**Brief introduction of SocketPro continuous SQL-stream sending and processing system (Part 2: MySQL)**

**Introduction**

Most of client server database systems only support synchronous communication between client and backend database by use of blocking socket and some chatty protocols that require a client or server to wait for an acknowledgement before sending a new chunk of data. The wait time, which is also called as latency, could be starting from a few tenths for a local area network (LAN) to hundreds of milliseconds for a wide area network (WAN). Large wait times can significantly degrade the quality of an application.

Fortunately, UDAParts has developed a powerful and secure communication framework named as SocketPro, which is written with continuous inline request/result batching and real-time stream processing capabilities by use of asynchronous data transferring and parallel computation for the best network efficiency, development simplicity, performance, scalability, and many great and even unique features at the site (<https://github.com/udaparts/socketpro>).

Further, UDAParts has applied the powerful SocketPro framework onto popular opened source databases such as Sqlite and MySQL as well as others through ODBC drivers to support continuous SQL-stream sending and processing. At the last, these components for databases are **totally free forever** to the public. For reduction of learning complexity, I recommend you study the SQL-stream sample for Sqlite (<sqlstream_sqlite.pdf>) first before playing these MySQL sample projects as Sqlite and MySQL samples share the same client API functions.

MySQL is currently the most popular open-source client-server distributed database management system. After studying MySQL source code, UDAParts has applied SocketPro SQL-stream technology onto MySQL and developed a server plug-in. UDAParts has compared SQL-stream technology with MySQL Connector/Net. Our performance study shows that SQL-stream technology can be up to one hundred times faster than MySQL Connector/Net.

**Source codes and samples**

All related source codes and samples are located at <https://github.com/udaparts/socketpro>. After cloning it into your computer by GIT, pay attention to the subdirectory *mysql* inside the directory *socketpro/stream\_sql*. You can see these samples are created from .NET, C/C++, Java and Python development environments. However, we use C# code (*socketpro/stream\_sql/mysql/test\_csahrp*) for client and C++ code (*socketpro/stream\_sql/mysql/smysql*) for server side development at this article for explanations.

You should distribute system libraries inside the directory of *socketpro/bin* into your system directory before running these sample applications.

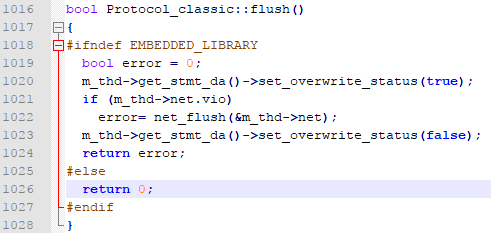
In regards to SocketPro communication framework, you may also refer to its development guide documentation at *socketpro/doc/****SocketPro development guide.pdf*.**

**Enabling MySQL prepare statements within server plug-in**

**Although MySql server plug-in does support general SQL statements, it does not support prepare statements at all. After digging into MySQL source code, UDAParts has figured out how to enable MySQL prepare statements within server plug-in. To add support of prepare statements within MySQL server plug-in, it is required to modify the four implementation files, *protocol\_callback.cc*, *protocol\_callback.h*, *protocol\_classic.cc* and *sql\_prepare.cc* before compiling the MySQL application *mysqld*. It is noted that this is a temporary solution. Once MySQL server plug-in supports prepare statements in the future, UDAParts will use MySQL native implementation instead as expected.**

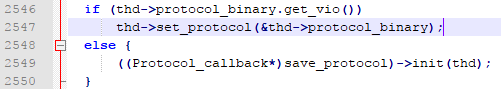
***protocol\_callback.cc and protocol\_callback.h:* At the very beginning, add a forward class declaration (*class Item\_param;*) at the header of the file *protocol\_callback.h*. Next, it is required that the class *Protocol\_callback* has to be added with two public methods (*send\_out\_parameters* and *init*) and one protected member *m\_thd*. These members have already been implemented within the two files at the directory *socketpro/stream\_sql/mysql/mysql-5.7.18*.**

