How to Measure Up: Contributing to the CWBP

DOUGLAS EADLINE

Many a cluster conversation takes the following form: "I upgraded to the new motherboard and now my cluster blows the doors off everyone else's in my building." Being a scientist, I have attempted to quantify the phrase "blows the doors off" for quite some time. I'm sure those living in Florida this year may have a whole different take on this phrase, but I believe it came from automobile racing where one car would be going so fast that they would create a enough draft to "suck" the doors from the other car. Not being a fluid dynamics expert, I'm not sure if such an effect is even possible and thus even possible to quantify. In any case, I also must allow for the fact that this cluster user may have actually modeled wind tearing doors off of his facility, but my experience tells me it is yet another worthless performance comparison.

When I have encountered these types of comments I always ask, politely, if there are any quantitative numbers to go along with the "door scale." Usually the answer is no. And I understand why. Running benchmarks is difficult. Not only must you be careful with your assumptions, but getting things to actually work can be a real pain.

The ClusterWorld Benchmarking Project is intended to help make better quantitative comparisons. We have discussed the project before (see the *The Right Stuff* columns in the May and August 2004 *ClusterWorld Magazine* issues), and have now put a specification in place so we can move forward with the actual implementation. As mentioned, benchmarks will be assigned to various levels of test. A description of these levels is given in the *CWBP Benchmark Levels* sidebar (*page 32*).

Mention the Intention

An important point to remember is that the CWBP is not planning to write the actual benchmarks. There are plenty of benchmarks that are freely available (both codes intended as benchmarks and real applications). Our job is to collect them and make them easy to use on clusters.

There are three types of comparisons for which the project is designed:

1. Comparing your cluster with your cluster. Although it sounds silly, this is probably the most important use of the benchmarks. For example, the software environment for clusters is quite dynamic, and the assumption is that new versions of software are faster than old versions. Without some way to test the old against the new, there is no way to know if updates and changes are moving you in the right direction for *your cluster.*

2. Helping to design your cluster. As we know, the range of choices for cluster components is quite large. Choosing the right components requires more than a data sheet and some good words from a sales person. Indeed, the nature of clusters is that not until the whole system is tested can you be sure that everything is working correctly. Having a set of tests to evaluate and help steer the design, procurement and acceptance process is essential to optimizing your purchase. Otherwise, you end up guessing.

3. Cluster-to-cluster comparison for the same code and data set. These are perhaps the trickiest types of comparison, but a user should be able to test the same application code on a different cluster and expect the results to useful.

Astute readers will notice that I have put cluster-to-cluster comparisons last on the list. I have done this to de-emphasize the competitive nature of these benchmarks. Ultimately, some will report that they "blew the doors off" of a benchmark. At least we will be able to ask for the numbers.

As an aside, in recent times, I have been accused of taking on the role of *cluster mystic* when it comes to performance questions. If asked by any young cluster wonk, "How fast is your cluster?" I reply, "When you can tell me how tall is a building, I will tell you how fast is my cluster." That usually gets rid of them and spares me from providing a long polite explanation as to why their question is *non sequitur* — i.e., stupid.

Now that we have declared our intention, let's talk

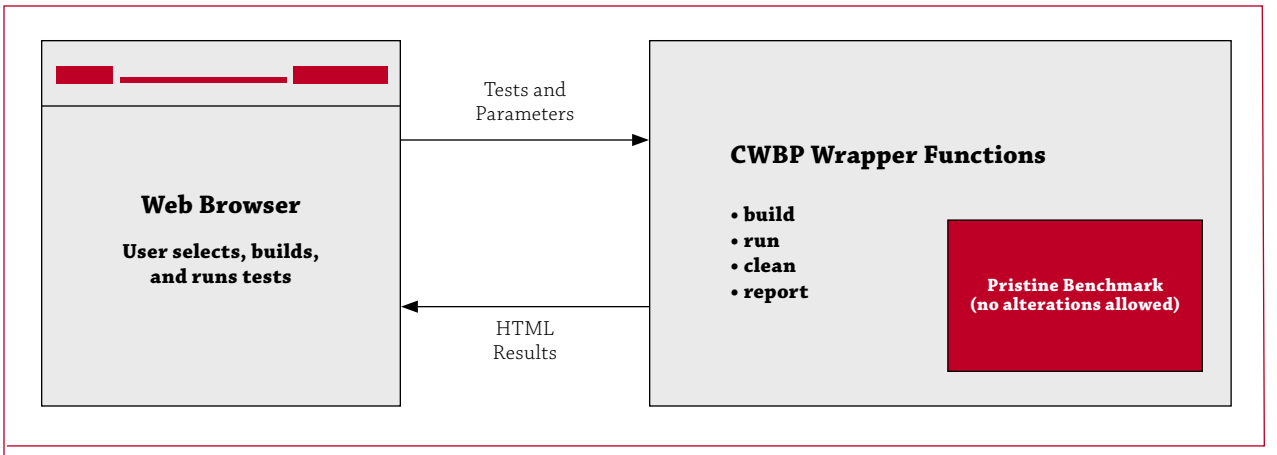

FIGURE ONE: Architecture Overview

about how we plan on making all this work. And, more importantly, how you can get involved.

