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 protocol that requires 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.

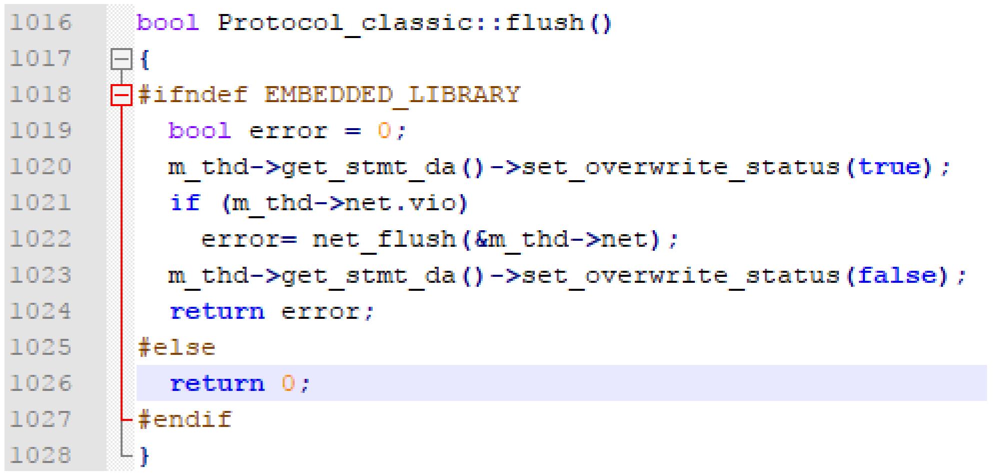
In regards to SocketPro communication framework, you may also refer to its development guide documentation at *socketpro/doc/SocketPro development guide.pdf*.

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 must be 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. UDAParts has pre-compiled MySQL server application mysqld located at the directory socketpro/stream_sql/mysql/(mysql-5.7.18|mysql-5.7.19) in case you are not familiar with C/C++ and want to skip the following paragraphs directly to the paragraph beginning with ***Restart mysqld:***.

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* must 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*.

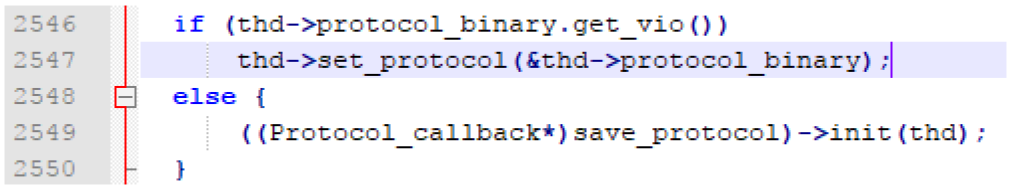
protocol_classic.cc: The method *Protocol_classic::flush* has to be modified as shown in the below Figure 1 because the member *vio* could be null for MySQL server side plug-in.



```
bool Protocol_classic::flush()
{
#ifndef EMBEDDED_LIBRARY
  bool error = 0;
  m_thd->get_stmt_da()->set_overwrite_status(true);
  if (m_thd->net.vio)
    error= net_flush(&m_thd->net);
  m_thd->get_stmt_da()->set_overwrite_status(false);
  return error;
#else
  return 0;
#endif
}
```

Figure 1: The implementation of the method Protocol_classic::flush for enabling MySQL prepare statements within server plug-in