***protocol\_classic.cc:* The method *Protocol\_classic::flush* has to be modified as shown in the below Figure 1 because the member *vio* could be null for MySQL server side plug-in.**



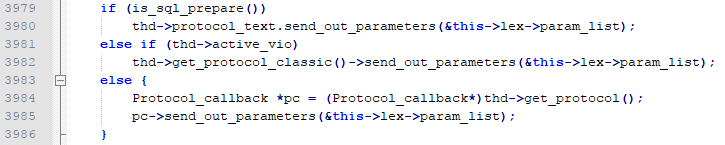
***Figure 1: The implementation of the method Protocol\_classic::flush for enabling MySQL prepare statements within server plug-in***

***sql\_prepare.cc:* At the very beginning, add one include for referring the file *protocol\_callback.h*. Afterwards, find the method *mysqld\_stmt\_execute*, and use the below code as shown at the below Figure 2 to replace this call *thd->set\_protocol(&thd->protocol\_binary);*.**



***Figure 2: Modification of the method mysqld\_stmt\_execute within the file sql\_prepare.cc***

**Next, find the method *Prepared\_statement::execute*, navigate to its end, and find this if statement (*if (error == 0 && this->lex->sql\_command == SQLCOM\_CALL)*). Use the code snippet as shown in the below Figure 3 to replace all its internal braced code.**



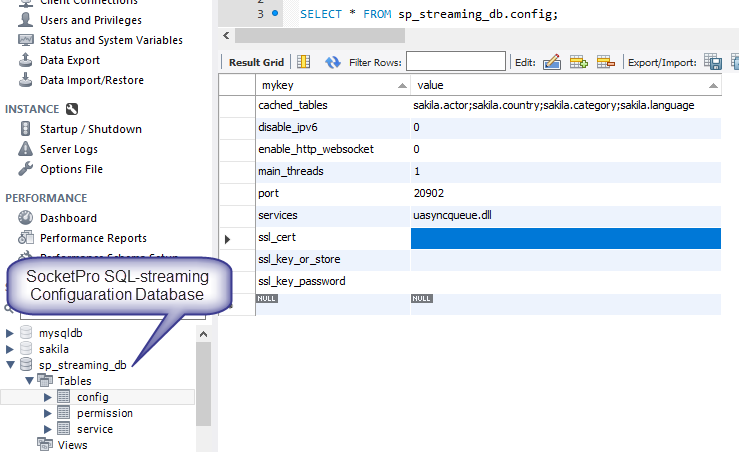
***Figure 3: Modification of the method Prepared\_statement::execute within the file sql\_prepare.cc***

**You can see the above code to check if active *vio* is available at run time. If it is not available for server plug-in prepared statements, we use callback protocol instead.**

**Before starting the newly compiled *mysqld* application on your system, you should explicitly set *plugin\_dir* to the directory containing MySQL plugin libraries (for example, *plugin\_dir =/usr/lib/mysql/plugin*) within the section *mysqld* of MySQL configuration file. Also, it is better to increase *thread\_stack* to 512K (*thread\_stack = 512K*) by changing the MySQL configuration file. Next, copy MySQL SQL-stream plugin *libsmysql.so* (smysql.dll on window platforms) into the MySQL plugin directory.**

**Register SocketPro MySQL SQL-streaming plugin and its configuration database**

**As described at** [this site](https://dev.mysql.com/doc/refman/5.7/en/install-plugin.html)**, register MySQL SQL-stream plugin by calling statement *INSTALL PLUGIN UDAParts\_SQL\_Streaming SONAME ‘libsmysql.so’* from the application *mysql*. If successful, you should see a new database *sp\_streaming\_db* created as shown in the below Figure 4.**



