链滴

CentOS 下 MySQL 版本升级、配置调优

作者：houjie
原文链接：https://ld246.com/article/1516181433279
来源网站：链滴

最近需要对一台服务器上的MySQL进行升级.
为了使用MySQL5.5以后高性能的InnoDB引擎,将低版本的MySQL进行升级

CenOS 6.9(Final) 下升级 MySQL5.1 到 5.6

参考文献 : CentOS 6.5下升级mysql5.1，以及从5.1升级到5.7
参考文献 : [# MySQL数据备份之mysqldump使用
](https://www.cnblogs.com/feichexia/p/MysqlDataBackup.html)

1.安装mysql-server

yum install mysql-server

2.备份数据库

mysqldump -u xxx -h xxx -P 3306 -p --all-databases > databases.sql

mysqldump 详细使用方法

i. 最简单的用法:

mysqldump -uroot -pPassword [database name] > [dump file]

上述命令将指定数据库备份到某dump文件（转储文件）中，比如:

mysqldump -uroot -p123 test > test.dump

生成的test.dump文件中包含建表语句（生成数据库结构）和插入数据的insert语句。

ii. --opt

如果加上--opt参数则生成的dump文件中稍有不同:

- 建表语句包含drop table if exists tableName
- insert之前包含一个锁表语句lock tables tableName write，insert之后包含unlock tables

可见其并发安全性更好一些

iii. 跨主机备份

使用下面的命令可以将host1上的sourceDb复制到host2的targetDb，前提是host2主机上已经创建targetDb数据库:

mysqldump --host=host1 --opt sourceDb| mysql --host=host2 -C targetDb

-C指示主机间的数据传输使用数据压缩

iv. 只备份表结构

mysqldump --no-data --databases mydatabase1 mydatabase2 mydatabase3 > test.dump

将只备份表结构。--databases指示主机上要备份的数据库。如果要备份某个MySQL主机上的所有数库可以使用--all-databases选项，如下：

```
mysqldump --all-databases > test.dump
```

v. 从备份文件恢复数据库

```
mysql [database name] < [backup file name]
```

3.停止mysql服务

```
service mysqld stop
```

4.卸载旧版的mysql

```
yum remove mysql mysql-*
```

5.查看已安装的mysql扩展，并卸载

```
rpm -qa | grep mysql
yum remove mysql mysql-server mysql-libs compat-mysql51
yum list installed | grep mysql
yum remove mysql - libs
```

6.安装mysql5.7

```
rpm -Uvh http://repo.mysql.com/mysql-community-release-el6-5.noarch.rpm
```

6.1安装

```
yum install mysql-community-server
```

7.查看版本号

```
mysql -v
```

8.启动mysql服务

```
service mysqld start
```

9.初始化mysql5.7

```
mysqld --initialize
rm -rf /var/lib/mysql
service mysqld start
```

10.启动服务的时候根据提示修改密码

```
/usr/bin/mysqladmin -u root password '123456'
```

11.状态设置

```
chkconfig mysqld on
chkconfig --list
netstat -antp
service mysqld status
```

12. 恢复备份

其实如果正确安装的话 10.11.12步骤可以省略的

新版本的MySQL可以通过之前的配置文件继承之前的连接参数和数据

进入需要还原的数据库后

```
source databases.sql;
```

或者

```
mysqldump -u root -p databases < databases.sql
```

至此,MySQL 版本升级及数据备份及更新完毕

MySQL 配置调优

使用了一台32核 97G内存的服务器 调优是必不可少的步骤

参考资料:《MySQL管理之道:性能调优,高可用与监控》第五章 性能调优 (这本书写的相当不错,推荐)

参考资料:MySql5.6 my.cnf配置优化

参考资料:MySQL配置文件my.cnf参数优化和中文详解

整理了调优之后的my.cnf文件如下

部分参数的含义以及调优过程记录在文件中

最重要的是如下几个参数:

