COLLABORATIVE TESTING SYSTEM

USERS GUIDE

VERSION: 1.0.9

1 MARCH, 2006

[AUTHORS: EUGENE DOUGHERTY, DAVID AUBREY, ROBERT ROBEY ALFRED TORREZ]

CONTENTS

1 INTRODUCTION

The **C**ollaborative **T**esting **S**ystem is a testing system for parallel MPI based software. It is a collaboratively designed and developed system that will improve project software testing effectiveness, maintainability, extensibility, portability, and performance.

CTS is a modular system written mostly in object-oriented Perl consisting of a principal script called “cts” which contains the user command line interface and the following modules:

1 System: Provides platform independence.

2 TestHarness: This module creates run directories, runscripts, and runs the test problems.

3 Comparator: A stand alone diff tool to determine pass/fail result status.

4 Reporter: Creates reports: text, email and html.

5 Database: Data manager.

2 Currently Supported Systems

- Various Linux clusters with full kernels and bproc based kernels. -- Lightning, Flash, Cadillac at LANL
- HP/Compaq/DEC OSF1based systems – Q systems at LANL
- SGI – Bluemountain at LANL
- Linux Workstations

2.1 Batch Systems

Currently LSF and the ???? batch system on White. There is also support for no-batch systems such as workstations.

2.2 MPI

Currently, the following MPIs are supported.

- MPICH
- LAMPI
- OpenMPI
- MvPICH
- HP/Compaq/DEC OSF1 Vendor MPI
- SGI Vendor MPI

3 SETTING UP CTS

3.1 Obtaining CTS

CTS is available from SourceForge as a gzipped tar file. Developers can also checkout the source directly from CVS.

[At Los Alamos National Laboratory (LANL), CTS can be downloaded or checked out of the local SourceForge repository or retrieved from ICN NFS disk space at */usr/projects/packages/xtools/cts*.]

3.2 Configure

Type: `make config INSTALL_DIR=</installdir>`

3.3 Build

Type: “make build”

3.4 Install

Type: “make install

The successful installation will result in the creation of directories bin, man, lib, and html in the install directory. CTS is ready to run.

3.5 Examples

Type “make examples

The examples are a prototypical project setup with a build using autoconf and automake and then running a small test suite.

3.6 Combined

Type “make INSTALL_DIR=</installdir>

The makefile will configure, build, install and then run the examples.

4 Input Files

CTS executions rely on defaults and automatic configurations as much as possible, but three critical input files must be present:

4.1 *.cts File

The *.cts file contains key project information that the CTS needs in order to run the code. It contains executable names, locations, test suites to be run, desired reports, etc. See example *.cts files in the examples directory in the source code. This file can be specified in the command line execution of the CTS, or can be specified via the environment variable CTS_FILE. Example names include "nightly.cts" and “interactive.cts”.

```
# This is an example.cts file
# VERBOSE
PARALLEL_EXECUTABLES    : cpi=../src/cpi
PARALLEL_EXECUTABLES    : fpi=../src/fpi
DEFAULT_NUM_CPUS        : 4
DEFAULT_TIME_LIMIT      : 0:30
#DEFAULT_TESTSUITES      : example.suite
TESTSUITE_DIRECTORIES   : cts_tests
TEST_DIRECTORIES        : cts_tests
REPORTS                 : text html
PROJECT                 : CTS
if (`uname -n` == /^ffe?\d|^flash/) then
   SYSTEM_NAME : flash
   #BATCH : nobatch
   #QUEUES : largeq longq
endif
if (`uname -n` == /^l\d/) then
   SYSTEM_NAME lambda
   BLOCKED_QUEUES          : intq
   QUEUES : largeq longq
   MPIRUN : /usr/local/bin/mpijob mpirun
endif
if (`uname -n` == /^qsc\d/) then
   SYSTEM_NAME qsc
   BLOCKED_QUEUES          : devq
endif
if (`uname -n` == /t\d|theta/) then
   SYSTEM_NAME theta
endif
if (`uname -n` == /^cadillac|^tit??/) then
   SYSTEM_NAME workstation
endif
if (`uname -n` == /^pink/) then
   SYSTEM_NAME pink
endif
if (`uname -n` == /^q\d/) then
   SYSTEM_NAME Q
endif
if (`uname -n` == /^cx\d/) then
```

```
   SYSTEM_NAME CX
   BLOCKED_QUEUES          : devq
endif
if (`uname -n` == /^c[ab]/) then
   SYSTEM_NAME C
endif
if (`uname -n` == /^b\d/) then
   SYSTEM_NAME Blue
endif
if (`uname -n` == /^white/) then
   SYSTEM_NAME White
endif
if (`uname -n` == /^l[lc]/) then
   SYSTEM_NAME Lightning
endif

if (`printenv CRONJOB` == 1) then
  CROSS_PLATFORM_BASE     : /usr/projects/packages/xtools/nightly
        CROSS_PLATFORM_PATTERN        :    ${system}/${Fcompiler}_${mpi}_${type}/
cts_results.txt
  CROSS_PLATFORM_REPORTER : Flash
  CROSS_PLATFORM_GROUP    : cts-dev
  CROSS_PLATFORM_MODE     : 0660
  CROSS_PLATFORM_MAIL     : cts-team@lanl.gov
endif
```

The following is a list of keywords that are recognized in the .cts file with a brief explanation of the function and possible settings.

