HDF5 Performance Framework Back-End System

Hyo-Kyung Lee

This is supplemental documentation to the HDF5 Performance Framework(HPF) user manual covering the HPF back-end system in depth. It is an essential guide for the HPF developers and advanced users who are in charge of making HPF system up and running.

1 Introduction to HPF Back-End System

The HPF back-end system is primarily responsible for scheduling and executing benchmark programs. In addition, it is responsible for detecting any system environment change on a target platform as well as notifying benchmark results to HPF users via daily e-mail if there's any performance degradation or improvement.

UNIX shell scripts and PHP scripts are used to implement the HPF back-end system. Thus, PHP interpreter is not an option but must be installed in advance. The standard UNIX cron daemon will execute the HPF back-end scripts once they are submitted as a cron job.

2 Anatomy of Shell Script

There are three shell scripts for HDF5 version 1.6, 1.8 and 1.9. They are all essentially the same in structure and function but they are separated for the scheduling and customization purposes.

2.1 Customization

In terms of customization perspective, each shell script has 2 parts. The head part is for customization and the body part requires a minimal customization. The shell script detects system environment changes, builds the HDF5 library and the benchmark programs, tests them, stores their results to database and reports results to users.

The head part starts with the following comment block:

```
##################################################
# Please edit the following parameters before you submit this script into cron.
##################################################
```

The customization begins with the optional path setting. All UNIX commands (e.g. **svn**,

php, **(g)cc**, **(g)make**) that are necessary to build the HDF5 library and the HPF benchmark progrmas must be accessible through the PATH environment variable.

```
# PATH=/usr/local/bin:/usr/ucb/bin
# export PATH
```

Set the HDF5 library version --- it'll be 1.6, 1.8 or 1.9.

```
VERSION="1.6"
```

Set the directory for temporary files. Please do not set it under HDF5_PREFIX(see below). It should never be deleted by this script.

```
TEMP="/tmp/chicago_$VERSION"
```

Set cc/c++ or gcc/g++ version command. Solaris may require "cc -V" instead of "gcc -v".

```
CCV="gcc -v"
CPPV="g++ -v"
```

Set the HDF5 Installation Directory. This directory will be cleaned everyday.

```
HDF5_PREFIX="/home/local/hyoklee/chicago/hdf5-$VERSION"
```

Set the configuration option for building the HDF5 library.

```
HDF5_OPTION="--disable-shared --enable-cxx --enable-production --
prefix=$HDF5_PREFIX"
```

Set the configuration option for building the HPF.

```
PERF_OPTION="--disable-shared --prefix=/home/local/hyoklee/chicago --with-
hdf5=$HDF5_PREFIX --with-mysqlclient=/hdfdap/mysql"
```

Set the path to the HPF source.

```
PERF_SRC="/home/local/hyoklee/src/chicago/trunk/hdf5perflib/"
```

Set the path to php command. In the following example, it's simply set as **"php"** since it can be found from the default PATH environment.

```
# e.g. PHP=/opt/csw/bin/php
PHP="php"
```

Set the path to performance framework PHP source. This directory must have all PHP scripts used in the HPF.

```
PHP_SRC="/home/local/hyoklee/src/chicago/trunk/hdf5perfphp/hdf/"
```

Set the path to the **make** command.

```
# e.g. MAKE=/usr/ccs/bin/make
MAKE="gmake"
```

The back-end script will run the same test 3 times and record the best performance because test results may be affected by the system load. Set the interval in seconds between trials.

```
# 600 seconds = 10 minutes
INTERVAL="600"
```

Set the correct usage option of the **diff** command. Solaris doesn't require "-u" option in **diff**. This command is used for detecting system environment changes like OS version and gcc version.

```
DIFF="diff -u"
```

Finally, set the correct path for subversion repository. This must change according to the HDF5 library version the back-end script is going to build and test.

```
SVN_URL="http://svn.hdfgroup.uiuc.edu/hdf5/branches/hdf5_1_8"
```

The body part starts with the following comment block:

```
##############################################################
# Please DO NOT edit lines below.
##############################################################
```

No more customization is necessary after the above comment block.

2.2 Functionality

In terms of functionality, each shell script has 3 parts.

The first part contains the environment change detection. This function is achieved mainly through a cycle of executing a certain shell command, redirecting the result into a temporary text file and checking difference using the standard UNIX **diff** command. Let's go over the environment change detection cycle with excerpt.