***Figure 4: SocketPro SQL-streaming configuration database sp\_streaming\_db and table config***

The configuration database has three simple tables, config, service and permission as shown in the above Figure 4. It is expected that **SocketPro MySQL SQL-streaming plugin supports industrial security standard SSL3/TLSv1.x to secure communication between client and server. By default, a SocketPro client can use either IP v4 or v6 to access MySQL database at port number 20902. Pay attention to the record *cached\_tables*. If you set its value properly, all connected SocketPro clients can see data changes within these tables (for example. table *actor*, *country*, *category* and *language* within database *sakila*) in real time. Referring the sample *test\_cache* at directory *socketpro/stream\_sql/mysql*, you can use the real-time cache feature to improve your middle tier performance and scalability by reducing data trips between middle tier and database.**

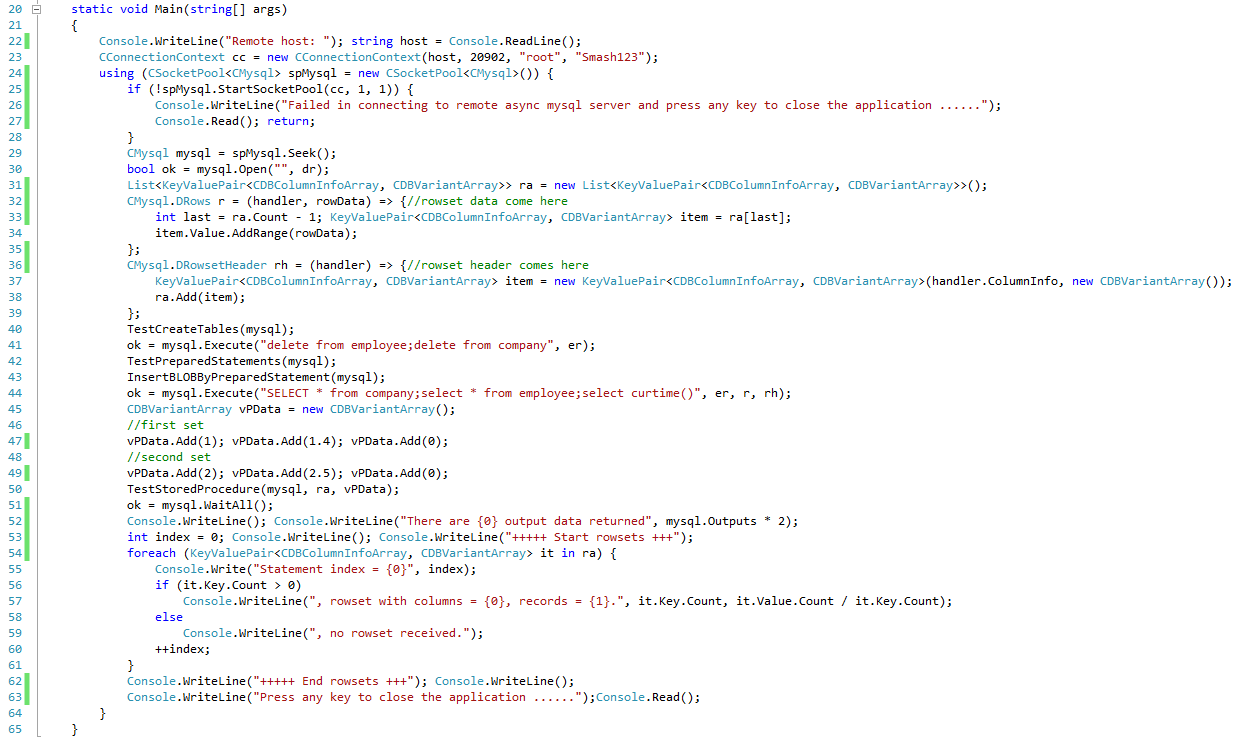
**One SocketPro server is capable to support many services at the same time by use of one TCP port. If you like, you can enable websocket from SocketPro MySQL SQL-streaming plugin by setting value to ‘1’ for record *enable\_http\_websocket*. Further, you can also embed other services by setting value properly of record *services* as shown in the above Figure 4. Once changing any one or more values within the table *config*, you should restart MySQL. Otherwise, the changes will not function correctly.**

