APPLICATION OF THE SAMGRID TEST-HARNESS FOR PERFORMANCE EVALUATION AND TUNING OF A DISTRIBUTED CLUSTER IMPLEMENTATION OF DATA HANDLING SERVICES

A. Lyon, A. Baranovski, G. Garzoglio, L. Loebel-Carpenter, R. Herber, R. Illingworth, R. Kennedy, A. Kreymer, A. Kumar, L. Lueking, W. Merritt, I. Terekhov, J. Trumbo, S. White, S. Veseli, FNAL, Batavia, IL 60510, USA, M. Burgon-Lyon, R. StDenis, Glasgow University, S. Belforte, INFN, Trieste, U. Kerzel, Karlsruhe University, M. Leslie, V. Bartsch, S. Stonjek, Oxford University, F. Ratnikov, Rutgers University, A.Sill, Texas Tech University

ABSTRACT

The SAMGrid team has recently refactored its test harness suite for greater flexibility and easier configuration. This makes possible more interesting applications of the test harness, for component tests, integration tests, and stress tests.

This new implementation of the test harness is a Python framework which uses XML for configuration and small plug-in python modules for specific test purposes

We report on the architecture of the test harness and its recent application to stress tests of a new analysis cluster at Fermilab, to explore the extremes of analysis use cases and the relevant parameters for tuning in the SAMGrid station services. One current testing application is running on a 128-CPU analysis cluster with access to 6 TB distributed cache and also to a 2 TB centralized cache, permitting studies of different cache strategies.

We have also studied the service parameters which affect the performance of retrieving data from tape storage. The use cases studied vary from those which will require rapid file delivery with short processing time per file, to the opposite extreme of long processing time per file.

These results are interesting for their implications with regard to Grid operations, and illustrate the type of monitoring and test facilities required to accomplish such performance tuning.

INTRODUCTION

The SAMGrid system is a large scale distributed system offering data storage, transfer and bookkeeping services to aid in the processing of peta-byte scale data from the D0 and CDF experiments, it is also being tested for use by MINOS and CMS. Unit, Stress and Performance Tests

Testing is an essential part of developing any large scale distributed system. SAMGrid tests fall into two main categories. The correctness of code is checked by 'Unit tests' which verify small sections of code in isolation.

While unit testing will catch many bugs, production quality code also needs to be stress tested under loads that resemble the environment that they will be deployed in. Particularly in multithreaded systems, this stress testing can reveal flaws that unit testing does not.

There is also a need for performance testing. Optimizing any tuneable parameters in a system requires producing a controlled load and monitoring the performance impact of altering various settings. As we will show, the SAMGrid test harness has proved particularly valuable for this kind of testing.

THE TEST HARNESS

Motivation for the SAMGrid Test Harness

Before the test harness was used, tests were written on an ad-hoc basis. Stress testing was performed using a mixture of Perl, Python, and Bash scripts which were difficult to maintain and configure. The output from stress tests was difficult to read as the output from parallel processes were interleaved. When the need arose to reconfigure the stress test harness for use at CDF, it became apparent that refactoring the test harness was the best way forward.

Features

In redesigning the test harness we aimed for a clean design that simplified test configuration and provided facilities for stress, unit and performance testing within a single unified framework. This framework was to provide configuration and output formatting features while requiring as little additional effort as possible from the test programmer. The test harness was rewritten in object oriented Python, using pyxml to allow XML configuration.

The harness is multithreaded and can be configured to run any test to completion before starting the next test, or

to start a separate thread for a test. Using this feature a unit test can easily be converted into a stress test, simply by running many unit tests concurrently. The harness will keep track of each thread and log its output on completion.

Configuration

Configuration is handled by a single simple XML file, detailing a suite of tests. Test suites may be set to loop any number of times.

Each test is configured by its test option subtags, which are key-value pairs. Global options may also be specified, and accessed by all tests.

Any test may be configured to depend on any other, and will run only if that test has completed successfully. Thread locking is used to ensure tests will wait until their dependencies have completed before running, and cycle detection is used to prevent accidental deadlock.

In order to provide more accurate simulation of real world load, randomised or exact delays may be specified between running tests. This is especially useful when attempting to simulate real world loads.

Output

A test's output is separated into standard input, standard output, a Boolean to indicate whether the test was successful and various timing fields. Separate fields are also available for providing reasons for the tests failure, or for indicating that the test failed to run because it's dependencies failed.

The harness produces either XML or HTML output. XML output provides an easy way for other programs to interpret the output of the harness, whereas HTML output is of more use if the results are being interpreted directly. Results are colour coded depending on whether the test passed or failed.

Realtime onscreen output is also available, however when large numbers of processes are run in parallel, messages from each are necessarily interleaved. File output is preferred as it allows messages from concurrent tests to be collected and separated, making interpreting their results much easier.

Documentation

Documentation can be automatically generated by the test harness using the various introspective description methods each *Test* object must provide.

Extensive documentation on using, configuring, and extending the test harness is available online[1].

Extensibility

Adding new tests has been made as simple as possible to encourage use of the test harness. All tests derive from a Test class shown in figure 1, and must provide details of any options it accepts.