Laziness for Dummies

Having lived a fair portion of my professional life in front of CRTs, I can say that some of the best programming practices involve what I like to call the lazy approach. That is, if it is already done, try not to re-write it, but rather use it untouched. You can always optimize later if needed. Of course, this approach often makes the more eloquent coders among us cringe, but this is my project so we will do it my way until someone more experienced and eloquent than I comes along and takes over (hint, hint).

In general, my plan is as follows: to build a framework for which existing benchmarks can be easily added — without having to recode or modify the benchmark application itself. The idea is to have firm lines between what we do and what the benchmark does. Of course, if the application authors may want to build in hooks for the CWBP framework, such a situation would be ideal. However, I believe a modular, but somewhat clunky, design may work best in the end (at least it lets us get started quickly).

The only requirements for the benchmark or applications used in the framework are that they be open source and meaningful. These requirements will ensure that native compilations will be done for the code on the test systems and that users will have full responsibility for compilers and compiler options, if allowed. In addition, the whole process will be open. Not that anyone ever cheated on a benchmark.

A boundary of sorts will then exist between the actual tests and the framework that uses them. This feat will be accomplished with some nice low-tech wrapper functions. The specification presented here will allows the CWBP framework to call any benchmark that has been "suitably wrapped."

From the user's standpoint, the benchmarks will run from a web browser. In addition, results will be cast in HTML so that they can be easily shared and viewed by everyone. An overview of the design is shown in *Figure One*.

Let's Wrap

From the Framework standpoint, each benchmark will need to provide some standard wrapping functions (many of which may already be available from the benchmark itself) that will assist in making, running, cleaning, and reporting results. The directory specification is shown in the *Directory Specifications* sidebar (*page 34*). The following wrapping functions must be provided for each benchmark.

1. **A makefile or script.** These wrappers are responsible for building the binaries, this process may call a `configure` option as well. Once binaries are built they are copied into the `/run` directory.

2. **A "make run" makefile option or script.** These wrappers are responsible for running the binaries. They will copy any data that is needed from the `/data` directory to the `/run` directory. Any temporary or output files will be created in the `</run` directory.

3. **A "make report" makefile option or script.** These wrappers are responsible for producing an HTML report (see below) for the run and placing it in the `/report` directory. The report will be viewable from a browser. The report will include the run parameters as well as the test results.

4. **A "make clean" and "make veryclean" makefile option or script.** This script will provide a "make clean" option that will remove everything

in the `/run` directory and "clean" the benchmark code directory. A "make veryclean" wrapper will also delete all reports, and pretty much try to put things back to the way they were before any tests were run.

5 **An Input Parameter File.** This file will contain all the information need to compile, run and verify the test. See below for more information.

In general, the "real work" will probably be creating the HTML output from the test results as many applications and benchmarks have automatic build scripts or makefiles. The build wrappers will ensure that the framework can interact with the benchmarks in a consistent fashion. Example wrappers are providedat *cwbp.sourceforge.net*.

Also, note that the wrapper functions can be written in any language. The user who has taken the time to build the wrappers will have the prerogative to use what very language works best — provided it is part of a standard Linux distribution.

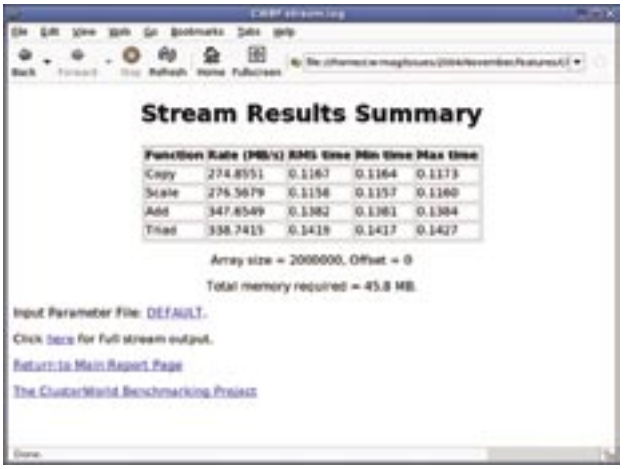

FIGURE TWO: Example CWBP Report Page

CWBP Benchmark Levels

Micro-Benchmarks
These types of benchmarks require one or two nodes and look at very low-level performance features which may include memory bandwidth, multiprocessor scaling, network speeds, storage speeds, and other single-system tests.

Whole System Benchmarks
At a slightly higher level, a set of systemwide general benchmarks and tests will provide an sense of how the cluster is functioning as a whole system. Examples of these types of benchmarks include NAS Parallel Benchmarks, the HPL benchmark, and some of the MPI suites that are available.