- **port** : 端口
- **character_set_server** : 编码
- **max_connections** : 最大连接数
- **innodb_buffer_pool_size** : innodb缓冲池大小
- **innodb_buffer_pool_instances** : innodb缓冲池实例个数

```
# my.cnf配置文件
# 生产环境下mysql配置优化 (72G内存)
# @author   :   h-j-13
# @time     :   2018-01-17


[mysqld]
# ------------主要配置-----------------
# 端口
```

```
port = 3306

# 数据地址
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

# 用户
user=mysql

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

# 字符编码
character_set_server = utf8

# 表名大小写敏感
lower_case_table_names = 0

open_files_limit = 10240
# inux下设置最大文件打开数
# open_files_limit最后取值为 配置文件 open_files_limit，max_connections*5， wanted_files= 1
+max_connections+table_cache_size*2 三者中的最大值。

binlog_cache_size = 4M
# 1~2M
# 如果有很大的事务,可以适当增加这个缓存值,以获取更好的性能

back_log = 600
# 默认值 80
# 在MYSQL暂时停止响应新请求之前，短时间内的多少个请求可以被存在堆栈中。如果系统在短时
内有很多连接，
# 则需要增大该参数的值，该参数值指定到来的TCP/IP连接的监听队列的大小。

max_connections = 1024
# 默认值 151
# MySQL允许最大的进程连接数，如果经常出现Too Many Connections的错误提示，则需要增大
值。

max_connect_errors = 4000
# 默认值 100
# 设置每个主机的连接请求异常中断的最大次数
# 当超过该次数，MYSQL服务器将禁止host的连接请求，直到mysql服务器重启或通过flush hosts
令清空此host的相关信息。

external-locking = FALSE
# 使用–skip-external-locking MySQL选项以避免外部锁定。该选项默认开启

max_allowed_packet = 32M
# 默认值 4MB
# 设置在网络传输中一次消息传输量的最大值。系统默认值 为4MB，最大值是1GB，必须设置1024
倍数。

sort_buffer_size = 2M
# Sort_Buffer_Size 是一个connection级参数，在每个connection（session）第一次需要使用这个b
```

ffer的时候，一次性分配设置的内存。
Sort_Buffer_Size 并不是越大越好，由于是connection级的参数，过大的设置+高并发可能会耗尽统内存资源。例如：500个连接将会消耗 500*sort_buffer_size(8M)=4G内存
Sort_Buffer_Size 超过2KB的时候，就会使用mmap() 而不是 malloc() 来进行内存分配，导致效率低。 系统默认2M，使用默认值即可

join_buffer_size = 2M
# 默认 128K
# 用于表间关联缓存的大小，和sort_buffer_size一样，该参数对应的分配内存也是每个连接独享。系默认2M，使用默认值即可

thread_cache_size = 300
# 默认 38
# 服务器线程缓存这个值表示可以重新利用保存在缓存中线程的数量,当断开连接时如果缓存中还有空,
# 那么客户端的线程将被放到缓存中,如果线程重新被请求，那么请求将从缓存中读取,如果缓存中是的或者是新的请求,
# 那么这个线程将被重新创建,如果有很多新的线程，增加这个值可以改善系统性能.通过比较 Connections 和 Threads_created 状态的变量，可以看到这个变量的作用。
# 设置规则如下：1GB 内存配置为8，2GB配置为16，3GB配置为32，4GB或更高内存，可配置更大。

thread_concurrency = 8
系统默认为10，使用10先观察
设置thread_concurrency的值的正确与否, 对mysql的性能影响很大, 在多个cpu(或多核)的情况下错误设置了thread_concurrency的值, 会导致mysql不能充分利用多cpu(或多核),
出现同一时刻只能一个cpu(或核)在工作的情况。thread_concurrency应设为CPU核数的2倍. 比如一个双核的CPU, 那么thread_concurrency的应该为4;
2个双核的cpu, thread_concurrency的值应为8
殊不知，thread_concurrency是在特定场合下才能使用的，参考mysql手册：
这个变量是针对Solaris系统的，如果设置这个变量的话，mysqld就会调用thr_setconcurrency()。个函数使应用程序给同一时间运行的线程系统提供期望的线程数目。
另外需要说明的是：这个参数到5.6版本就去掉了

