Test Automation for Multi-Platform Client/Server Software

Heesun Park, Ph.D
SAS Institute, Inc.
Cary, North Carolina
Sashsp@sas.com

Abstract

This paper is based on our testing experience of two major releases (version 6 and version 7) of the multi-platform client/server software, SAS/SHARE product which is the database management system for SAS users. It describes our test automation strategy for version 6 and identifies the achievements and weaknesses of our approach and explains how we improve our test process. The test process improvement effort for version 7 includes reduction of test automation level, use of macros that facilitate reproduction of defects in an interactive manner and extensive use of mega tests for early detection of defects as well as cross host and cross version tests.

Background

The SAS system started out as statistical analysis software on MVS in mid 1970's. With major rewrite for portability in mid 1980's, it became an information delivery and application development software which runs on all major platforms including mainframes, UNIX machines and PCs. Currently, the SAS system is the leader in data warehousing, data mining and decision support server arenas. There are many factors that contribute to the success and the expansion of the system. They include the portability of the application on multiple platforms, seamless client/server support on all platforms and capability of extracting data from many relational database management systems.

Application portability gives users freedom to choose hardware without losing their investment in application development. Also, the portable tests, which are SAS applications, allow testers to cover huge number of client and server combinations in a reasonable time frame.

As for the client and server support, we provide file transfer, remote computing, and remote library service capability with the SAS/CONNECT product and multi-user database management support with SAS/SHARE.

Another key ingredient to the success of the SAS system lies in its basic data storage format. From the beginning, SAS system used a "table" format for its data storage, which later became the de facto standard for all contemporary relational database systems.

In this paper, we will concentrate on test automation strategy for SAS/SHARE product, which is supported on all client/server platform combinations.

Version 6 Test Strategy

SAS Version 6 represents the first portable system targeted for multiple platforms. The last release of version 6 system was supported on 11 platforms including MVS, CMS, VAX, SUN, HP, IBM RS6000, OS2, WIN95 and WINNT. Client and server combinations for SAS/SHARE product were supported on all platform combinations and in many cases with multiple access methods on one platform. For example, on MVS, we support four different access methods: Cross Memory Services (XMS) for clients on the same CPU, Advanced Program to Program Communication (APPC) protocol for clients on IBM SNA networks, VTAM Logical Unit 0 (LU0) protocol for MVS mainframe network clients, and the TCP/IP access method for clients on UNIX and PCs. Roughly, we have more than 200 client and server combinations to cover and it is impractical to cover each and every combination with full test suites.

Our test strategy focuses on prioritization of test combinations, selection of test cases to run on each combination and full test automation, to provide maximum coverage with minimum risk in a reasonable time frame.

As for the prioritization of combinations, we consider inputs from marketing, technical support and development groups. In most cases, it is not difficult to identify major and important combinations since we clearly experience the proliferation of UNIX boxes and PCs.

Selection of test cases to run on each combination becomes very important since we may have to resort to a small subset due to our test resource constraints. I used principles introduced in SRE (Software Reliability Engineering) test method by John Musa [1]. In short, we came up with most likely "operational profile" or the scenario that users may use most often and selected test cases that covered this scenario. Later, this subset was used as a maintenance test suite.

Test automation for client/server environment is a unique challenge by itself. SAS has very sophisticated homegrown test tool that you can use to submit test tables in a fully automated manner to a single host. Our test automation effort focused on the test programs themselves. Our objective was to run the test tables for any client and server combination from the driver host. A test program should be able to determine client and server hosts dynamically from the command line input, run the test case accordingly, collect server log and client log for comparison with a benchmark. This client and server test automation logic will be explained in detail shortly.

