HDF5 Performance Framework Front-End System

Steve Konstanty and Hyo-Kyung Lee

1 Introduction

This document provides in-depth information about the HDF5 Performance Framework(HPF) front-end system and instructions on how to use it. The HPF front-end is a web based utility that allows a user to plot out line graphs and to view journal entries and warning report from the data stored in the HPF database tables. This utility uses PHP scripts along with the *jpgraph* library to display data.

2 HPF Front-End System

In this section, we'll go over the system requirements and how HPF front-end system works internally.

2.1 Requirements

The HPF front-end system was developed using the following technologies: XHTML, CSS, JavaScript, and PHP. It requires a PHP-enabled web server like Apache 2.x and MySQL Server 5.x. This application requires having the PHP GD graphics library installed and enabled, PHP session support enabled, and for non-Windows operating systems, all *cleartype* fonts used by the *jpgraph* library need to be downloaded and installed. More information on the installation of fonts and the complete *jpgraph* library can be found at http://www.aditus.nu/jpgraph/.

2.2 How It Works

In general, HPF front-end system has three components - obtaining input parameters from users, retrieving data from storage system and displaying data on the screen. The HPF front-end system gets user parameters from HTML form, retrieves data from MySQL DB and passes them to *jpgraph* or HTML table. Since getting parameters from user via HTML forms and displaying data in a HTML table are trivial, we will emphasize on retrieving data from MySQL and plotting data through *jpgraph*. Here, we'll use Step-by-Step Interface example to illustrate how three components work together.

2.2.1 Obtaining Parameters from User

In the *Step-by-Step interface* that starts from *0.php* and ends at *6.php* page, we use a simple HTML form that makes use of **<select>** element and calls a third party JavaScript pop-up menu for dates. We added this calendar function not only for aesthetics and ease of use, but also for forcing users to enter the correct date format in a proper order: yyyy/mm/dd. The PHP scripts in *Step-by-Step interface* method are named after the step sequence as *0.php*, *1.php* ... and *6.php* to indicate the order of processing.

Every form contains **<select>** elements that are generated dynamically from one of the tables in the MySQL database. Each form submits to the next page specified in **<form action='*x*.php'>** tag. A very simple validation is done via PHP to let the user know that all fields are required. Without the required information the script will not proceed to the next step. We've added the ability for the forms to be "sticky" so that the user doesn't have to fill in the same input again.

In *5.php* form, *cal_conf2.js* and *cal2.js* are the required JavaScript files to allow the calendar to work properly. The only file that should be modified is *cal2.js*. It contains the settings for font-size, dimension, and color of calendar near the top part of the script. We've also added a validation mechanism for calendar input --- the starting date cannot be a date that occurs after the ending date.

Each step is modeled according to the hierarchy of HPF data model. For example, a user can select routine first, action second and then instance. User's input in each step is submitted to the next step via POST method with hidden <**input**> tags. Thus, all previous inputs are carried over to the next step in order to create a new MySQL DB query to retrieve the filtered data and then a new form is generated based on the query result. In the final step, all user's inputs are reflected in a single query and that query is used to retrieve data.

2.2.2 Retrieving Data with Parameters from User

Upon successful entry of all required information in *0.php* to *5.php*, those variables are used in building a MySQL query in *6.php* which is the last step of *Step-by-Step interface*. To perform any MySQL query for data retrieval, it is crucial to include the following two statements:

```
include ("includes/data.inc.php");
include ("includes/connect.inc.php");
```

If no records are found that meet a query's criteria then a message is returned that says "**No records were found**".

If a set of records is found then it is processed in the *6.php* script for graph generation. If only one plot data point is found, then an error message that says "**At least 2 plot points are required to generate a plot**" will appear.

The main query below retrieves data from the table *TestInstanceBest* and they are used as our plot data --- x and y axis data points.

```
  $query = "SELECT ti.TestResult_Value, DATE_FORMAT(ti.TestInstance_Date,'%m/%d')
FROM TestInstanceBest ti WHERE Host='$host' AND DatasetName='$instance' AND
LibraryVersion='$version' AND ti.TestInstance_Date BETWEEN '$start_date' AND '$end_date'
ORDER BY TestInstance_Date";
```

The rows returned by the above query is used to create two separate arrays. One for the x axis and one for the y axis. The x axis array consists solely of dates that meet our query's criteria. The y axis array has benchmark result values.

```
$result = mysql_query($query);
$num = mysql_num_rows($result);

if ($num){
  while ($row = mysql_fetch_array($result, MYSQL_NUM)){
    $datax[] = $row[1];
    $datay[] = $row[0];
  }
}
```

2.2.3 Plotting Graphs from Retrieved Data

The *6.php* requires the *jpgraph* libraries and they should be placed at the top of that page.

```
    include ("jpgraph/src/jpgraph.php");
    include ("jpgraph/src/jpgraph_line.php");
```

The *$datax* and *$datay* in the 2.2.2 are the two data variables that are passed to the *jpgraph* plotting functions.