query_cache_size = 64M
# 在MyISAM引擎优化中，这个参数也是一个重要的优化参数。但也爆露出来一些问题。机器的内存来越大，
# 习惯性把参数分配的值越来越大。这个参数加大后也引发了一系列问题。我们首先分析一下 query_ache_size的工作原理：
# 一个SELECT查询在DB中工作后，DB会把该语句缓存下来，当同样的一个SQL再次来到DB里调用，DB在该表没发生变化的情况下把结果从缓存中返回给Client。
# 这里有一个关建点，就是DB在利用Query_cache工作时，要求该语句涉及的表在这段时间内没有生变更。那如果该表在发生变更时，Query_cache里的数据又怎么处理呢?
# 首先要把Query_cache和该表相关的语句全部置为失效，然后在写入更新。那么如果Query_cache常大，该表的查询结构又比较多，查询语句失效也慢，
# 一个更新或是Insert就会很慢，这样看到的就是Update或是Insert怎么这么慢了。
# 所以在数据库写入量或是更新量也比较大的系统，该参数不适合分配过大。而且在高并发，写入量的系统，建议把该功能禁掉。

query_cache_limit = 4M
# 指定单个查询能够使用的缓冲区大小，缺省为1M

query_cache_min_res_unit = 4k
# 默认是4KB，设置值大对大数据查询有好处，但如果你的查询都是小数据查询，就容易造成内存碎和浪费

```
# 查询缓存碎片率 = Qcache_free_blocks / Qcache_total_blocks * 100%
# 如果查询缓存碎片率超过20%，可以用FLUSH QUERY CACHE整理缓存碎片，或者试试减小query_
ache_min_res_unit，如果你的查询都是小数据量的话。
# 查询缓存利用率 = (query_cache_size – Qcache_free_memory) / query_cache_size * 100%
# 查询缓存利用率在25%以下的话说明query_cache_size设置的过大，可适当减小;
# 查询缓存利用率在80%以上而且Qcache_lowmem_prunes > 50的话说明query_cache_size可能
点小，要不就是碎片太多。
# 查询缓存命中率 = (Qcache_hits – Qcache_inserts) / Qcache_hits * 100%

default-storage-engine = INNODB
# default_table_type = InnoDB
# 开启失败

thread_stack = 256K
# 默认 32bit - 192K
# 64bit - 256K
# 设置MYSQL每个线程的堆栈大小，默认值足够大，可满足普通操作。可设置范围为128K至4GB，
认为256KB，使用默认观察

transaction_isolation = READ-COMMITTED
# 设定默认的事务隔离级别.可用的级别如下:READ UNCOMMITTED-读未提交 READ COMMITTE-
已提交 REPEATABLE READ -可重复读 SERIALIZABLE -串行

tmp_table_size = 256M
# tmp_table_size 的默认大小是 32M。如果一张临时表超出该大小，MySQL产生一个 The table tbl
name is full 形式的错误，如果你做很多高级 GROUP BY 查询，增加 tmp_table_size 值。如果超过
值，则会将临时表写入磁盘。

max_heap_table_size = 256M

expire_logs_days = 7

key_buffer_size = 2048M
# 批定用于索引的缓冲区大小，增加它可以得到更好的索引处理性能，对于内存在4GB左右的服务器
说，该参数可设置为256MB或384MB。

read_buffer_size = 1M
# 默认128K
# 128K~256K
# MySql读入缓冲区大小。对表进行顺序扫描的请求将分配一个读入缓冲区，MySql会为它分配一段
存缓冲区。
# read_buffer_size变量控制这一缓冲区的大小。如果对表的顺序扫描请求非常频繁，并且你认为频
扫描进行得太慢，
# 可以通过增加该变量值以及内存缓冲区大小提高其性能。和sort_buffer_size一样，
# 该参数对应的分配内存也是每个连接独享。

read_rnd_buffer_size = 16M
# 128K~256K
# MySql的随机读（查询操作）缓冲区大小。当按任意顺序读取行时(例如，按照排序顺序)，将分配
个随机读缓存区。
# 进行排序查询时，MySql会首先扫描一遍该缓冲，以避免磁盘搜索，提高查询速度，如果需要排序
量数据，可适当调高该值。
# 但MySql会为每个客户连接发放该缓冲空间，所以应尽量适当设置该值，以避免内存开销过大。
```

```
bulk_insert_buffer_size = 64M
# 批量插入数据缓存大小，可以有效提高插入效率，默认为8M

myisam_sort_buffer_size = 128M
# MyISAM表发生变化时重新排序所需的缓冲 默认8M

myisam_max_sort_file_size = 10G
# MySQL重建索引时所允许的最大临时文件的大小 (当 REPAIR, ALTER TABLE 或者 LOAD DATA INF
LE).
# 如果文件大小比此值更大,索引会通过键值缓冲创建(更慢)

# myisam_max_extra_sort_file_size = 10G 5.6无此值设置
# myisam_repair_threads = 1   默认为1
# 如果一个表拥有超过一个索引, MyISAM 可以通过并行排序使用超过一个线程去修复他们.
# 这对于拥有多个CPU以及大量内存情况的用户,是一个很好的选择.

myisam_recover
# 自动检查和修复没有适当关闭的 MyISAM 表
skip-name-resolve

server-id = 1