Existing text outputs are saved first for the upcoming comparison:

```
if [ -e $TEMP/compiler_options_perf.txt ]; then
  mv $TEMP/compiler_options_perf.txt $TEMP/compiler_options_perf.old.txt
fi
```

Here's the part of getting environment data by executing commands and redirecting the output:

```
# Get today's environment data.
uname -a >& $TEMP/uname.txt
$CCV >& $TEMP/cc_version.txt
$CPPV >& $TEMP/cpp_version.txt
echo $HDF5_OPTION > $TEMP/config_hdf5.txt
echo $PERF_OPTION > $TEMP/config_perf.txt
```

Finally, **diff** command is run in order to count how many changes are detected.

If there is no change in all 5 system environment categories, nothing will be recorded.

```
OUTPUT=$TEMP/diff.out
y=0

# Check today's environments against yesterday's environments.
if [ -e $TEMP/uname.old.txt ]; then
diff -u $TEMP/uname.txt $TEMP/uname.old.txt > $OUTPUT
if [ -s $OUTPUT ] ; then
y=`expr $y + 1`
fi
else
y=`expr $y + 1`
fi
```

The second part has a loop that iterates the same benchmark 3 times to ensure that test results are not susceptible to the system's load average.

```
x=0
while [ $x -lt 3 ]
do
$MAKE check
x=`expr $x + 1`
sleep $INTERVAL
done
$MAKE distclean
```

The last part records the test results (**TestInstance.php**) and determines the winner from the 3 trials in the previous part (**checker_best.php**).

```
$PHP $PHP_SRC/TestInstance.php $PERF_SRC/results.xml
$PHP $PHP_SRC/checker_best.php $VERSION >& /dev/null
```

3 Scheduling Cron Job

In general, a back-end script consumes a lot of system resources since it rebuilds the HDF5 library and most HDF5 benchmark programs involve heavy file I/O operations. Thus, scheduling a set of cron jobs for three back-end scripts requires a careful consideration. As a rule of thumb, there are three things that you should keep in mind.

First, it is best to allocate it at the time when the system is not busy. Although the back-end script is designed to run the benchmark programs three times with interval, you cannot guarantee the best performance if they are all executed when a system load is very high. The ideal case is to have a dedicated machine that runs the back-end script only and nothing else from other users.

Second, it is important to avoid overlap among back-end scripts of different HDF5 versions: 1.9, 1.8 and 1.6. It is good to have an enough system cool-down period between two scripts to get the most accurate measurement of benchmark programs. For example, if one script takes about 1 hour 30 minutes, it's good to schedule three cron jobs at 6:00pm, 8:00pm and 10:00pm for each HDF5 version.

Finally, all back-end scripts should finish before the end of a day(i.e. 11:59pm) since timestamps are critical in calculating the best record from DB and generating a performance warning report.

The cron jobs for **sync.php** and **mailer.php** need to be placed only one machine that has a full access to MySQL DB since they should run only once per day. They aggregate and fix ids for the results stored in DB and send an e-mail if there's any performance issue. It is important to note that, unlike other PHP scripts, they look for the time stamp of previous day in DB to ensure that all performance results are recorded and ready for a warning analysis. The installation of cron job for **mailer.php** is optional since there's an equivalent HPF front-end script called **report.php** which is accessible from the web server.

Here's a real example of scheduling used on *hdfdap* machine.

```
00 18 * * * /usr/local/home/hyoklee/src/chicago/trunk/hdf5perfphp/run_1.6.hdfdap.sh
00 20 * * * /usr/local/home/hyoklee/src/chicago/trunk/hdf5perfphp/run_1.8.hdfdap.sh
00 22 * * * /usr/local/home/hyoklee/src/chicago/trunk/hdf5perfphp/run_1.9.hdfdap.sh
15 01 * * * /usr/bin/php /usr/local/home/hyoklee/src/chicago/trunk/hdf5perfphp/hdf/
sync.php
55 01 * * * /usr/bin/php /usr/local/home/hyoklee/src/chicago/trunk/hdf5perfphp/hdf/
mailer.php
```

All shell scripts are expected to finish before 11:59pm and **mailer.php** will generate a warning report on the next day based on the results of previous day. The **sync.php** must be called before the **mailer.php**.

4 PHP Scripts

There are many PHP scripts that are invoked inside the HPF back-end system and it is beneficial to understand their roles. In general, anything DB related function of back-end script is written in PHP for fast and portable implementation.