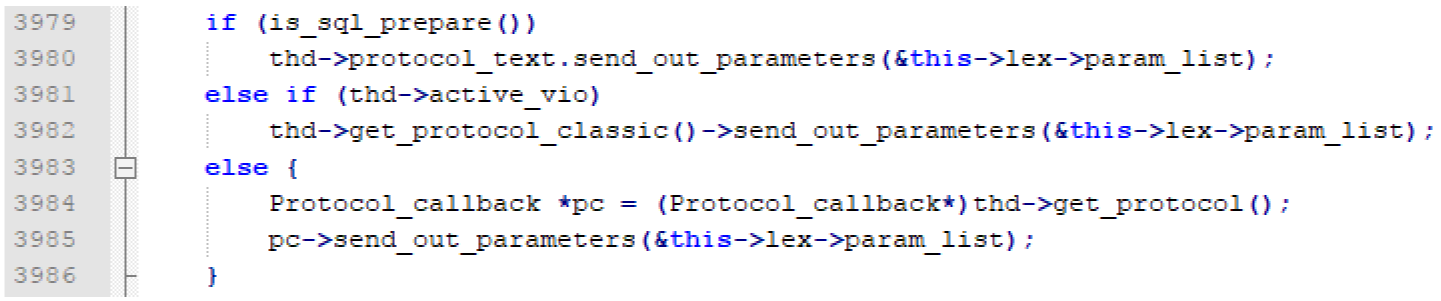
sql_prepare.cc: At the very beginning, add one include for referring the file *protocol_callback.h*. Afterwards, find the method *mysqld_stmt_execute*, and use the below code as shown at the below Figure 2 to replace this call *thd->set_protocol(&thd->protocol_binary);*.



```
if (thd->protocol_binary.get_vio())
    thd->set_protocol(&thd->protocol_binary);
else {
    ((Protocol_callback*)save_protocol)->init(thd);
}
```

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.



```
if (is_sql_prepare())
    thd->protocol_text.send_out_parameters(&this->lex->param_list);
else if (thd->active_vio)
    thd->get_protocol_classic()->send_out_parameters(&this->lex->param_list);
else {
    Protocol_callback *pc = (Protocol_callback*)thd->get_protocol();
    pc->send_out_parameters(&this->lex->param_list);
}
```

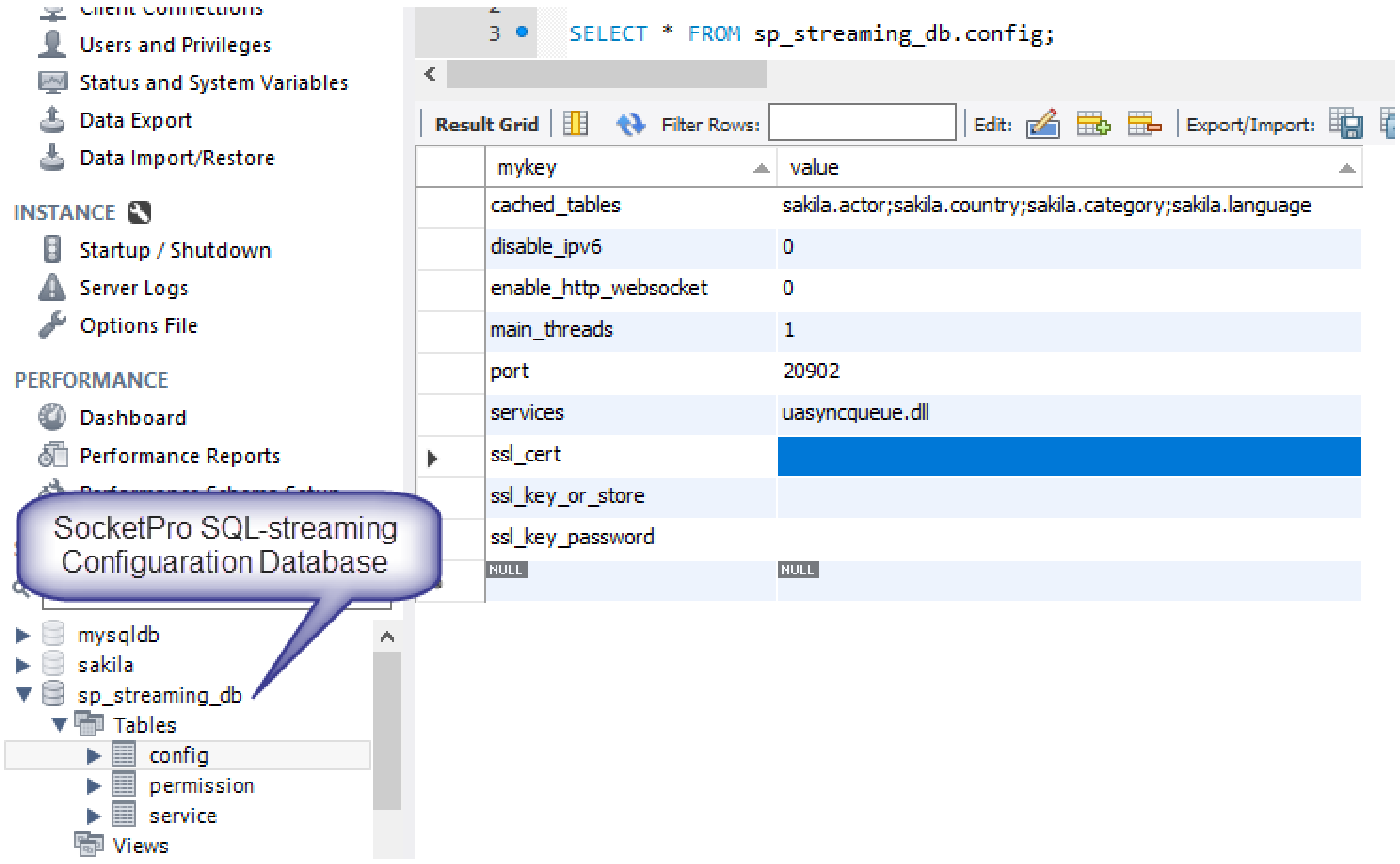
Figure 3: Modification of the method Prepared_statement::execute within the file sql_prepare.cc