# innodb_additional_mem_pool_size = 16M
# 这个参数用来设置 InnoDB 存储的数据目录信息和其它内部数据结构的内存池大小，类似于Oracle
library cache。这不是一个强制参数，可以被突破。
# 5.6不赞成使用,一个将被废除的设置
# InnoDB: Warning: Using innodb_additional_mem_pool_size is DEPRECATED. This option may
be removed in future releases

innodb_buffer_pool_size = 50G
# 这对Innodb表来说非常重要。Innodb相比MyISAM表对缓冲更为敏感。
# MyISAM可以在默认的 key_buffer_size 设置下运行的可以，然而Innodb在默认的 innodb_buffer
pool_size 设置下却跟蜗牛似的。
# 由于Innodb把数据和索引都缓存起来，无需留给操作系统太多的内存，因此如果只需要用Innodb
话则可以设置它高达 70-80% 的可用内存。
# 一些应用于 key_buffer 的规则有 — 如果你的数据量不大，并且不会暴增，那么无需把 innodb_buf
er_pool_size 设置的太大了

innodb_buffer_pool_instances = 8
# innodb_buffer_pool_instances可以开启多个内存缓冲池，把需要缓冲的数据hash到不同的缓冲
中，这样可以并行的内存读写。
# innodb_buffer_pool_instances 参数显著的影响测试结果，特别是非常高的 I/O 负载时。
# 实验环境下， innodb_buffer_pool_instances=8 在很小的 buffer_pool 大小时有很大的不同，而
用大的 buffer_pool 时，innodb_buffer_pool_instances=1 的表现最棒。

# innodb_data_file_path = ibdata1:2G:autoextend
# 设置过大导致报错，默认12M观察
# 表空间文件 重要数据

# innodb_file_io_threads = 4
# 默认值 4
# 文件IO的线程数，一般为 4，但是在 Windows 下，可以设置得较大。

# innodb_thread_concurrency = 0
```

```
# 服务器有几个CPU就设置为几，建议用默认设置，一般为8.

innodb_flush_log_at_trx_commit = 2
# 如果将此参数设置为1，将在每次提交事务后将日志写入磁盘。为提供性能，
# 可以设置为0或2，但要承担在发生故障时丢失数据的风险。设置为0表示事务日志写入日志文件，日志文件每秒刷新到磁盘一次。
# 设置为2表示事务日志将在提交时写入日志，但日志文件每次刷新到磁盘一次。

innodb_log_buffer_size = 64M
# 使用默认8M
# 此参数确定些日志文件所用的内存大小，以M为单位。
# 缓冲区更大能提高性能，但意外的故障将会丢失数据.MySQL开发人员建议设置为1 - 8M之间

innodb_log_file_size = 1024M
# 使用默认48M
# 此参数确定数据日志文件的大小，以M为单位，更大的设置可以提高性能，但也会增加恢复故障数库所需的时间

innodb_log_files_in_group = 3
# 使用默认2
# 为提高性能，MySQL可以以循环方式将日志文件写到多个文件。推荐设置为3M

innodb_max_dirty_pages_pct = 90
# 默认 75
# 推荐阅读 http://www.taobaodba.com/html/221_innodb_max_dirty_pages_pct_checkpoint.html
# Buffer_Pool中Dirty_Page所占的数量，直接影响InnoDB的关闭时间。参数innodb_max_dirty_pages_pct
# 可以直接控制了Dirty_Page在Buffer_Pool中所占的比率，而且幸运的是innodb_max_dirty_pages_pct是可以动态改变的。
# 所以，在关闭InnoDB之前先将innodb_max_dirty_pages_pct调小，强制数据块Flush一段时间，能够大大缩短 MySQL关闭的时间。

innodb_lock_wait_timeout = 120
# 默认为50秒
# InnoDB 有其内置的死锁检测机制，能导致未完成的事务回滚。但是，如果结合InnoDB使用MyISM的lock tables 语句或第三方事务引擎,则InnoDB无法识别死锁。
# 为消除这种可能性，可以将innodb_lock_wait_timeout设置为一个整数值，指示 MySQL在允许其事务修改那些最终受事务回滚的数据之前要等待多长时间(秒数)

innodb_file_per_table = 0
# 默认为No
# 独享表空间（关闭）



[mysqld_safe]
# -----------安全性配置-----------------
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
# sql_mode 详见 https://segmentfault.com/a/1190000005936172
# sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES

[client]
```

```
# ------------客户端配置----------------
socket = /var/lib/mysql/mysql.sock


[mysql]
# ------------启动配置------------------
#这个配置段设置启动MySQL服务的条件；在这种情况下，no-auto-rehash确保这个服务启动得比
快。
no-auto-rehash


[mysqldump]
# ------------备份设置------------------
quick
max_allowed_packet = 128M
```