4.1 svn.php

The back-end shell script calls this PHP file first to record the subversion revision number of the HDF5 library it checked out.

```
svn co $SVN_URL svn | grep "Checked out revision" | cut -f4 -d ' ' | cut -f1 -d '.' >  $TEMP/
svn.log
```

```
$PHP  $PHP_SRC/svn.php $VERSION `cat $TEMP/svn.log` >& /dev/null
rm -rf $TEMP/svn.log
```

4.2 TestInstance.php

This is a new PHP file that was added in June 2009. This file is called inside the back-end shell script to solve the problem of the MySQL C client library.

In older HDF Performance Framework, test instances used to call the MySQL C client library directly. However, calling MySQL client APIs directly by benchmark programs often failed to write the results into MySQL server DB correctly for many reasons. It also created duplicate test routines and action names with different ids which gave ended up adding the **sync.php** script.

To solve the above chronic problem, we decided to avoid the use of MySQL client library directly by HPF C/C++ APIs. Instead, all benchmark programs will create a local XML file that will be transferred into the MySQL database later by a cron job. In this way, even if there's an error on MySQL server DB, we can have a local copy of test results and it can be delivered into DB later via PHP cron job when the MySQL is back.

Another benefit is that we don't have to worry about updating the APIs related MySQL client library inside HPF for future maintenance since all DB transactions will be handled by PHP-MySQL module. This will ensure the maximum portability of the HPF system.

This script reads an XML file and parses it. It is important to set the path and file name of the XML file correctly inside the benchmark programs.

```
$PHP $PHP_SRC/TestInstance.php $PERF_SRC/results.xml
```

This script needs some improvement since it cannot handle TestRoutine and TestAction tag inside the XML file. It should check the existing routine/action names and create new ones if necessary.

4.3 checker_best.php

If you take a look at the back-end script near line 130, there's a code that calls the **checker_best.php**:

```
$PHP $PHP_SRC/checker_best.php $VERSION >& /dev/null
```

This **checker_best.php** script finds the best value from the table **TestInstance** and duplicate the best record into a separate table called **TestInstanceBest**.
If there's a tie in any record, it simply copies one that has the lowest **TestInstance_ID**.
Since a back-end script runs the same benchmark program 3 times, there should be 3 times more entries in the **TestInstance** table than in the **TestInstanceBest** table.

The real computation for determining the best record is done within a function called *get_best_record()* in the **analyzer.php**. The **checker_best.php** simply includes the **analyzer.php** at the beginning.

4.4 environment.php

If you take a look at the back-end script near line 208, there's a code that calls the **environment.php**:

```
$PHP $PHP_SRC/environment.php $VERSION $TEMP/uname.txt $TEMP/cc_version.txt
$TEMP/cpp_version.txt $TEMP/config_hdf5.txt $TEMP/config_perf.txt $TEMP/
compiler_options_hdf5.txt $TEMP/compiler_options_perf.txt >& /dev/null
```

The **environment.php** records any system environment changes into a MySQL DB table called environment. Such changes include C/C++ compiler version, HDF5/HPF configuration option and HDF5/HPF compiler option.

4.5 sync.php

As noted in 4.2, the use of MySQL client library often generate duplicate TestRoutine and TestAction entries in HPF DB. The **sync.php** will clean up any duplicates from the routine and action tables and ensure that the front-end system can run smoothly. This is a stand-alone script that should be called only once, not by the back-end script shell.

4.6 mailer.php

This PHP script sends a warning message in a tabular form to subscribers who are interested in HDF5 performance. For each routine, action, machine, HDF5 version and instance, the script checks if there is any 20% or more increase(performance loss) or decrease(performance gain) in a new record vs moving averages of previous records. Currently, yesterday's record is compared against the average of last 7 days and last 7 days of average is compared against the last 30 days of average excluding the current one. That is, a day 10 record is compared against day 3 to 9 and a day 1 to 7 record is compared against the day 1-30 record of previous month.

Such comparison is actually done in **analyzer.php** and it is included in the **mailer.php** at the top. Thus, if you want to change the parameters for comparison, you need to modify **analyzer.php**.

The **mailer.php** retrieves subscriber list from subscribe table. Retrieval is actually done in a function called *get_subscribers()* in **util.php**. The current implementation retrieves all subscribers in a single string by appending one by one and send a bulk mail by putting the long string in *To:* line of e-mail. This method can be improved by sending one e-mail per subscriber in the future.

Again, like **sync.php**, this is a stand-alone script that should be called only once, not by the back-end script shell.