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.

Restart mysqld: 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. After resetting all these configuration settings and replacing mysqld with new one, restart MySQL service or daemon.

Register SocketPro MySQL SQL-streaming plugin and its configuration database

As described at this site, 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.



```
SELECT * FROM sp_streaming_db.config;
```

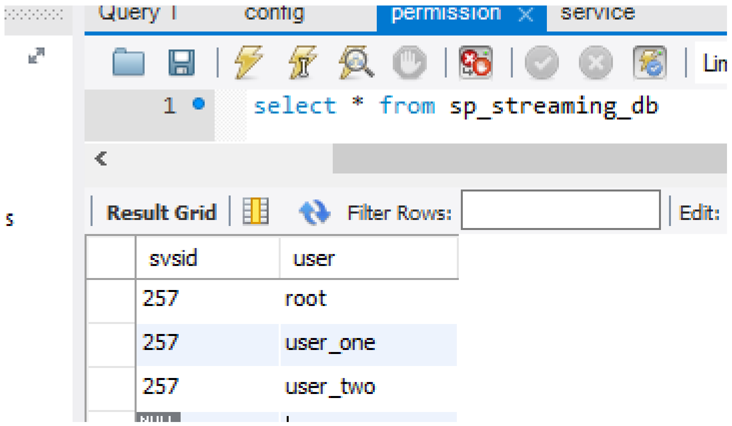
mykey	value
cached_tables	sakila.actor;sakila.country;sakila.category;sakila.language
disable_ipv6	0
enable_http_websocket	0
main_threads	1
port	20902
services	uasyncqueue.dll
ssl_cert	
ssl_key_or_store	
ssl_key_password	
NULL	NULL

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.

In regards to the table *permission*, SocketPro MySQL SQL-streaming technology uses its records to authenticate clients for embedded services as shown in the following Figure 5. 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 does not use records within the table *sp_streaming_db.permission* for authentication.



```
select * from sp_streaming_db
```