Here's is an example code that generates a simple plot with *jpgraph*.

```
  // Create a graph.
  $graph = new Graph(480,240);
  $graph->SetScale("textlin");
  $graph->yaxis->scale->SetAutoMin(0);

  // Create the linear plot
  $lineplot=new LinePlot($datay);

  // Add the plot to the graph
  $graph->Add($lineplot);
```

```
$graph->img->SetMargin(60,60,20,70);
$graph->title->Set($instance." on ".$host);
$graph->xaxis->title->Set("Dates");
// Auto tick doesn't work well.
$graph->xaxis->SetTickLabels($datax);

if(count($datax) > 7){
  $graph->xaxis->SetTextTickInterval(7);
}

$graph->yaxis->title->Set("Time in seconds");
$graph->yaxis->title->SetColor('red');

$graph->title->SetFont(FF_FONT1,FS_BOLD);
$graph->yaxis->title->SetFont(FF_FONT1,FS_BOLD);
$graph->xaxis->SetFont(FF_ARIAL,FS_NORMAL,9);
$graph->yaxis->SetFont(FF_ARIAL,FS_NORMAL,9);
$graph->yaxis->SetTitleMargin(35);

$graph->xaxis->SetTitleMargin(35);

$graph->xaxis->SetLabelAngle(45);

$lineplot->SetColor("blue");
$lineplot->SetWeight(2);

$graph->yaxis->SetColor("red");
$graph->SetShadow();
$graph->footer->center->Set($version);
$graph->Stroke();
```

The numbers and strings in the above example can be customized to generate a slick graph. More information on *jpgraph* can be found on the following support page: http://www.aditus.nu/jpgraph/.

3 HPF front-end PHP sources

This section explains each PHP script used in the HPF front-end system. The order of subsection follows the link items presented in the HPF home page:

http://hdfdap.hdfgroup.uiuc.edu/h5perf/index.html

3.1 See today's warning report: report.php

Although a daily warning e-mail can be sent by back-end script, the same report can be seen by clicking **report.php** on the web. It has the exactly same functionality that **mailer.php** provides except that it doesn't send any e-mail. Instead, it shows the warning report in a HTML table.

For each routine, action, machine, HDF5 version and instance, this script checks if there is any 20% or more increase 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 week-long 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 **report.php** at the top. Thus, if you want to change the parameters for comparison, you need to modify **analyzer.php**.

3.2 See today's graph for version: today.php, today2.php, today3.php

The **today.php** and **today3.php** generates 11 graphs of all 11 test instances that include all test results from 4 hosts for HDF5 library version 1.9.x and 1.8.x. The **today2.php** generates 8 graphs of all 8 test instances that include all test results from 4 hosts for HDF5 library version 1.6.x.

The above three PHP files are identical except the version number of HDF5 in MySQL query and the legend at the bottom of the graph.

This script checks the today's date first and retrieves data from the last 7 days and plot them using *jpgraph*. It is a variation of *6.php* discussed in the previous subsection 2.2.3 and it has capability of cascading several graphs into one.

It is useful to compare the performance across different platforms and to check quickly if there's any missing record.

3.3 Step-by-Step Interface: 0.php, 1.php, ..., 6.php

The 0.php, 1.php ... 6.php scripts are responsible for generating a custom graph and they are named after the sequence of steps. Each step passes the parameters from the previous step through **<input type=hidden>** tag to the next step. A detailed explanation is already covered in section 2 of this document.

3.4 Version Interface: version.php

The **version.php** script is a great way to compare HDF5 versions on specific test instance. Currently, this script accepts an input from last *x* days amount of data and a host name to plot graphs for all 5 test instances. Each instance graph has 3 different lines which indicate three different HDF5 library versions.

3.5 Subscribe / Unsubscribe Alert via e-mail: subscribe.php and unsubscribe.php

These two simple scripts are responsible for writing and deleting a (name, e-mail) record using the **subscribe** table in DB.

3.6 Add / View journal entry: journal.php, journal1.php, journal2.php and journal_v.php

These scripts are responsible for writing and viewing a journal entry using the **journal** table in DB.

When a user enters a journal entry using **journal.php**, **journal1.php** will ask for a confirmation for the entry and store it into DB using **journal2.php**. Each journal entry may contain some characters that MySQL DB don't like so they are escaped with *urlencode()* method.

The **journal_v.php** lists journal entries in a single table with the latest entry first.

3.7 View system environment changes: environment_v.php

This script is responsible for inspecting any system environment changes detected by back-end script. It uses the same table generation method used in the **journal_v.php** and thus displays the latest change first. The **environment** table in DB has many fields and the output from this script is quite ugly. To view the full content of the table, it is necessary to scroll left and right from a web browser.

3.8 HDF Performance Warning Report via e-mail: daily.php

This script is a new graph display script that accepts parameters from URL. This replaced **today.php** in the daily e-mail warning report since it allows users to view the graph on a specific date. For example, the warning e-mail may contain the URL like:

http://hdfdap.hdfgroup.uiuc.edu/h5perf/
daily.php?version=1.9.1&start=20080322&end=20080329

By putting the dates and version in the URL along with this script, HPF user can send an interesting graph to others more easily by forwarding the warning e-mail on a specific date.

4 Future Work

In the future we think it would be helpful to have an option that highlights the highest point of the graph. We have also been thinking that it would be a good feature to zoom in or out a part of graph and present more information about particular days. The ultimate HPF front-end will look like Google stock chart that integrates both journal entries, system environment changes and graphs in a single page per instance.