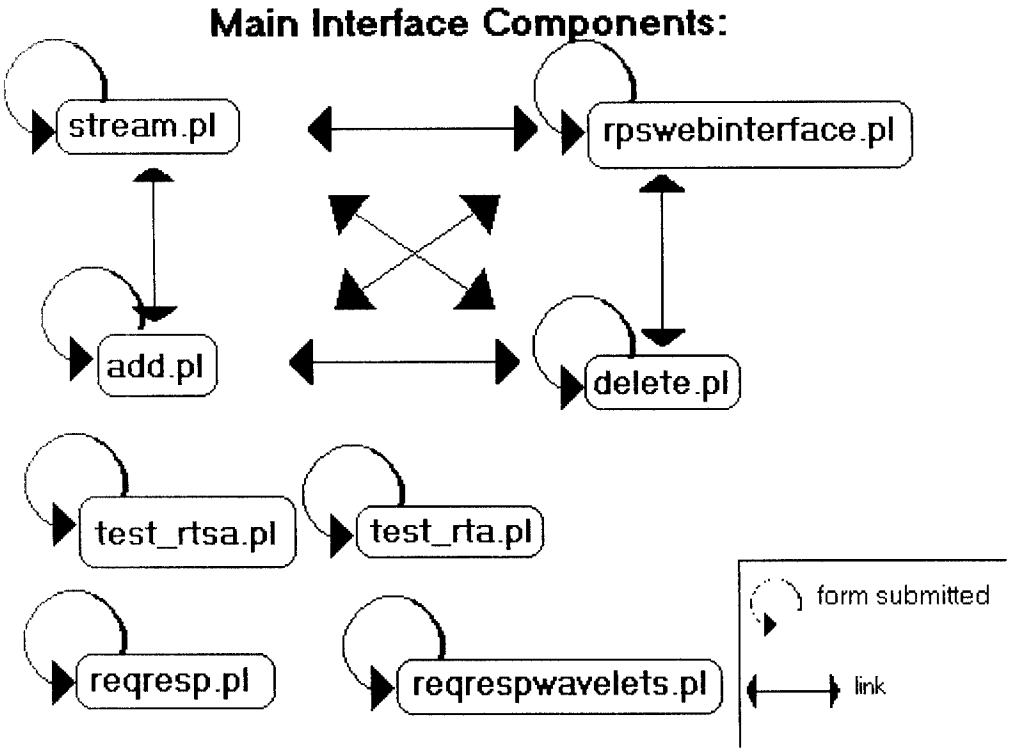
RPS Web Interface

Alex Shoykhet

Summary:
This is a description of the installation and usage of the RPS web interface.

Overview of RPS Web Interface:



component	purpose	uses
add.pl	add endpoints to database	cgi , rps database module
delete.pl	delete endpoints from database	cgi , rps database module
reqresp.pl	user gets predictions based on a time signal they upload	cgi , gnuplot
reqrespwavelets.pl	user gets wavelet transform based on a time signal they upload	cgi, gnuplot
rpswebinterface.pl	graph/print buffered predictions/measurements	cgi , gnuplot, rps database module
stream.pl	graph/print streaming predictions/measurements	cgi , javascript for output, rps database module
test_rta.pl	get running time predictions for a process	cgi, gnuplot , rps database module
test_rtsa.pl	get resource suggestion based on deadline for a process	cgi, gnuplot, rps database module

uitlity	function
purge.pl	cleans out the gnuplot and temp directories
start_pred_reqresp_server.pl	starts the server for reqresp.pl on a prespecified port

Installation:

General Steps:
Move all of the scripts, and the db,gnuplot,images,temp,and RPS folders (and their contents) to the same folder, make sure everything has executable permission.

Database:
This interface can run either with Oracle or a text-based interface, to switch between the two, change the line at the top of each Perl script that calls initEnv.
("ORACLE","bufferclients" | "streamclients") change to
("TEXT", "db/bufferclients.txt" | "db/streamclients.txt") depending on whether you want to access the streaming resources or the buffered resources.

To set up the text database, you need to have the folder db in the same folder as your perl scripts, and it needs to contain the files bufferclients.txt and streamclients.txt.
Note: the text version of the database does not have any authentication yet.

To set up the Oracle database, you need to run the sql file createtables.sql, located in the db folder, in sqlplus. If the user or password of the database changes, you can change the corresponding values in the rps_env module.

Modules:
rps_env.pm :
This module sets the environment variables, and calls the appropriate database constructor based on whether the Oracle or Text database was selected.
rps_oracle_db.pm , rps_text_db.pm:
These modules contain all the database access subroutines, their appropriate constructors are called from rps_env.pm.