**Under most cases, it is not expected that you will update the table *service*.**

**SocketPro MySQL SQL-streaming plugin uses the two tables *mysql.user* and *sp\_streaming\_db.permission* to authenticate all clients for all services. However, its SQL-streaming service only uses the table *mysql.user* for authentication.**

**Main function**

SocketPro is written from bottom to support parallel computation by use of one or more pools of non-blocking sockets. Each of pools may is made of one or more threads, and each of threads hosts one or more non-blocking sockets at client side. However, we just use one pool for clear demonstration here, and the pool is made of one thread and one socket for this sample at client side as shown in the below Figure 4.



*Figure 4: Main function for demonstration of SocketPro MySQL SQL-stream system at client side*

***Starting one socket pool:*** The above Figure 1 starts one socket pool which only has one worker thread that only hosts one non-blocking socket at line 25 for demonstration clarity by use of one instance of connection context. However, you can create multiple pools within one client application if necessary. Afterwards, we get one asynchronous MySQL handler at line 29.

***Opening database:*** We are able to send a request for opening a MySQL server database at line 30. If the first input is an empty or null string as shown at this example, we are opening one default database for a connected user, for example. If you like to open a specified database, you can simply give a non-empty valid database name string. In addition, you need to set a callback or Lambda expression for tracking returning error message from server side if you like as shown at line 30. It is noted that SocketPro supports only asynchronous data transferring between client and server so that a request could be inputted with one or more callbacks for processing returning data. This is completely different from synchronous data transferring. In addition, we create an instance of container that is used to receive all sets of records in coming queries at line 31.

***Streaming SQL statements:*** Keep in mind that SocketPro supports streaming all types of any number of requests on one non-blocking socket session effortlessly by design. Certainly, we can easily stream all SQL statements as well as others as shown at lines 40 through 50. All SocketPro SQL-stream services support this unique feature for the best network efficiency, which will significantly improve data accessing performance. As far as we know, you cannot find such a wonderful feature from other technologies. If you find one, please let us know. Like normal database accessing APIs, SocketPro SQL-stream technology supports manual transaction too as shown in the [previous article](sqlstream_sqlite.pdf).

***Waiting until all processed:*** Since SocketPro only supports asynchronous data transferring, SocketPro must have a way to wait until all requests and returning results are sent, processed and returned. SocketPro does come with this method WaitAll at client side to serve this purpose as shown at line 51. If you like, you can use this method to convert all asynchronous requests into synchronous ones.

**TestCreateTables, TestPreparedStatements and InsertBLOBByPreparedStatement**

As shown at lines 40, 42 and 43 in the above Figure 4, we could ignore them because the they are truly the same as ones in the [previous article](sqlstream_sqlite.pdf). Let’s focus executing MySQL stored procedures with input-output and output parameters.

**TestStoredProcedure**

MySQL fully supports stored procedures. SocketPro SQL-stream technology does too. Further, SocketPro SQL-stream technology supports executing multiple sets of MySQL stored procedures with input-output and output parameters in one call as shown at lines 45 through 50 in the above Figure 4.

**Performance study**

SocketPro SQL-stream technology has excellent performance in database data accessing for both query and update. You can see two performance test projects (cppperf and netperf) available at socketpro/samples/module\_sample/usqlite/DBPerf/. The first sample is written by C++ and the other by C#. In addition, MySQL sakila sample database, which is located at the directory socketpro/samples/module\_sample/usqlite/DBPerf, is used for you to play after running the sample test\_csqlite for creating a global sqlite database usqlite.db.

Our performance study shows that it is easy to get query executed at the speed of 12,000 times per second and socket connection. For insert, you can easily get the speed like 50,000 inserts per second for Sqlite.