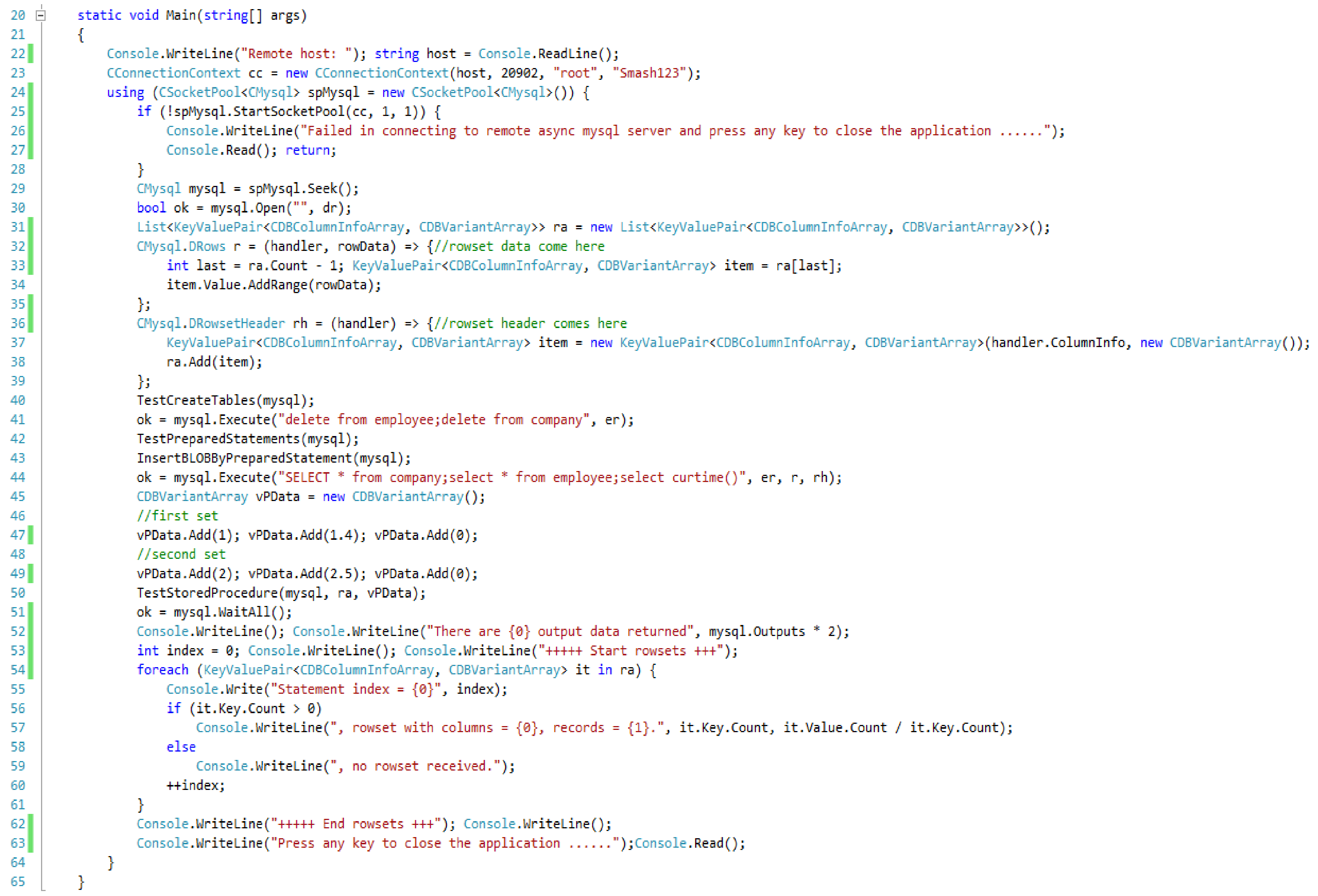
svsid	user
257	root
257	user_one
257	user_two

Figure 5: Three users (root, user_one and user_two) allowed for SocketPro asynchronous persistent message queue service (service id=257)

Under most cases, you are not required to touch the table *service*.