BATCH
BLOCKED_QUEUES
CROSS_PLATFORM_BASE
CROSS_PLATFORM_GROUP
CROSS_PLATFORM_MAIL
CROSS_PLATFORM_MODE
CROSS_PLATFORM_PATTERN
CROSS_PLATFORM_REPORTER
DEFAULT_NUM_CPUS
DEFAULT_TESTSUITE
DEFAULT_TIME_LIMIT
MPIRUN
PARALLEL_EXECUTABLES
PROJECT
QUEUES
REPORTS
SERIAL_EXECUTABLES
SYSTEM_NAME
TESTSUITE_DIRECTORIES
TEST_DIRECTORIES
VERBOSE

4.2 *.suite Files

This is a list of test cases and/or additional suites that will be run as part of a given test suite. An example suite file name could be regression.suite. See the examples directory in the source checkout for example files. Edit the file template.suite and save a copy for each suite. The suite file(s) must be specified in the *.cts file.

```
# This is an example test suite file
include simple.suite                    # This line included another test suite.
not_so_simple.suite                     # So does this line.
+case1                  8     2:00      # This line includes a test called case1 to be run on 8 processors with a time-limit of 2 hours.
case2 ..................................# adds a test call case2
- case4                                 # This line excludes case4.
EXCLUDE case5                           # We are not case sensitive.
If (case4 && (-x real_fast_code)){      # Why run the test suite if the executable failed to build?
   not_so_simple.suite
}
Else { simple.suite}                    # We may eventually allow embedded if blocks, but not in version 1.
```

4.3 *.test Files

Each test case must have an associated *.test file. This file can contain the number of processors, time limits for failure criteria, comparator commands, additional post-processing commands, etc. You may choose to use the test case name as the prefix – e.g. Sedov.test. The prefix may be anything (e.g. case name), or the file may be simply named ".test". See the examples directory in the source checkout for example files. This file is automatically referenced at runtime.

```
# This is an example.test file
DATA   mydate.file
SERIAL_CODE inf/slow_input_file                 # This must be a legacy code.
READ_FAST_CODE  --run_fast   fast_input_file
COMPARE_OUTPUT
```

4.4 TestSuite Directory

This directory is the source of the test cases. It contains input files for the application code that CTS will be running, and additional application code input files such as meshes and standards files. The TestSuite directory may contain subdirectories that contain individual test cases, but those directories must be named identically to the test case input files they contain, minus the suffix. For example, if sedov.input is an input file for an application code, it should reside in the directory TestSuite/sedov.

This directory gets copied into what becomes the working directory for a CTS execution. The "testing" directory is created by CTS during each initial execution. It contains a replication of the test case directories where the individual test problems get run. Thus, it is also the location where output from the executable may go as well as some CTS-specific output such as results.

4.5 Executable

This is the executable that is used to run the test problems. It may be specified in the *.cts file or via the command interface to CTS, which may be command-line or, in later releases, a GUI.

4.6 Link Files

Some application codes require additional files in order to run. CTS calls those “link files” and must include them as input in order to run. These may be specified or indirectly referenced in *.cts file or via the command interface.

5 RUNNING CTS

5.1 Interactive Usage

To run interactive via the command line, change directory to the cts install dir. This will be the directory that contains bin, lib, man, and html. Enter the command:

./bin/cts --cts myctsfile.cts

STDOUT will display current job status. Jobs will run in batches of size as configured in the *.cts file. Reports and their output location will be written as specified in the *.cts file.

5.2 Automated Usage

Create cron job to launch CTS. Example cron_cts:

```
15 06 * * * perl /usr/projects/project1/cts/bin/cts --cts nightly.cts
```

In this example, the cts script will be executed in the cron environment every morning at 6:15 am. It may be necessary to modify your cron environment to run correctly. Next, load the cron job onto machine with the command:

```
crontab cron_cts
```

When the cron job has run, check for output in the directories you've specified in the *.cts file, or exercise cron by changing execution time to (e.g. current time + 2 minutes) near future and loading cron job. The cron job may require some environment tuning to achieve desired results such as sourcing a "project_cts" .cshrc file, loading specific modules or setting PATH environment variables etc.

6 Output File Descriptions

6.1 Runscript

This is generated at run time for each test case and may be used directly as a diagnostic or run script resource.

6.2 Tmpfile

Shell script that runs the problem - generated at run time for each test case and may be used directly as a diagnostic or run script resource.

6.3 Copy of Test Directory

The test case problem subdirectory is copied into "testing" where is may be operated on by CTS without changing the original problem directory's contents.

6.4 *.db File

This file contains all results data for a given problem execution. It is subsequently used by the reporter module. If the "storable" perl module is available, the file format is binary. Otherwise, it is written in ASCII text by the dumper module.

6.5 Outfile

This is STDOUT from the test execution. It is linked to in the html report.

6.6 bsub_<problemname>

This is the bsub output, also linked to in the html report.

6.7 Report Files

Results.txt is the default text report file. cts_results.html is a single html document that contains all test results for a given execution of CTS. That is, if multiple suites are run, the results from all suites will be included in cts_results.html. Results from the individual suites are written to <suite name>_results.html.

7 Adding Test Cases

Make test directory

mkdir my_new_test

Create test input file

cp my_new_test.in my_new_test/.

Create my_new_test.test file to run problem

my_prog my_new_test.in
ctsdiff -r .001 -or -a .001 – gold_my_new_test.out my_new_test.out

Run problem and generate gold standard file

Add test case to my.suite file

noh
my_new_test

8 Drilling Down on Failing Tests

Make test directory

mkdir my_new_test

Create test input file

cp my_new_test.in my_new_test/.

Create my_new_test.test file to run problem

my_prog my_new_test.in
ctsdiff -r .001 -or -a .001 – gold_my_new_test.out my_new_test.out

Run problem and generate gold standard file

Add test case to my.suite file

noh
my_new_test