将新的my.cnf复制到 /etc/my.cnf中后

使用命令重启MySQL

```
service mysqld restart
```

结果报错

```
mysql deamon failed to start
```

肯定配置文件的哪里的设置出现了问题

查看mysql日志文件

```
vim /var/log/mysqld.log
```

可见

```
...
    537 2018-01-17 15:49:10 13759 [Warning] InnoDB: Starting to delete and rewrite log files.
    538 2018-01-17 15:49:10 13759 [Note] InnoDB: Setting log file ./ib_logfile101 size to 1024
B
    539 InnoDB: Progress in MB: 100 200 300 400 500 600 700 800 900 1000
    540 2018-01-17 15:49:14 13759 [Note] InnoDB: Setting log file ./ib_logfile1 size to 1024 MB
    541 InnoDB: Progress in MB: 100 200 300 400 500 600 700 800 900 1000
    542 2018-01-17 15:49:20 13759 [Note] InnoDB: Setting log file ./ib_logfile2 size to 1024 MB
    543 InnoDB: Progress in MB: 100 200 300 400 500 600 700 800 900 1000
    544 2018-01-17 15:49:26 13759 [Note] InnoDB: Renaming log file ./ib_logfile101 to ./ib_logf
le0
    545 2018-01-17 15:49:26 13759 [Warning] InnoDB: New log files created, LSN=680535170
1
    546 2018-01-17 15:49:26 13759 [Note] InnoDB: 128 rollback segment(s) are active.
    547 2018-01-17 15:49:26 13759 [Note] InnoDB: Waiting for purge to start
    548 2018-01-17 15:49:26 13759 [Note] InnoDB: 5.6.39 started; log sequence number 68053
17091
    549 2018-01-17 15:49:26 13759 [ERROR] /usr/sbin/mysqld: unknown variable 'default-char
cter-set=utf8'
    550 2018-01-17 15:49:26 13759 [ERROR] Aborting
```

查看资料后得知 default-character-set 参数应该被替换为 character_set_server

更新my.ini 重启后成功开启数据库

结果意想不到的事情发生了,有些数据库连不进去
提示为不存在这个库,但并不是所有的库都无法访问,只有那些带着大写字母的库才能不能访问
故想到可能是MySQL大小写敏感问题,查阅资料后果然在自己配置文件中发现了

lower_case_table_names = 1

这一条,遂将其参数值改回0
更新后成功,使用正常

至此完成了MySQL 的配置调优

参考资料 : linux下mysql提示"mysql deamon failed to start"错误的解决方法
参考资料 : /usr/bin/mysqld: unknown variable 'default-character-set=utf8'的解决
参考资料 : MySQL之——提示"mysql deamon failed to start"错误的解决方法
参考资料 : [mysql大小写敏感配置
](http://blog.csdn.net/fdipzone/article/details/73692929)