Title: CBTS (Continuous Build and Test System)

Author(s): Charles N. Zeeb
Antony J. Zukaitis

Intended for: NECDC|06

Los Alamos National Laboratory, an affirmative action equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.
CBTS (Continuous Build and Test System)

Charles N. Zeeb and Anthony J. Zukaitis

Abstract LA-UR: LA-UR-06-5774

Since codes need to run on an ever changing number of machines and environments, there has long been an interest in automated testing environments and several have been developed. Unfortunately, the use of these systems has often been limited and usually require constant upkeep.

There are several features that would make an automated test system more appealing. One is that past results should be easy to store and search. Another is that the test system should transfer easily from one platform to another. Third, it should be possible to control the environment in which the tests are run.

To attempt to implement these features, CBTS (Continuous Build and Test System) was created. While efforts have been made to make CBTS general, currently it is integrated with the current MCNP Make system. To handle the storing and searching of past results, CBTS is tightly integrated with a SQL database (PostgreSQL).

CBTS is written in Perl which reduces but does not eliminate platform specific problems. A particular emphasis of the system is to make it relatively easy to define the environment as needed. The system has been shown to work on a variety of platforms including Linux, OSF1, and Mac OS X in both serial and parallel. Also, it has been run with and without the LSF queueing system.
CBTS (Continuous Build and Test System)

Charles N. Zeeb  
(czeeb@lanl.gov)

Anthony J. Zukaitis  
(zukaitis@lanl.gov)
Types of Tests Done by CBTS

- Database run of a test set
  - Test set description is read from the database
  - Results of builds and tests stored in the database
  - Builds and tests are done using the latest version of the code checked out from CVS
  - Done after new code is committed or on a regular schedule such as nightly
Types of Tests (Cont.)

- Developer run of a test set
  - Test set description is read from the database
  - Results of builds and tests are NOT stored in the database but are compared to the expected results in the database
  - Builds and tests done on a user specified copy of the code, usually the user’s current work on the code
Types of Tests (Cont.)

- A program exists allowing the developer to do individual builds or tests in a CBTS defined shell environment
- Independent run of a test set
  - Test set description is read from a test set description (ts) file
  - No information is read from or stored to the database
Features of CBTS

- Flexible system for specifying conditions for builds and runs
- Results stored in a database
  - Easy storage and searching of results
  - Automatically tracks changes in build and run environments
  - Able to define exceptions which describe acceptable results for individual tests
Build and Run Environments

- Perl files specify possible environments for each machine (compilers etc.)
- All builds and runs are done in environments created “from scratch”
  - Remove problems from undesired environment variables
  - A program exists to allow users to do individual builds and runs in this environment
MCNP Tests

- CBTS currently used with MCNP
- Tests done using MCNP’s Makefiles
- Results compared by diff command to templates
  - Linux, OSF1, Windows
  - If not same machine as template, a “fuzzy diff” that ignores the last significant figure is used
- Many machines do not match templates
Exceptions

- Database can store exception for each test which is an expected diff file
- If a test’s diff file matches its exception, that is a “defined pass”
- While it is hoped to remove all “defined passes”, this allows changes in results to be identified and corrected until that occurs
Sample CBTS Hierarchy

```
  Test Set
   /  \
  /    \nBuild  Build  Build
   |      |
  /  \
Test Target  Test Target  Test Target
   |      |      |
  /  \
Run  Run  Run
   /  \
Test Suite  Test Suite  Test Suite
   /  \
Test Test Test Test Test Test Test Test Test
```
Sample CBTS Test Set Description (ts) File

machine=flash test_set=nightly
B "pgi 5.2-4 lampi/gcc latest rossi plot cheap" CONFIG="portland rossi lampi cheap"
    FCVER=5.2-4 MPI_VER=1.5.14 CC_VER=3.4.3
T -C ROSSI test1
T -C ELECTRONS test1
T test
B "intel/lampi/gcc latest rossi cheap" CONFIG="intel rossi lampi cheap"
    FC_VER=9.1.033-f MPI_VER=1.5.14 CC_VER=3.4.3
T -C ROSSI test1
R NMPI=4 NTRD=4
T -C ELECTRONS test1
R NMPI=4 NTRD=4
T test
R NMPI=4 NTRD=4
Hierarchy I: Test Set

- Highest level of the hierarchy
- Two parts
  - Test set name
  - Machine
- Specified on the first line of a ts file
- Database automatically records changes to the basic machine description (from the "uname -a" command)
Hierarchy II: Build

- Two classes of options to “make build” (for building the code)
  - CONFIG="...." MCNP specific compile options
  - Other environment variables such as FC
- Specified by lines starting with “B” in a ts file
- “B” line also includes a build name for identification (entered in double quotes)
- Builds can pass or fail
- Database automatically records changes to the shell variables describing the environment for the build
Hierarchy III: Test Target

- Suites of tests require two arguments for make command
  - Test target such as test1
  - Specify the directory where the tests are (e.g. -C ROSSI)
- Specified by lines starting with "T" in a ts file
Hierarchy IV: Run

- Can specify any number of MPI processes and/or OMP threads
- Defaults to a sequential run
- Specified by lines starting with "R" in a ts file
- If errors are serious enough, target/run make command itself can fail to complete
- Database automatically records changes to the shell variables describing the environment for the run
Hierarchy V: Test Suite

- One test target can run several test suites
- CBTS describes a test suite as all results in a directory
- Test suite states:
  - Pass: All tests results give zero-sized diffs
  - Defined pass: At least one test a defined pass
  - Fail: At least one test does not match the template or the exception
Hierarchy VI: Test

- Level at which exceptions are stored
- Three possible outcomes of a test
  - Pass: Zero-sized diff file
  - Defined pass: Diff file matches exception
  - Fail: Nonzero-sized diff file that doesn’t match the exception
Database Schema

- Middle column: Definition of the test sets in the database
- Right column: Results of the runs of the test sets
- Left Column: “Everything else” including automatic tracking of machine/environment changes