The harness will then read in these options from the XML file, fork of a separate process if necessary, and start the test with the options the user specified.

No restrictions are placed on the test code other than that it must terminate and must produce a Boolean 'passed' output. Helper methods are available to run a command line application and pass or fail the test depending on the applications return code or on the presence of some string in standard output.

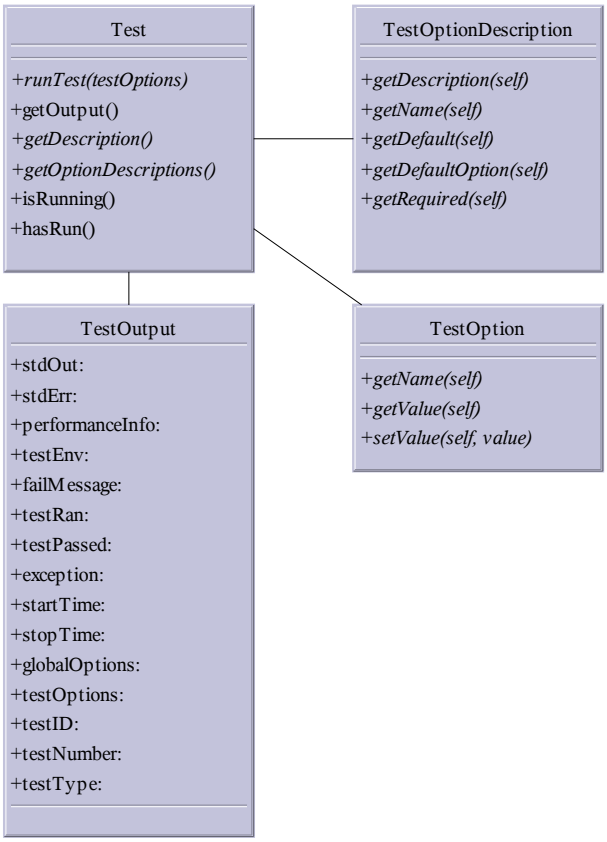

Figure 1: The Test Object

Performance Monitoring

In performance testing it is often necessary to monitor parameters throughout the lifetime of an entire suite of tests, rather than as part of a particular test. An example of such a parameters might be a systems load average.

The test harness provides a framework for performance monitoring code to be registered and polled at regular intervals. The monitor is registered in the XML file, and the value of the monitored code is logged throughout the duration of the test suite, formatted and output with the results of the test.

PERFORMANCE TUNING SAMGRID WITH THE TEST HARNESS.

One main use of the test harness is to exercise the SamGrid system under different conditions with different options chosen for SamGrid components. We have begun such a program to explore the SamGrid parameter space to search for an optimal set of options to run. The SamGrid stager service is responsible for transferring a data file to a worker node for processing. The source of the file may be another SamGrid station, a tape system, or local cache. The stager service has various throttles in order to limit the amount of network traffic. One such throttle is "max-transfers", which restricts the number of simultaneous file transfers that a stager will perform. If the number of needed transfers exceeds the max-transfers setting, those extra will be queued and wait for other transfers to complete.

When the cache on a station is empty, most data files come from a tape system. Fermilab uses Enstore[2] for tape services. We have noticed that when a SamGrid station is very busy and heavily accessing Enstore tapes, many tape transfers will wait in the Enstore queue and long file arrival times result. We seek to understand the role of the max-transfers parameter in this problem.

We use the SamGrid test harness as a mechanism for loading a test SamGrid station with test projects. A project is a SamGrid request to run a particular application on a set of files. The project may involve running one job, or many parallel jobs. In this case, we have the test harness load a test station with 200 projects, each with one job asking for 10 different files, all coming from tape. There is little overlap between the sets of files. The test application merely sleeps for a minute between file transfers. The projects are started in rapid succession, and then the station is left alone to fulfill the file transfer requests. We use SamGrid monitoring tools to examine the course of events on the station.

We show here results from running two extreme cases: allowing each worker to have five simultaneous transfers (max-transfers=5) and just one transfer (max-transfers=1).

Figure 2 shows the number of projects running on the SamGrid test station vs. hours since the start of the Test Harness. One sees a fast rise and then a long decay as projects slowly complete. For the max-transfers=5 case, the entire test ran for about eight hours, while the test took ten hours for the max-transfers=1 case.

Figures 3 and 4 show histograms and a box-and-whisker plot of the number of hours projects were running. For the max-transfers=5 case, the mean duration for a project was five hours, while the mean was eight hours for the max-transfers=1 case.

