Testnodes: a Lightweight Node-Testing Infrastructure

R Fay and J Bland

Department of Physics, University of Liverpool, Liverpool, L69 7ZE, UK

fay@hep.ph.liv.ac.uk

Abstract. A key aspect of ensuring optimum cluster reliability and productivity lies in keeping worker nodes in a healthy state. Testnodes is a lightweight node testing solution developed at Liverpool. While Nagios has been used locally for general monitoring of hosts and services, Testnodes is optimised to answer one question: is there any reason this node should not be accepting jobs? This tight focus enables Testnodes to inspect nodes frequently with minimal impact and provide a comprehensive and easily extended check with each inspection. On the server side, Testnodes, implemented in python, interoperates with the Torque batch server to control the nodes production status. Testnodes remotely and in parallel executes client-side test scripts and processes the return codes and output, adjusting the node's online/offline status accordingly to preserve the integrity of the overall batch system. Testnodes reports via log, email and Nagios, allowing a quick overview of node status to be reviewed and specific node issues to be identified and resolved quickly. This presentation will cover testnodes design and implementation, together with the results of its use in production at Liverpool, and future development plans.

1. Introduction

1.1. The Problem
In late 2007 the largest part of the Liverpool HEP cluster was the MAP-2 supercomputer, which had been ranked 86th in the TOP500 [1] when it was commissioned in 2003. By 2007 it had inevitably dropped out of the TOP500 but remained a significant computing resource.

But as it aged, system failures naturally became more frequent, a problem exacerbated by the size of the system – 24 racks of single-core worker nodes, 940 in total.

These failures often led to ‘black holes’ in the batch system; nodes that would receive jobs, but then instantly fail them, effectively emptying the queue and resulting in intermittently high failure rates, and relatively poor reliability and efficiency. This required a high level of monitoring and manual intervention to remove problem nodes from the batch cluster while they were repaired. In particular, failures that occurred outside working hours could result in extended periods of degraded batch system performance before the failure could be addressed, despite best efforts from the system administrators.

1.2. Solution
Nagios [2] was in use for monitoring, but was under a heavy load, struggling to keep up with the level of testing already in place. Increasing the frequency of testing to a level sufficient to adequately minimise ‘black holing’ on the cluster was not feasible at the time with the resources available.
Additionally, implementation of the desired framework within Nagios presented some challenges and limitations.

Monitoring the batch system itself, taking nodes offline when jobs failed, was an option but could only detect failures after the event, not before, could not detect when failures had been resolved without additional effort, and could not detect errors where the fault resulted in a failure within the job payload itself (e.g. where software dependencies expected by the job have become unavailable), rather than a failure from the perspective of the batch cluster.

Some batch systems, including that in use at Liverpool, Torque [3], support the integration of node health check scripts presenting another potential solution. This approach was considered, but ultimately rejected. In Torque, the check is initiated and executed on the worker node by the pbs_mom daemon (the batch system’s worker node agent), which blocks until the health check is completed, raising potential issues with responsiveness and placing unnecessary constraints on test execution time. Given the nature of the test (looking for unexpected system failures amongst other issues) and the environment it would be running in (potentially extremely heavily loaded worker nodes), a requirement for health check scripts that could be guaranteed to never block and to always complete in a short period of time was not seen as entirely realistic. Additionally, the error state is reported only through setting a node attribute message to be detected by the batch system scheduler, which can then set the node status to ‘down’. This approach was considered to have several potential problems. Firstly, relying on the node to test itself and successfully report errors essentially implements a ‘default online’ state – that is, the node is designated online unless an error is successfully reported (as opposed to the node being designated offline unless a node health check is successfully completed). A consequent possible scenario could have the test script failing in such a manner as to exit without successfully reporting a failure, resulting in the node remaining online despite having potential issues. Secondly, the scheduler only provided the ability to mark the node down, rather than offline. Jobs running on a ‘down’ node would be considered unviable by the scheduler and killed, whereas nodes running on an ‘offline’ node would be allowed to continue, potentially allowing successful completion. As not all node failures would result in all jobs failing, the option of making the nodes ‘offline’ was considered preferable, preventing new jobs running on the node, but still allowing existing jobs to attempt to complete. Thirdly, this approach did not easily provide a means of avoiding a ‘flapping’ scenario – where an error is intermittently detected, initially resulting in the node being marked down, preventing new jobs starting and failing existing jobs, only for the error to then disappear, putting the node back online and starting new jobs, before the error returns causing those jobs to fail.

