CHARLES A. BACON

Testing in the Grid Environment

A well-run test suite is an invaluable aid to application support both in a cluster and in the Grid. The difference is that the Grid coordinates non-centralized resources that cross organizational boundaries. In this column we present guidelines for establishing a successful test suite, with special focus on the challenges raised by distributed sites without centralized control.

A test suite should be designed with the goals of the testing organization kept in mind. Is the infrastructure intended to support general users? Or is it intended to run a specific application? In either case, you should set up tests that prove that the system does what you need.

Well-written tests serve at least two distinct functions. They provide notification of failure, and they assist in the debugging process. Tests should be run often enough that they detect problems before the user community does. The tests should also be precise enough to place the problem in a well-defined category.

We assume here that the cluster administrators will provide the supporting software infrastructure for applications. Of course, users can construct their own software infrastructure without involving the administrators.

Internal and External Testing

Because the success of a cross-site application depends on the success of individual sites, the first set of tests should apply to a single administrative domain. To that extent, these tests are like any other tests you would run in a cluster environment. The extra difficulty is that different sites may implement their cluster systems in different ways. This situation can expose flawed assumptions present during the design of a test.

For example, a functionality test that passes at one site may fail at another site even though both expose the same behavior to the end user. This situation is particularly noticeable during tests using `grep` to look for particular strings that are present in one environment but not in another. Because multiple, equivalent implementations may exist, the tests must target the desired functionality, not implementation side effects.

Another set of single-domain tests focuses on the available software. Shared libraries, versions of compilers, linkers, other executables, or running services can all affect the success of an application. This is why most cluster systems provide tools to manage the user environment. Regardless of the tools used, there should be tests available to ensure that the appropriate software is in the right place. For instance, you might run a series of autoconf tests under the different user environments to ensure that the correct versions appear in the user's path. And, of course, the environment itself should be well documented.

The second set of tests should be cross-domain tests. These include tests that run on a single site but are launched remotely, as well as larger cross-site tests. In the single-application scenario, one good test is running a copy of the application itself. Even in that case, the suite of tests should include incremental tests of the application's requirements.

An example of this incremental approach is testing the Grid Resource Allocation Manager (GRAM) from the Globus Toolkit™. You can run `globusrun -a` to test the authentication operation of the GRAM gatekeeper. This action will test that the port is open to the firewall, that the service is up, that the Certificate Authority certificates are in place, that the system times are within five minutes of each other, and that the local user is in the `grid-mapfile`. This basic remote test also includes several subpieces that can be tested locally. Running the tests both locally and remotely can help locate failures. Then, after you have done authentication only with `-a`, running `globusrun` without `-a` will test the operation of the GRAM jobmanager, which is responsible for converting the job request into a local scheduler submission. `globusrun` will also test the ability of the remote site to send callbacks to the client. If the test works within the site but not remotely, the problem is more likely to stem from the network callbacks than from the scheduler.

When Things Go Wrong

The Grid environment can increase the complexity of your computing system. It includes factors outside your control. For example, there may be redundant power, multiple networks, and good air conditioning at your site, but between you and another site there are many possible failure points. Much like disk arrays, increasing the number of pieces in the system can lower the mean time between failures.

In order to address this situation, a test harness is essential. A good test harness enables frequent automatic test execution and voluntary manual test execution, collects the reports, and provides several views on the test results. The test harness also makes sure that when tests fail, those responsible find out immediately. TeraGrid (*teragrid.org*) makes their Inca Test Harness and Reporting Framework (te*ch.teragrid.org/inca*) available as a possible starting point. If your tests are related to code developed in CVS, Tinderbox (*www.mozilla.org/tinderbox.html*) may also be useful.

False alarms are the bane of a testing system. If they happen frequently, the tests will be ignored. You should also be on the watch for false test success. A good practice is to test the system in a state that should provoke failures. For instance, you may bring down one of the services, launch the test suite, and verify that the tests you expect to fail during this outage do indeed report failure. There is nothing worse than placing your faith in an unreliable test. In other words, don't forget to test the tests!

Your choice of tests affects your ability to narrow the scope of the problem. Equally important, however, is the frequency of the testing. Run tests often so they can tell you immediately when things go wrong. If tests worked yesterday but not today, you have a well-defined set of changes to search. The more time that passes between tests, the harder it will be to find the relevant change.

Machines for Testing

Change management is important to the success of a large-scale system. A best practice for critical systems is to have multiple identical copies of the system available. One can be used for development, another for quality assurance, and still another for production. Thus, changes can be tested in one environment before being implemented in the next. It should always be possible to roll back changes to a previous version.

The test environment must be documented and reproducible. Cluster management software is a good way to ensure that if a node is destroyed, a new node can be built as an exact replica and reproduce any errors.

The distributed nature of the Grid makes the process more complicated than the single-site case. For one thing, not all clusters use the same change management system. Furthermore, the systems involved can be some of the largest computing clusters in existence. It's probably a fantasy that a site will have two identical clusters and be able to dedicate one of them to testing. A reasonable alternative is to have a miniature copy available so that you can evaluate new software, patches, and kernels on the test system before rolling out the larger change. Unfortunately, some errors may show up only in the larger environment.

Multiple sites will rarely agree to simultaneous infrastructure outages for implementing changes on the main systems. As a result, you must ensure that sites can perform upgrades independently.

The Map Is Not the Territory

Mapmakers understand that their product is just a representation of the world. If the map disagrees with the world, the map is incorrect. In the same way, if tests succeed but production code fails, something needs to be fixed.

It's easy for developers and administrators to neglect to test the actions of a real user on the Grid. Ideally, the tests would be written by a separate team that was motivated to stress the installation. Fortunately or not, depending on your perspective, many applications make the need for designated crash teams unnecessary. They are the reminder that, even if all your tests pass, something may be wrong on the system.

Typically you can test your Grid environment by enlisting the help of a subset of your users who have good test cases already known to work in other environments. In the early stages of constructing your Grid, a failure will usually mean that some kind of dependency has failed. At this point, it is a good time to create new tests, using as small a piece of code as will exercise the failure. Alternatively, it may be a time to reconcile two different choices to solving the dependency problem. Consider using Bugzilla (*bugzilla.mozilla.org*) as a cross-site trouble ticket system to track such issues.

As the Grid environment matures, the nature of the failures will change. Resolving these problems may require small refinements to the environment or modification of the existing tests. Sometimes, however, a failure may not be due to the system infrastructure at all, but rather to an application error that has gone undetected until now.

The Reward

Participating in the Grid environment can be rewarding, but success is preceded by a lot of hard work. Having a good test suite will help in taking a Grid project from concept to implementation. The tests can help validate that the system is working as designed and can speed the process of recovering from failures.

Globus Toolkit is a registered trademark held by the University of Chicago. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38 and under Contract DE-AC03-76SF0098 with the University of California; by the National Science Foundation; by the NASA Information Power Grid program; and by IBM.

Charles Bacon is a member of the Globus Alliance who works on Grid administration and testing. He's an active participant in the discuss@globus.org community.