Test Portability

It is worthwhile to mention our portable test structures. It is not uncommon that we have host specific environments and access method specific options in any client and server combination. For instance, server name may vary by access method or security options may differ by operating system. It is very clear that we cannot afford multiple copies of the same test solely because of host differences. We made our tests fully portable by utilizing the SAS macro facility extensively. Like other macro languages, SAS macro facility is basically a text substitution tool with some conditional statement capability. The SAS system also provides a sysparm option (it reminds me of good old

MVS JCL) that you can use to pass in any character string that can be accessed within the SAS system. We exploit this option and pass in client, server, access method and other host specific information as a sysparm value and process this information, converting the single sysparm value into multiple global macro variables which are available throughout the SAS session. The portable tests are written with macro variables whenever necessary. These macro variables are resolved dynamically before test execution. Here is an example:

1) sysparm is specified with SAS invocation

```
sas –sysparm  mvs#h8x$tcp –autoexec shrauto
```

2) specified macro, shrauto, is called automatically and generates macro variables by parsing the sysparm value:

```
&server_node = sdcmvs
&client_node= vivaldi.unx.sas.com  (for h8x, or HP, machine)
&server_name=share1
&access_method=tcp
```

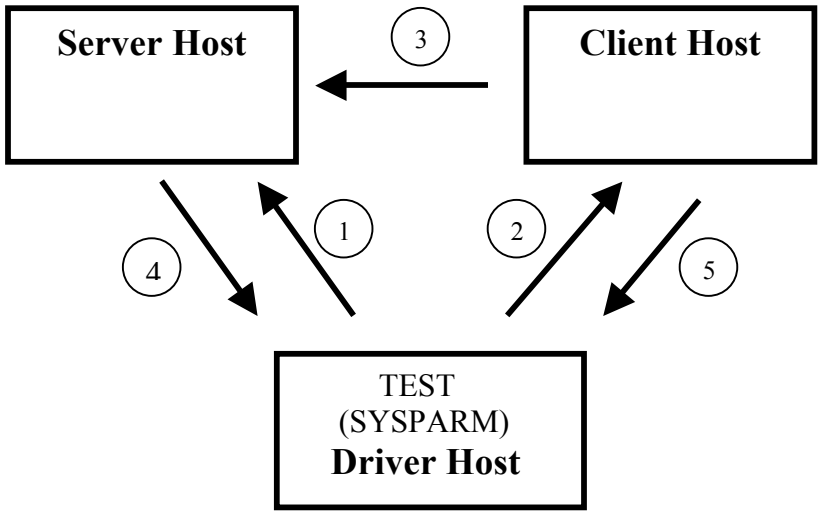
3) portable tests use these macro variables

```
options comamid=&access_method;
libname  mylib server= &server_node..&server_name;
```

All our portable tests are developed on our base development machine, which is HP UNIX, and get ported to all platforms on a weekly basis.

Version 6 Test Automation Scheme

As mentioned earlier, our objective was to run our test tables on any client/server combination from a driver host in a fully automated manner. We achieved this goal by using our own SAS/CONNECT product as a vehicle to connect (signon) to client and server hosts as depicted in Figure 1.



1 Sign on to server host and bring up share server
2 Sign on to client host and remote submit share test code
3 Share test runs from client host to server host
4 Stop the server and download the server log
5 Test finishes on client host

Figure 1. Version 6 test automation scheme

As the test begins execution on a driver host, it connects to the server host, brings up a SHARE server, connects to the client host and brings up a remote SAS session on the host. SHARE tests are remote submitted from the driver host to the client host and test results are returned to the driver host. At the end of the test, the server is stopped and its log is downloaded to the driver host for comparison.

This process was a major achievement in test automation for the client/server environment covering multiple platforms and their combinations.

Towards the end of the version 6 test cycle, we were able to run our test suites nightly in a fully automated fashion. When everything went well it produced fairly high level of test output. Version 6 was shipped slightly behind the schedule.

Version 7 a New Challenge

In his paper on test automation, Pettichord [2] appropriately pointed out that**, *"test automation always breaks down at some point"***. I totally agree. Test automation scheme is known to be vulnerable to product or version changes but ironically it is not that easy to predict the problems until you face them.

During version 7 development, we have experienced a great deal of frustration with our test automation process since our SAS/CONNECT product itself is under development for version 7 and fails time to time. Another drawback of this process is extremely high overhead for test set up such as signon, signoff, uploads of test information and downloads of server logs, etc. It is really handy for nightly cron jobs but rather painful to use during the day. We also notice that it requires some amount of rework when we try to recreate a problem in isolation without the SAS/CONNECT environment in the picture for developers.