A hybrid solution was also considered, using the node health check script feature of Torque, but rather than using the scheduler to detect and respond to errors, a cron job would run a site-developed script on the server to scan worker node status for error attributes instead. This would allow greater control of node status (e.g. marking nodes ‘offline’ instead of ‘down’), ‘flapping’ detection, and reporting. However, due to the potential issues involved with relying on the pbs_mom daemon running on the node to execute the node health check script and report results, as outlined in the previous paragraph, it was decided that as well as developing a script to respond to errors and control worker node production status, the script would also handle the initiation and execution of the node health check script on the nodes.

Consequently, the solution Liverpool adopted was a simple monitoring and node production status control framework written from scratch.

1.3. Initial Outcome

‘Testnodes’ was implemented in Python [5] with a single focus to testing the cluster; essentially asking the question, “Is there any reason why this node should not be available for jobs?” and then marking the node offline or online in the batch system accordingly. Failure of the node check script to complete and return a result is considered a reason to mark the node offline, giving a conservative ‘default offline’ approach. The ability to read a list of nodes to be considered automatically offline and...
excluded from testing was added, enabling the issue of ‘flapping’ nodes to be addressed, as well as allowing nodes to be taken offline from the batch system for other reason (e.g. other testing, experimentation, or benchmarking). Tests (see table 1.) were initially created based on known reasons for previous node failures, with a few further tests added over time as new failures and error conditions occurred. With the tests run in parallel from a management node, it became possible to thoroughly test the entire MAP-2 cluster within a few minutes.

Table 1. Tests carried out on worker nodes by testnodes.

| Check                        | Method                                      |
|------------------------------|---------------------------------------------|
| Disk partitions not read-only| Check return of ‘touch’ command             |
| CVMFS not crashed            | Check for cvmfs stacktrace files            |
| CVMFS reporting OK           | Check output of ‘cvmfs_config probe’        |
| CVMFS mounts accessible      | Check output of ‘cvmfs_config probe’        |
| rpc.statd process running    | ps command                                  |
| /dev/null character device   | ‘ls –l /dev/null’ output check              |
| Disk SMART status            | ‘smartctl –H’ command                       |
| Software shares mount        | Check nfs mounts                            |
| Disks not full               | Check output of ‘df’ command                |
| Running kernel up to date    | Output of ‘uname’ command                   |
| LCG certificates up to date  | Check version of lcg-CA RPM                 |
| emi middleware up to date    | Check emi-version RPM                       |
| Middleware configured        | Check date and last lines of yaimlog        |

Since introduction, testnodes has carried out over half a million tests of the entire Liverpool cluster. It had an immediate impact on availability and reliability (as measured by EGEE availability and reliability reports [4]), which increased from 87% and 89% in the prior two quarters to 93% and 96% in the next respectively. This is largely attributable to failing nodes being immediately removed from production, eliminating costly periods of failure, particularly those occurring outside normal working hours. By October 2008, with refinement of testnodes, Liverpool was one of only seven (out of 263) sites to achieve 100% reliability and availability, despite running one of the older clusters at the time. To date, Liverpool has typically maintained over 98% availability and 99% reliability on a consistent basis.

2. New Design
While successful in achieving its goals, the original implementation of testnodes was relatively crude. Developed as a proof-of-concept and quick solution to an immediate problem, it was written in largely monolithic code with embedded configuration and did not lend itself to being easily adapted for other batch systems or other purposes. While the script was used by other sites, e.g. Lancaster, these limitations constrained its uptake particularly as some sites, including Lancaster, moved away from Torque.