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 be 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 with this sample at client side as shown in the below Figure 6.



```
static void Main(string[] args)
{
    Console.WriteLine("Remote host: "); string host = Console.ReadLine();
    CConnectionContext cc = new CConnectionContext(host, 20902, "root", "Smash123");
    using (CSocketPool<CMysql> spMysql = new CSocketPool<CMysql>()) {
        if (!spMysql.StartSocketPool(cc, 1, 1)) {
            Console.WriteLine("Failed in connecting to remote async mysql server and press any key to close the application ......");
            Console.Read(); return;
        }
        CMysql mysql = spMysql.Seek();
        bool ok = mysql.Open("", dr);
        List<KeyValuePair<CDBColumnInfoArray, CDBVariantArray>> ra = new List<KeyValuePair<CDBColumnInfoArray, CDBVariantArray>>();
        CMysql.DRows r = (handler, rowData) => {//rowset data come here
            int last = ra.Count - 1; KeyValuePair<CDBColumnInfoArray, CDBVariantArray> item = ra[last];
            item.Value.AddRange(rowData);
        };
        CMysql.DRowsetHeader rh = (handler) => {//rowset header comes here
            KeyValuePair<CDBColumnInfoArray, CDBVariantArray> item = new KeyValuePair<CDBColumnInfoArray, CDBVariantArray>(handler.ColumnInfo, new CDBVariantArray());
            ra.Add(item);
        };
        TestCreateTables(mysql);
        ok = mysql.Execute("delete from employee;delete from company", er);
        TestPreparedStatements(mysql);
        InsertBLOBByPreparedStatement(mysql);
        ok = mysql.Execute("SELECT * from company;select * from employee;select curtime()", er, r, rh);
        CDBVariantArray vPData = new CDBVariantArray();
        //first set
        vPData.Add(1); vPData.Add(1.4); vPData.Add(0);
        //second set
        vPData.Add(2); vPData.Add(2.5); vPData.Add(0);
        TestStoredProcedure(mysql, ra, vPData);
        ok = mysql.WaitAll();
        Console.WriteLine(); Console.WriteLine("There are {0} output data returned", mysql.Outputs * 2);
        int index = 0; Console.WriteLine(); Console.WriteLine("+++++ Start rowsets +++");
        foreach (KeyValuePair<CDBColumnInfoArray, CDBVariantArray> it in ra) {
            Console.Write("Statement index = {0}", index);
            if (it.Key.Count > 0)
                Console.WriteLine(", rowset with columns = {0}, records = {1}.", it.Key.Count, it.Value.Count / it.Key.Count);
            else
                Console.WriteLine(", no rowset received.");
            ++index;
        }
        Console.WriteLine("+++++ End rowsets +++"); Console.WriteLine();
        Console.WriteLine("Press any key to close the application ......");Console.Read();
    }
}
```

Figure 6: Main function for demonstration of SocketPro MySQL SQL-stream system at client side

Starting one socket pool: The above Figure 6 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, it is noted that you can create multiple pools within one client application if necessary. Afterwards, we get one asynchronous MySQL handler at line 29.

Opening database: We can 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.