Version 7 Test Automation Scheme

As an alternative, I have proposed and implemented a new SHARE server testing strategy using "Long Running Server". This strategy includes the following changes and advantages:

- A SHARE server needs to be brought up by hand or by a separate step on target host machine. To make this chore simple, I wrote a macro, %shrsvr, which will bring up server on any host. This SHARE server does not get stopped on every test. It stays up (long running) for at least the whole table. It eliminates the server manipulation overhead (signon to host, bring server up, bring down server, get server log for each test).
- For single user tests, which account for more than 90% of the entire SHARE test suite, we do not need to use a SAS/CONNECT remote session any more and all

tests run directly to the long running SHARE server. We still use the same test suites without any changes in test source or benchmarks by blanking connect related macros in the test with a new autoexec macro.

- Another change is that we no longer collect and compare server logs for each test any more. A rationale behind this decision is that if anything goes wrong on server, it should surface on the client side in one way or another. When that happens, we check out the server log manually. We also save a lot of overhead involving server log manipulation.
- One may point out that we lose a level of test automation that we enjoy since we have to bring up a SHARE server separately (not in the test program itself). To some extent, this is true. But it is rather easy to provide one more activity in the script that runs the regularly scheduled batch jobs. For nightly cron jobs, we bring up a long running server from a separate test table that houses the server program before we run the test table that contains single user tests. For multi-user SHARE tests, we still take advantage of the long running server and use SAS/CONNECT remote sessions for multiple clients.

Figure 2 below shows our new version 7 test automation scheme:

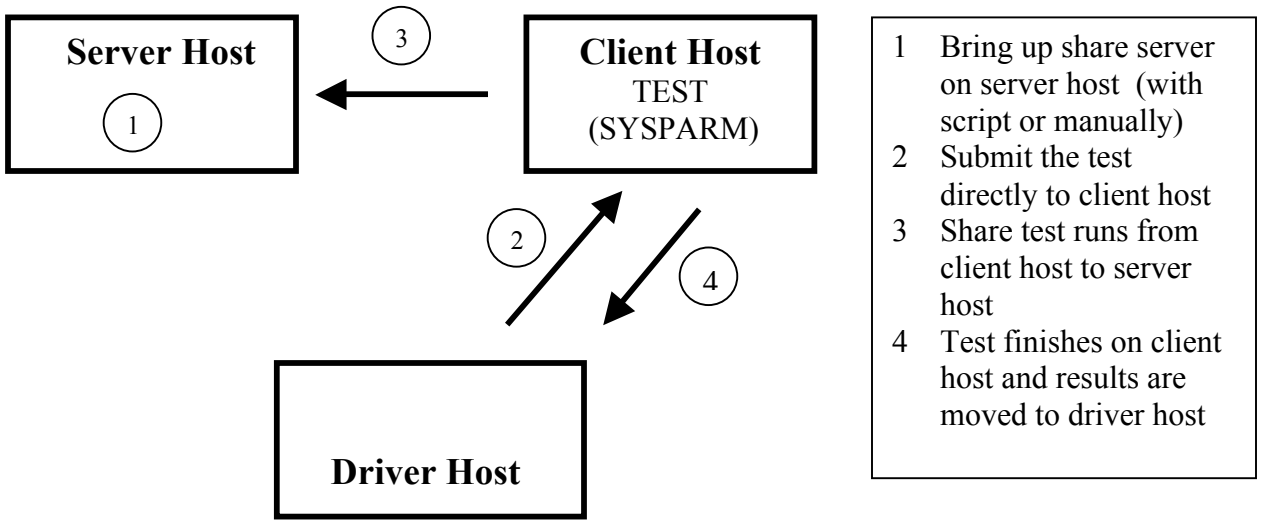


Figure 2. Version 7 test automation scheme

Notice that after the long running SHARE server comes up on a server host, test cases are directly sent to the client host for execution. This change eliminates the lengthy signon and signoff process that we used for version 6 and reduces the test execution time very significantly. Also, server logs are no longer collected for comparison.