Application Benchmarks
At this level, real applications will be tested. These will include representative applications from areas of bioinformatics, weather forecasting, computational chemistry, rendering, and others. As stated, these will, in part, be based on freely available applications.

Workflow Benchmarks
The topmost level will test how well a specific workflow is handled by the system. This level obviously brings in the batch scheduling capabilities of the cluster. It requires a set of applications from level III that may look similar to the users typical workflow. These applications would be queued on the cluster and then run to determine how much work gets done in a specific amount of time.

Input Parameter File

In order to manage data for each benchmark, an Input Parameter File (IPF) will be used. The file will work as follows. Each submitted test will have a default parameter file created by the benchmark submitter (benchmark wrapper). The web interface to the CWBP will parse this file and if allowed provide a field for the user to change some of the parameters. For instance, compile option flags could be one parameter under user control. The choice of what the user controls will be up to the person who prepared the benchmark wrapper. You will always have the option to go back to the default settings. It will also be the preparer's responsibility to make sure that user parameters are correctly used for the benchmark run.

The input specification file may vary from benchmark to benchmark, but it will at a minimum have the following fields:

Benchmark Name:
Version Number:
Source of Benchmark:
Benchmark Filename:

Compiler:
Compiler Options:
Requirements:

Binary Name:
Command Line Options:
Output Data File:
Verification File:

The benchmark name, version, source, and file name are all needed to track the benchmark. The compiler may be a user selectable list of compilers known to work with the benchmark or a single compiler. The same goes for the compiler options. The requirements field is a bit trickier. If for example, the benchmark submitter requires that certain software be present or that the benchmark has only been tested on a specific OS version or cluster version (i.e. Scyld, Rocks, etc.), then the framework will check for these requirements. If the requirements are not satisfied, the benchmark will not run. The user will be informed of the situation.

Directory Specifications

The CWBP framework will need a predicable way to build and run tests. Those tests and requisite wrappers need to be placed in the following directory structure by either a tar file or an rpm. The following specification applies to all tests.

1 Installation requires a tar file or rpm that extracts into:

```
/opt/cwbp/level/name
```

where `level` represents the test level (see the *CWBP Benchmark Levels* sidebar) and `name` identifies the name of the test. The `level` and `name` values will used by the framework to identify the benchmark.

2 The benchmark `name` directory is constructed as follows:

`/build` - includes wrapper functions
`/data` - includes any data files
`/run` - a directory where the executable are run
`/report` - a directory to hold the results
`/benchmark` - directory that holds the benchmarks as extracted from original benchmark source. i.e. the pristine benchmark code.

As mentioned, the `/build` will have scripts or makefiles that will allow building, cleaning, running, and reporting results. The build scripts will put binaries in the `/run` directory. The run scripts will copy any data from the `/data` directory to the `/run` directory and run the code. The report script will create an HTML report and place it in the `/report` directory. A clean script will clean the `/run` directory and `/benchmark` directories.

The binary name and command line arguments are used to run the program. Note that any input files for the program will be moved to the `/run` directory automatically.

Finally, the output file is the name of the raw output that will be used for the HTML generation and included in the report. In this way, all data for the benchmark, summary HTML report, and detailed results will be immediately available.

The verification field is used if the test can be verified against a known data set. Obviously, some benchmarks that measure certain system parameters (like memory bandwidth) cannot be verified. In this case, the field will be left blank.

The Report

An example report page is shown in *Figure Two* (*page 32*). The final version will probably be a little bit different, but the available data will essentially be the same.

Of particular note is the Input Parameter File link. If the benchmark defaults where used, then a link will show "DEFAULT" and reference the defaults file. If the user had changed any of the defaults, then the link will show "USER" and reference the user modified Input Parameter File. There is also a link to the raw data as well.

Invitation

By the time you read this, version .1 of the framework and some example tests will be available from *cwbp.sourceforge.net*. Updated project news will also be up on the website.

It is fully expected that the initial design presented here will undergo changes as the project matures. Therefore, a more detailed specification will be placed on the website so that the community can post comments and make suggestions. At this point, the design may need to be enhanced for the level four tests.

New benchmarks can be submitted, but will only be added after testing by qualified CWBP members. You are encouraged to become a qualified CWBP member. We have no T-shirts, but we do have the noble goal of total cluster benchmark domination.

Which brings us to the request for help. *ClusterWorld Magazine* will help with the project as much as possible, It will not, however, move very fast if we are the sole contributors. My intention is to solicit help from the community at large and in particular from those members who have expertise in many of the important application areas. In this way, we can add tests quickly and "blow the doors off" all the other benchmarking efforts out there.

Douglas Eadline is the editor-in-chief of ClusterWorld Magazine. *He can be reached at deadline@clusterworld.com.*