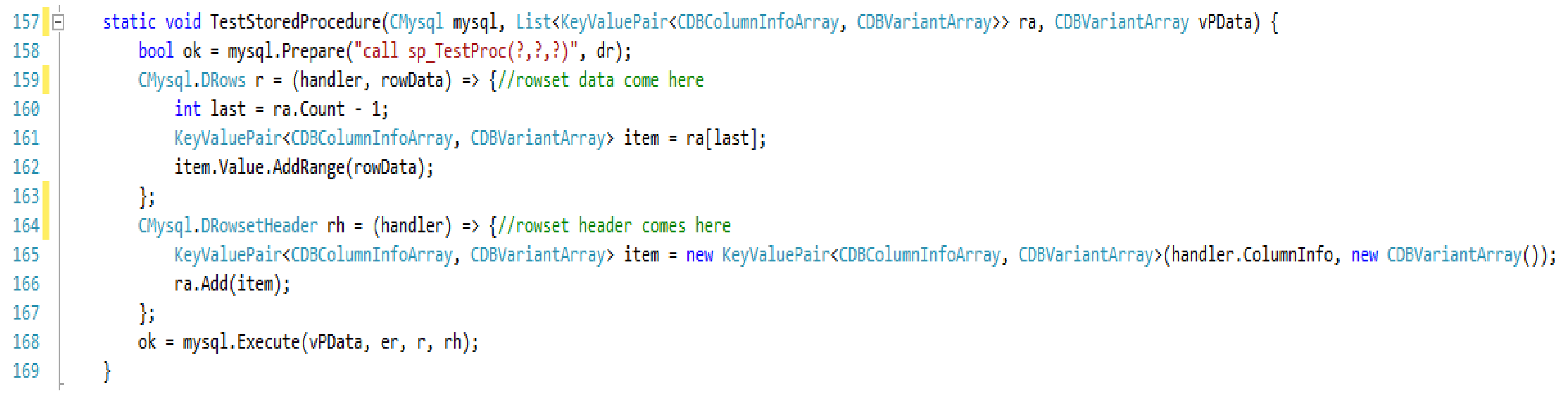
Waiting until all processed: Since SocketPro only supports asynchronous data transferring, SocketPro must have a way to wait until all requests and returning results are sent, processed and returned. SocketPro does come with this method WaitAll at client side to serve this purpose as shown at line 51. 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 6, we could ignore them because they are truly the same as ones in the previous article. 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 6. The below Figure 7 shows how to call a MySQL stored procedure which may have input, input/output and output parameters and return multiple sets of records.



```
157  static void TestStoredProcedure(CMysql mysql, List<KeyValuePair<CDBColumnInfoArray, CDBVariantArray>> ra, CDBVariantArray vPData) {
158      bool ok = mysql.Prepare("call sp_TestProc(?,?,?)", dr);
159      CMysql.DRows r = (handler, rowData) => {//rowset data come here
160          int last = ra.Count - 1;
161          KeyValuePair<CDBColumnInfoArray, CDBVariantArray> item = ra[last];
162          item.Value.AddRange(rowData);
163      };
164      CMysql.DRowsetHeader rh = (handler) => {//rowset header comes here
165          KeyValuePair<CDBColumnInfoArray, CDBVariantArray> item = new KeyValuePair<CDBColumnInfoArray, CDBVariantArray>(handler.ColumnInfo, new CDBVariantArray());
166          ra.Add(item);
167      };
168      ok = mysql.Execute(vPData, er, r, rh);
169  }
```

Figure 7: Call MySQL stored procedure which returns multiple sets of records and output parameters

It is very simple to call stored procedure through SocketPro SQL-stream technology at line 168 as shown in the above Figure 7. It is noted that all output parameter data will be directly copied into the passing parameter data array *vPData*. The callback *rh* at lines 164 through 167 is called when record set meta data comes if available. Whenever an array of record data comes, the callback *r* at lines 159 through 163 will be called. You can populate all queried meta and record data into an arbitrary container like *ra*, for example, from the two callbacks.

Performance study

SocketPro SQL-stream technology has excellent performance in database data accessing for both query and update. You can see two MySQL performance test projects (cppperf and netperf) available at socketpro/stream_sql/mysql/DBPerf/. The first sample is written by C++ and the other by C#. A sample project mysqlperf writtern from C# is provided for you to compare SocketPro SQL-stream technology with MySQL .NET provider in performance.

Our performance studies show that it is easy to get query executed at the speed of 15,000 times per second and socket connection. For insert, the speed could be easily more than 60,000 records per second. SocketPro SQL-stream technology is significantly faster than MySQL .NET provider in performance. The improvement of SocketPro SQL-stream technology over MySQL .NET provider is mainly dependent on network type, LAN or WAN. The improvement could be up to 200 times for WAN for small SQL requests having small sizes of returning data. The improvement could be between 0.5 and 5 times for LAN or localhost.

Points of interest

At last, SocketPro MySQL SQL-stream plugin doesn't support cursors at all, but it does provide all required basic client/server database features. Further, the SQL-stream plugin does have the following unique features.

1. Continuous inline request/result batching and real-time SQL-stream processing for the best network efficiency
2. Bi-directional asynchronous data transferring between client and server, but all asynchronous requests can be converted into synchronous ones
3. Superior performance and scalability because of powerful SocketPro communication architecture
4. Real-time cache for table update, insert and delete as shown at the sample project *test_cache* at the directory *socketpro/stream_sql/mysql/test_cache*
5. All requests are cancelable by executing the method *Cancel* of class *CClientSocket* at client side
6. Both windows and Linux are supported
7. Simple development for all supported development languages
8. Both client and server components are thread-safe. They can be easily reused within your multi-threaded applications with much fewer thread related issues