Test Process Improvement by De-automation

We have noticed drastic improvement in test efficiency. For example, our level 2 test table that houses about 100 tests used to take 3 hours to complete with version 6 process, but with our new version 7 test automation scheme it took less than 25 minutes. This is one good example of increasing test output by reducing the level of test automation.

Our version 6 defect analysis [3] indicates that we got more than the expected number of defects from our tech support group and beta sites where they stress tested the version 6 SHARE product in a user environment. I have to acknowledge that version 6 SHARE server did not get much stress test since server's life cycle was very short. With the "long running server" strategy on version 7, the SHARE server runs longer and definitely runs under a stressed environment, which gives us a better idea on server survivability and "failure intensity"[1].

One big hidden cost of test automation lies in the fact that as the level of test automation grows, it becomes increasingly difficult to reproduce the defects in isolation. Defect description by testers gets lengthy and time consuming, and sometimes it is not well received by developers because it looks more complicated than it should. According to my experience, most developers are very reluctant to learn a test automation scheme or anything related to that subject. From the developers' point of view, recreating the defect in a simple and reliable manner is most important in debugging and fixing the defects quickly. To save both the testers' and developers' time in defect description and defect reproduction, it is very important to have a tool that allows one to run automated test cases interactively in isolation.

Running Automated Tests Interactively

As mentioned above, one of the shortfalls of test automation is the difficulty of defect reproduction in isolation. I believe good test automation strategy in general and especially for client/server software should include tools that facilitate interactive running of automated tests. During the version 7 cycle, I proposed and implemented two macros that make testers' and developers' job much easier.

The first macro is %shrsvr macro, which brings up a SHARE server on any host. By default, it brings up the same server that is used by automated tests. The second macro, %shruser, is for client side. On any client host, it provides all necessary test set up based on current environment and macro parameters and includes the test case that we want to run interactively. For example, if our automated test***, sqltest***, fails from UNIX client to MVS SHARE server during the batch test, we can easily recreate the same problem interactively as follows:

- Bring up sas session on MVS and issue : %shrsvr;
- Bring up sas session on unix and issue : %shruser(snode=mvs, test=sqltest);

With version 7, we have more test combinations due to our backward compatibility support. In other words, we have to touch combinations such as server on version 6 and

client on version 7 and vice versa. It is very obvious that full test automation on these combinations is not a good investment at all [2] considering the maintenance costs involved. As an alternative, we developed very comprehensive mega tests that contain coverage for all major functions and features and ran them interactively with the macros mentioned above on cross host and cross version combinations. This process gave us comfortable level of coverage on a vast number of combinations.

Icing on the Testing Cake

These two macros were very well received by both developers and testers and contributed heavily in strengthening developer / tester relations. At the end of the version 7 cycle, one of our developers sent us following comments:

"… I want to give kudos to SHARE testers. … those testing macros made testing, and more importantly debugging, almost a breeze. "

Our successful test organization and teamwork was very well depicted by Langston [4], in her award winning paper "How to have a perfect 'T' party".

Conclusion

Each and every software development and testing environment is different and has unique characteristics. In most cases, general test guidelines may not be directly applicable since software testing is more of a craftsmanship [5] rather than an industrial production. The same principle applies to test automation of multi-platform client/server software. Consequently, it is very important to understand the effects and implications of different levels of test automation for a given environment. My experience described in this paper is one good example of increasing the test throughput by reducing the level of test automation and the use of a macro facility that supplements the shortcomings of test automation in a client/server environment.

References

[1] J. Musa, "More Reliable, Faster, Cheaper Testing through Software Reliability Engineering – Overview ", Testing Computer Software conference, Washington, D.C., June 1998

[2] B. Pettichord, "Success with Test Automation", Quality Week'1996 conference, San Francisco, CA., May 1996

[3] H. Park, "611 IDB Defect Analysis", SAS Internal Testing News Letter, Cary, NC., 4th Quarter, 1995

[4] B. Langston, "How to have a perfect 'T' Party", STAR'98 East conference, Orlando, FL., May 1998

[5] P. Jorgensen, "Software Testing : A Craftsman's Approach", CRC Press, 1995