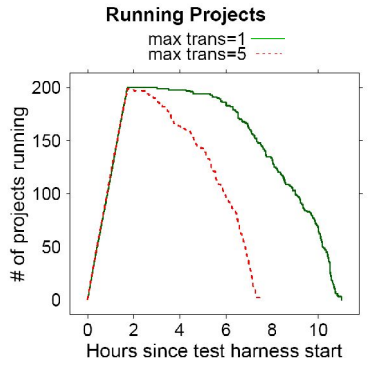

Figure 2: Number of running projects,

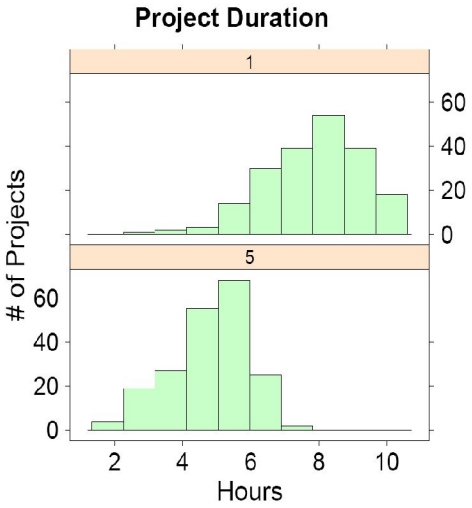

Figure 3: Project Duration

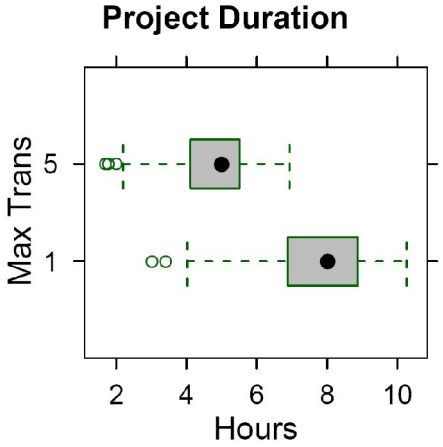

Figure 4: Project Duration, Box and Whisker

Figure 5 shows box-and-whisker plots for the number of hours a job was waiting idle for a file delivery. It is interesting to note that the average wait time for the max-transfers=1 case is shorter than the max-transfers=5 case. The reason for the shorter wait times for max-transfers=1 is clear from viewing Figure 7. Shown are box-and-whisker plots for each tape drive (mover) indicating how long a file transfer request waiting in the Enstore queue before being serviced.

The shorter Enstore queue wait time for max-transfers=1 is due the reduced number of transfers submitted to the Enstore tape request queue, as shown in Figure 6

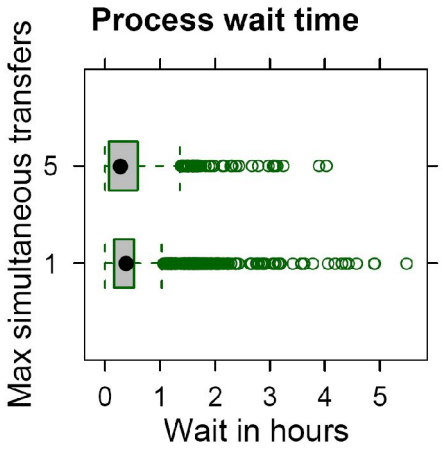

Figure 5: Process wait times

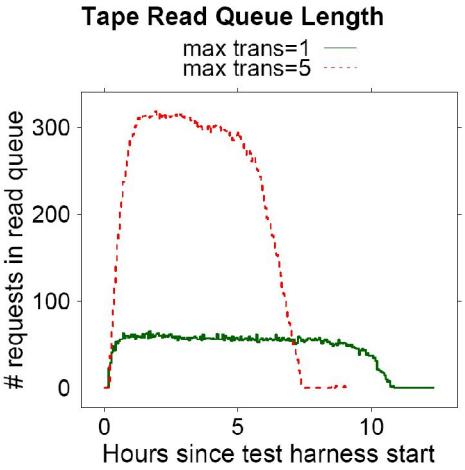

Figure 6: Enstore read queue length

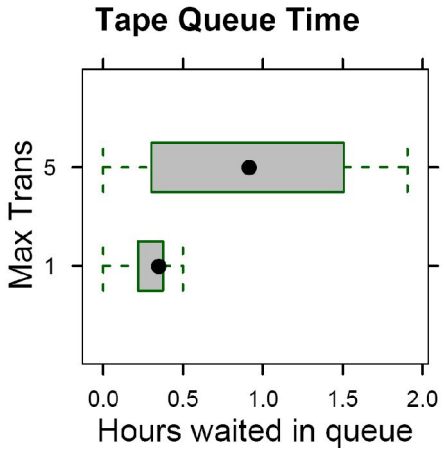

Figure 7: Tape Queue Time

This brief example illustrates the complexities of optimizing the SamGrid system. More detailed investigations are underway to understand the connection between SamGrid parameters, file delivery performance, and tape system load.

CONCLUSIONS

Tests like those described here were difficult before the refactoring of the test harness system. Now, we can perform such system wide studies easily, along with other tests such as installation verification. We expect to use the new test harness extensively to learn more about the performance potential of SamGrid

ACKNOWLEDGEMENTS

We would like to thank Fermilab Computing Division for its ongoing support of the SAMGrid project, and especially the CCF, CEPA, and Run II Departments. We would also like to thank everyone at D0 and CDF who has contributed to this project. We would also like to thank the UK Particle Physics and Astronomy Research Council (PPARC) for it's support of e-science research.

REFERENCES

[1] http://cdfsamth.fnal.gov/docs/
[2] http://www-isd.fnal.gov/enstore/design.html