Optimizing MySQL

By W.J. Gilmore

Table of Contents

Introduction.....1

Compiling and Configuring MySQL.....2

Tuning The Server.....4

Table Types.....6

Even More Optimization........8

Conclusion.....9

Introduction

In my two most recent articles (article I, article II), I discussed the virtues of database normalization and how steps taken towards ensuring that your tables are fully ‘normalized’ would ultimately result in a better organized, more easily maintained database. While database normalization certainly is a major step towards attaining the highest level of database efficiency, you may be interested to know that there is even more that can be done to attain the fastest database server possible.

The process I’m talking about is typically known as database optimization, and involves the analysis of several different facets of both your system hardware and your database system. In this article, I’ll discuss those facets that relate directly to the compilation, configuration and subsequent administration of the database server. More specifically, I’ll discuss optimization as it relates to the MySQL database server (http://www.mysql.com), although many of the concepts discussed herein could be applied to any database server on the market today.

Compiling and Configuring MySQL

While many novice developers think of source compilation as a necessary evil, it also plays a significant role in the performance of the resulting compiled program. Compare this process to building the same model car on two different assembly lines. The first line is riddled with lazy workers, faulty assemblage equipment, and implements a poor assembly strategy. The second line consists of knowledgeable workers who use the best tools and strategies to ensure the best possible product. While on the outside, both resulting cars may look the same, it is likely that there will be vast differences in performance. The same concept applies to compilers. Some just do a much better job than their counterparts.

Also, you should carefully consider all available compiler options. Chances are very good that you either will not need several of the default options, or should consider modifying some of those options to better suit your particular needs.

With these points in mind, what exactly can you do to help make your MySQL database blazing fast? I'll conclude this section with several tips:

By just using a better compiler and/or better compiler options you can get a 10−30% speed increase in your application." −− MySQL documentation

Use pgcc (Pentium GCC) is the GCC compiler
(http://www.gnu.org/software/gcc/gcc.html) optimized for those programs which will be used on systems using the Pentium processor. Using pgcc to compile the MySQL code will result in overall performance gains surpassing 10%! You can find more information about pgcc at http://www.goof.com/pcg/. Of course, if your server does not use the Pentium processor, don't bother; pgcc was written specifically for Pentium systems (thus the name).

Compile MySQL only with the character sets you intend on using
MySQL currently offers quite 24 different character sets, offering users worldwide with the possibility to insert and view the table data in their native language. By default, MySQL installs all of these charsets, however, chances are pretty good that you're only going to need one. You can disable all character sets except for the default 'Latin1' set by including the following option within the configuration line:

```
%>./configure −with−extra−charsets=none
[−−other−configuration−options]
```

Compile the mysqld executable statically
Compiling the mysqld executable without the shared libraries can also result in more efficient performance. You can compile mysqld statically by including the following option within the configuration line:

```
%>./configure –with–mysqld–ldflags=–all–static
[––other–configuration–options]
```

Sample Configuration
The following configuration command is commonly used to speed performance:

```
%>CFLAGS="–O6 –mpentiumpro –fomit–frame–pointer" CXX=gcc
CXXFLAGS="–O6
–mpentiumpro –fomit–frame–pointer –felide–constructors
–fno–exceptions –fno–rtti"
./configure ––prefix=/usr/local ––enable–assembler
––with–mysqld–ldflags=–all–static
––disable–shared
```

Explaining what exactly each of the gcc flags do is out of the scope of this article. However, all are explained in glorious detail within the gcc manual (http://gcc.gnu.org/), however I would not recommend curling up under the blankets with this one. Alternatively, you can review a list of all gcc options by executing man gcc at the command prompt.

Tuning The Server

While ensuring proper compilation is certainly important, it is only part of the battle. You may be interested to know that you can configure many of the MySQL variables that play an important role in the server's proper functioning. What's more, you can store these variable assignments within a configuration file, ensuring that they are in effect every time the MySQL server is started. This configuration file is known as the 'my.cnf' file.

The MySQL developers are so helpful that they have taken the time to provide you with several sample my.cnf files, found in the /usr/local/mysql/share/mysql/ directory. These files are titled my−small.cnf, my−medium.cnf, my−large.cnf and my−huge.cnf, the size specification found within each title describing the type of system that configuration file would be valid for. If you are running MySQL on a system with just relatively little RAM, and you only plan on using MySQL occasionally, then the my−small.cnf file would be ideal since it commands the mysqld daemon to use just a minimum amount of resources. Alternatively, if you were planning on building the next E−commerce superstore, and are using a system with 2G of RAM, then you probably want to use the mysql−huge.cnf file (among other things).

In order to use one of these files, you'll need to make a copy of the one that best fits your needs, renaming the copy as my.cnf. You have the option of using this copy to one of three scopes:

Global: Copying this my.cnf file to the server's /etc directory will make its variables global in scope, that is, valid for all MySQL database servers found on the server.
Local: Copying this file to [MYSQL−INSTALL−DIR]/var/ will render the my.cnf file local to that particular server. [MYSQL−INSTALL−DIR] denotes the directory in which MySQL is installed.
User: You can even limit the scope to a particular user, accomplished by copying the my.cnf file to the user's root directory.