The redesign (figure 1) uses a more modular approach, with the core framework relying on configurable plugin modules to interact with the local batch system, as well as defining the test(s) to be carried out and any resulting actions or notifications to be performed.
Both the original implementation and the redesign are written in Python, with the original using Python 2.4 and the new version Python 3.3. Both implementations follow the same general path of execution, going through an initialization process in which the list of subjects to be tests is obtained, followed by executing the tests on the subjects in parallel and carrying out any consequent actions. However, the re-design allows for additional configuration via a configuration file, along with the specification of plugins for activities.

2.1. Configuration

The new design allows a number of options to be set in a separate configuration file that would previously have required directly editing code in the main body of the program to alter.

2.1.1. Maximum number of simultaneous test threads. Depending on the nature of the test(s) being carried out and the consequent load on the host system, the maximum number of simultaneous test threads can be adjusted to maximise performance without overloading the system.

2.1.2. Timeouts for individual tests and complete testing. Depending on the nature of the test in use and the state of the environment in which it is running, it can be the case that completion may take a varying amount of time, or may even not be guaranteed to happen at all. The timeouts before a test is terminated can be set, to take into account the test(s) in use and the local environment.

2.1.3. Plugin directory. The location of the directory in which the various plugins can be found can be set, allowing (e.g.) the temporary use of a separate set of plugins for development and testing.

2.1.4. Input plugin(s). Defines the plugin(s) which provide the list of subjects to be tested.

2.1.5. Test plugin(s). Defines the plugin(s) which provide the test(s) to be run on the list of subjects.
2.1.6. Action/notify plugin(s) to act on individual test results. Defines plugins to act on individual test results. These can be run unconditionally, or dependent on the return codes of the tests.

2.1.7. Optional summary plugin(s) to act on all results. Optionally, a plugin can be run on the summary of results. This can be used, for example, to email a summary of tests or to update a number of Nagios passive tests at once.

2.2. Input Plugin
The input plugin returns a list of subjects to be tested, typically a list of nodes obtained via an interface to the batch cluster, e.g. this can interact with Torque’s ‘pbsnodes’ command to obtain a list of worker nodes from the batch system. Other information (e.g. initial state) can be maintained within the plugin module for use by subsequent plugin functions (e.g. actions).

2.3. Test Plugin
The test plugin defines the test to be carried out on a subject. It typically must be thread-safe; although it is possible to run effectively in a single-threaded mode by setting the maximum number of threads to one, this would not usually be desirable. The current implementation uses a test script installed and kept up to date on the nodes by Puppet, executed securely via SSH.

2.4. Action/Notify Plugin
The action plugin performs an action (which can be a notification action), which can be unconditional (will always be run), conditional on test results, or a default action where there is no matching condition (note that this would be run in addition to any unconditional actions). The current implementation marks nodes which have failed tests offline, marks nodes which are already offline but are now passing tests online, notifies administrators of results by email, and updates the status of passive tests configured in Nagios.

3. Conclusions and Future
Testnodes has proven to be a useful solution at Liverpool, for both grid clusters and local batch systems. Use of testnodes has resulted in improved reliability, availability, and efficiency of jobs, as evidenced by EGEE/EGI SAM reports and EGEE/EGI accounting [6], with a reduced workload on system administrators. The redesign in progress provides a platform for further adaptations and optimizations, with the use of plugins providing a cleaner implementation and enabling functionality to be extended and adapted to other batch systems in a clear and well-defined manner.

References
[1] MAP-2 system at TOP500 http://www.top500.org/system/173158
[2] Nagios monitoring system http://www.nagios.org
[3] Torque Resource Manager http://www.adaptivecomputing.com/products/open-source/torque/
[4] Service Availability Monitoring reports http://sam-reports.web.cern.ch/
[5] Python http://www.python.org
[6] EGI accounting http://accounting.egi.eu/egi.php