有用的。二进制日志一旦创建就将永久保存。如果不想让磁盘空间耗尽，你可以用 PURGE BINARY LOGS 来清除旧文件，或者设置 expire_logs_days 来指定过多少天日志将被自动清除。记录二进制日志不是没有开销的，所以如果你在一个非主节点的复制节点上不需要它的话，那么建议关闭这个选项。

interactive_timeout

mysql_real_connect()使用 client_interactive 选项的客户。 默认数值是28800，建议改为7200。

table_open_cache

MySQL每打开一个表，都会读入一些数据到table_open_cache缓存中，当MySQL在这个缓存中找不到相应信息时，才会去磁盘上读取。假定系统有200个并发连接，则需将此参数设置为200*N(N为每个连接所需的文件描述符数目)；当把table_open_cache设置为很大时，如果系统处理不了那么多文件描述符，那么就会出现客户端失效，连接不上。

max_allowed_packet

接受的数据包大小；增加该变量的值十分安全，这是因为仅当需要时才会分配额外内存。例如，仅当你发出长查询或MySQLd必须返回大的结果行时MySQLd才会分配更多内存。该变量之所以取较小默认值是一种预防措施，以捕获客户端和服务器之间的错误信息包，并确保不会因偶然使用大的信息包而导致内存溢出
skip_name_resolve

当客户端连接数据库服务器时，且当DNS很慢时，建立连接也会很慢。因此建议在启动服务器时关闭skip_name_resolve选项而不进行DNS查找。

SQL 语句调优

在应用层，通过pt工具和慢查询日志的配合，可以轻松地分辨出全表扫描的语句。

基本原则

- 避免全表扫描
- 建立索引
- 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
- 尽量避免大事务操作，提高系统并发能力
- 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。尽量避免使用游标，因为游标的效率较差。

雕虫小技

关于where 后的条件

- 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。
- 应尽量避免在 where 子句中使用 or 来连接条件,可以考虑使用union 代替
- in 和 not in 也要慎用，对于连续的数值，能用 between 就不要用 in，exists 代替 in
- 尽量避免在 where 子句中对字段进行表达式操作和函数操作

关于数据类型

- 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。

说，在一个相对较小的字段内搜索效率显然要高些。

- 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。
- 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。

关于临时表

- 避免频繁创建和删除临时表，以减少系统表资源的消耗。对于一次性事件， 最好使用导出表。
- 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。
- 如果使用到了临时表，在最后将所有的临时表显式删除时，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

关于索引

- 先应考虑在 where 及 order by 涉及的列上建立索引。
- 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。
- 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和 update 的效率，因为 insert 或 update 时有可能会重建索引，所以视具体情况而定。一个表的索引数最好不要超过7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.

数据库架构调优

从底层来到了应用层，最终到架构层，然而脱离业务逻辑谈架构就是耍流氓。数据库架构同样是依赖业务系统的，稳定而又弹性地服务业务系统是关键。架构调优的方向有：

- 分区分表
- 业务分库
- 主从同步与读写分离
- 数据缓存
- 主从热备与HA双活
- …..