So how do you set these variables within the my.cnf file? Furthermore, exactly which variables can you set? Although all variables are relative generally to the MySQL server, each has a more specific relationship to some component found within MySQL. For example, the variable max_connections falls under the mysqld category. How do I know this? I executed the following command:

```
%>/usr/local/mysql/libexec/mysqld −−help
```

This resulted in the display of the various options and variables relative to mysqld. You'll easily identify the variables, as all are found below the line:

```
Possible variables for option --set-variable (-O) are
```

You could then set any of these variables within the my.cnf file as follows:

```
set-variable = max_connections=100
```

This would result in the maximum number of simultaneous connections to the MySQL server being limited to 100. Be sure to insert the set-variable directive under the [mysqld] heading in the my.cnf file. If you don't understand what I'm talking about here, take a moment to review the configuration file.

Table Types

Many MySQL users might be surprised to know that MySQL actually offers five different table types, namely BDB, HEAP, ISAM, MERGE, and MyISAM. The BDB table type falls into a category all by itself, known as transaction−safe. The remaining tables fall into a second category, named non−transaction−safe. I'll discuss the details of each category and those tables that fall under each category in this section.

Transaction−safe

BDB
The Berkeley DB (BDB) tables are MySQL's transaction−capable tables, developed by Sleepycat Software (http://www.sleepycat.com). These tables provide functionality long−awaited by MySQL users, that is transaction−control. Transaction−control is an extremely valuable function in any RDBMS because it makes it possible to ensure that groups of commands are executed successfully, nullifying the results brought on by the commands if anything happens to go wrong during their execution. As you can imagine, transaction−control is very important in applications such as electronic banking.

Non−transaction−safe

HEAP
HEAP tables are the fastest MySQL table for accessing data. This is because they use a hashed index and are stored in dynamic memory. One very important point to keep in mind about HEAP tables is that if either MySQL or your server crashes, the data is lost!

ISAM
ISAM tables were the previous MySQL default until MyISAM was developed. I would recommend not using this table altogether, and work with the MyISAM table instead.

MERGE
MERGE tables are an interesting new kind of table, made available in Version 3.23.25. A MERGE table is actually a collection of identical MyISAM tables, merged together as one. The motive behind merging several identical tables is largely for efficiency reasons. Doing so can improve speed, searching efficiency, repair efficiency, and save disk space, depending on what you're attempting to do.

The MERGE tables are still considered beta, but should be officially declared stable very soon.

MyISAM
This is the default MySQL table type. It's based on the ISAM code, but with several useful extensions. Here are just a few reasons why MyISAM tables are good:

- MyISAM tables are smaller than ISAM tables, thereby using less resources.
- MyISAM tables are binary portable across various platforms
- Larger key sizes, larger key limits.

There are many advantages to MyISAM. Check out http://www.mysql.com/doc/I/S/ISAM.html for a complete

summary of this table.

Specifying Table Type

You specify a table's type at the time of creation. In the following example, I'll show you how to create a HEAP table:

```
mysql>CREATE TABLE email_addresses TYPE=HEAP (
->email char(55) NOT NULL,
->name char(30) NOT NULL,
->PRIMARY KEY(email) );
```

BDB tables require somewhat more configuration. Please refer to http://www.mysql.com/doc/B/D/BDB_overview.html for a complete summary of these tables and what must be done in order to take advantage of them.

Even More Table Types

To make your MySQL administration tasks even more interesting, the upcoming MySQL 4.0 will offer two new table types, named Innobase and Gemeni. There isn't too much information yet available about each, so stay tuned.

There is so much to be learned about the MySQL table types that this brief introduction certainly does not do them any justice. For a more complete introduction, I recommend taking some time to review the information found within the MySQL documentation (http://www.mysql.com).

Even More Optimization...

After all we've discussed so far, there is still more to be covered. The final concept that I'll discuss is the analysis and optimization of the MySQL tables using several of the database server's own built−in commands. Let's begin this section with an overview of those commands that help you to analyze what.

SHOW

You might be interested to know that you can view a summary of what has taken place on the MySQL server by simply executing:

```
mysql>show status;
```

This will result in a rather lengthy list of status variables and their respective values. Some of these variables include the number of aborted clients, number of aborted connections, number of connection attempts, maximum number of simultaneous connections, and a slew of other very useful information. This information is invaluable for determining system problems and inefficiencies.

SHOW is capable of doing much more than just displaying information regarding the MySQL server as a whole. It can also display valuable information about the log files, specific databases, tables, indexes, processes, and the privilege tables. Check out http://www.mysql.com/doc/S/H/SHOW.html for more information.

EXPLAIN

When prefacing a SELECT command, the EXPLAIN explains how that SELECT command is processed. This is not only useful for determining whether or not you should add indexes to a table, but also for determining exactly how a complex join is processed by MySQL.

OPTIMIZE

The OPTIMIZE function will allow you to recover space and defragment the data file, which is particularly important after a substantial number of updates or deletes have taken place on a table consisting of variable−length rows. OPTIMIZE currently only works with MyISAM and BDB tables.

There are several other very useful functions geared towards helping you coax your tables into tip−top shape. However, those discussed thus far should keep you busy for the moment. ;−)

Conclusion

As you've likely surmised over this lengthy article, there is quite a bit that you can do to increase the efficiency of MySQL, starting with the compilation of the database server, and continuing throughout the entire administration process. Of course, we didn't even get into maximizing table−join efficiency, intelligent use of basic SQL queries, hardware considerations. However, I'll be happy to discuss these matters in a later article, pending your interest. See you next time!