Perl Scripts:
The perl scripts are good to go as is, just make sure they're all in the same folder and be careful if you rename them. Also, you'll need to modify these to switch database types (as described in the Database setup portion of the installation instructions).

Other folders:

gnuplot is the folder where all the gnuplot data will be saved, make sure it's in the same directory as the perl scripts.

images is the folder where all the non plotting images are stored (buttons) and where the images for the javascript plotter of stream.pl are stored.

temp is the folder where uploaded files for the request response scripts are stored.
Note: purge.pl cleans out the folders gnuplot and temp, so be careful renaming them.

RPS is the folder where the perl modules are kept.

Usage:

Initialization:
Run purge.pl to keep the gnuplot and temp folders clean.
Run start_pred_reqresp_server.pl to start the pred_reqresp_server on port 7777.

Using the components:

add.pl:
This script simply inserts the form data that the user enters into the database. There is a check to make sure that all the form data is filled in and that the RPS client that the user entered is actually up and running (it does this by making sure that it gets data output from the client).

delete.pl
This script pulls up all of the unique (address, client , port) combinations and puts them in a select list. The user chooses one to delete and presses delete to remove it from the database.

reqresp.pl
This script allows the user to either upload a time-domain file, or enter text in a (1,2,3,4..) format. If they enter both, the script chooses the file. They also enter a buffer depth, an optional modifier, and the model that they want used. There is a help option so that they can see what models and optional modifiers exist.

help.pl
this is used by reqresp.pl and calls pred_reqresp_client to get its usage output and prints it in a popup window.

After the form is submitted, the script checks to see that the model is valid (by parsing the output of help and making sure that the correct number of parameters were included with the model). Then, pred_reqresp_client is called to get the output, which is plotted using gnuplot.
Note: The original measurements are not plotted, this can be an easy future extension.

reqrespwavelets.pl
This script allows the user to either upload a time-domain, or enter text in a (1,2,3,4...) format. If they enter both, the script chooses the file. They also enter a wavelet type (daub4 for example), number of levels, and a transform type. The script calls sample_static_sfwt to get the wavelet transform of the input data, and graphs it using gnuplot. The data is then put into a file and sample_static_srwt is called to reconstruct the time domain data out of the wavelet transform. The reconstructed signal and the original signal are put side by side for comparison.
Note: There is a "hook" in the code for dynamic and block wavelet types.

rpswebinterface.pl
This script allows the user to graph buffered clients (predictions | | measurements). They also have control over the number of graphs, buffer depth, output (graph | | text), error/no

error bars for predictions. Once the form is submitted, the data is split into measurements, predictions, and errors, gnuplot is called using the jpeg terminal type to graph the data.

autorefresh.pl:
this script is almost exactly equal to rpswebinterface except that it refreshes itself based on a user input refresh frequency. There is also a javascript controlled pause, play, stop functionality.

stream.pl:
This script is paired with rpswebinterface.pl and plots streaming predictions and measurements. The user specifies how much data they want to see at a time (buffer depth), and a time to die for the stream: this is important so that there are no trailing processes.
After the form is submitted, the main form is printed and as many base images are printed side by side as was specified by the buffer depth input. The stream is started in a textarea. When graph stream is pressed, a javascript function is started that manipulates the images to make the image array visually display the data.
Note: measurements and predictions cannot currently be plotted at the same time. A simple fix to this would be to have two popup windows open side by side, with a measurement stream on the left and a prediction stream on the right.

test_rta.pl:
This script lets the user select several resources at once that have a host load prediction buffer running. The user enters a process time and output preference (whether they want the background measurements and predictions that the process running time estimation is based on graphed in addition to the regular output). Once the form is submitted, test_rta is called for each selected resource and also, if the user wanted graphical output, predictions and measurements are obtained in a similar manner to rpswebinterface.pl. When all of the test_rta data is collected, the output is printed in ascending order of process running time estimation (i.e. the resources where the process was said to run the fastest are printed first).

test_rtsa.pl:
This script is nearly identical to test_rta.pl except that the user must enter a deadline for the process that they want predictions on. The script calls test_rtsa.pl to get a recommendation on which machine is most likely to meet the deadline. All of the selected resources are then plotted, in ascending order of process running time estimation (using test_rta).

Future extensions:
There are several minor extensions that can be made to the system as is, they are all mentioned in the body of this paper under the "**Note:**" headings. The major extension that is left is to integrate the system with the Urgis codebank, both in style and usage. In my estimation this should not be a major undertaking, as Urgis seems fairly flexible.