Design document for performance measurement point

P.D.Mealor

May 7, 2003

1 Version history

11 Feb 2003 Initial version

6 May 2003 Updated to match actual implementation

2 Introduction

This document describes a design for a performance peasurement point (PMP) for stage one of the e2e piPES project. The PMP will consist of a daemon that accepts measurement schedules from some external source and controls a cron daemon. The cron daemon will call wrapper programs to perform measurements, extract results from the tools used and store them in a performance database.

Areas for further work include the interaction with the performance database, including how a local performance database could be queried by another host.

3 Overview of requirements

The performance measurement point (PMP) must perform the following tasks:

1. Accept a list of commands in a crontab (style) format
2. Execute those commands at the requested times
3. Store the results of those commands in a database (locally or remotely)

In the case of results stored in a local database, the PMP has to be able to respond to requests from a virtual, centralised performance measurements database.

3.1 Accept a list of commands

The architecture calls for the scheduler to generate a list of crontab-style times and commands which should be passed to the PMP for execution. The PMP must check that it is receiving commands from an authorised host and that the commands it is to execute are valid (real commands which it is authorised to execute).

3.2 Execute the list of commands

As commands arrive in a crontab (style) format, they can be easily transformed to a real crontab file and executed with minimal fuss by a cron daemon. The commands will equate to the name of a tool and the parameters to pass to it. These commands can be translated to the full pathname of a wrapper script which can extract the output of the tool and store it in a performance measurement database (which might be local or remote).

The commands to execute could include the server-side parts of tools which require a client-server setup. This could be used to reduce processor use and the number of open ports available on the PMP.

3.3 Store the result

Whether the performance database is local or remote is not specified. The piPES architecture calls for an central virtual performance database, which appears as a single database, but could perhaps just forward requests on to other databases.

4 Protocols and interfaces

The PMP control daemon listens on a well known (definable) port for connections from authorised sources.

4.1 Authorisation and authentication

We have no authorisation or authentication yet. Many security options exist which can be plugged in place of standard TCP sockets. These include SSL, GSI, Kerberos (?), ((argh: piPES' preferred option)).

4.2 Accept a list of commands

The command list consists of a series of lines of ascii text, each line specifying a tool and the times that tool should be run, followed by a line containing the termination sequence.

The format is based on the crontab format, and looks like this:

```
minute hour dayofmonth month dayofweek
tool [parameters...]
```

Each field is separated by by one or more whitespace characters (space or tab). The first five fields describe the time and date at which the tool should be run, and their format is described in crontab(5).

The tool field should be the name of a tool available on the PMP. The PMP will translate that name into a command line to run, performing all necessary wrapping to extract and store output. The optional parameters fields will be passed to the iperf program on the command line.

The termination sequence consists of the simple string "EOF".

4.3 Respond to database requests

We are unsure as to how to proceed with responding to database requests.

There are a number of solutions which could be implemented with the minimum of fuss:

MySQL [**?**] Measurements stored in a MySQL database can be queried remotely. Measurements could be stored centrally, although this is perhaps not a scalable solution. It would be harder to form a system whereby a virtual central database could be used to forward requests to separate databases stored on each PMP.

R-GMA [**?**] R-GMA is a distributed information system, which presents information from many sources as if it were a single relational database (with some caveats). More complicated to initially set up, R-GMA can be used to provide an integrated view of all measurements from all PMPs.

OGSA-DAI [**?**] This is an OGSA[**?**] interface to various databases. An OGSA-DAI approach may be more suited to a second phase, OGSA PMP.

MySQL seems the obvious choice at present: it is lightweight and simple to set up, and the implementation whether the actual database is local or centralised will be similar if not identical.

R-GMA would be more suited to a more grid-like structure, where PMPs are spread across more than one administrative region or the PMPs were more heterogeneous in their nature.

5 Internals

The PMP daemon is written in Perl. Perl is ideal for this sort of lightweight development server, as has excellent text-processing capabilities. Additionally, Perl's memory handling will reduce the chance of compromise via memory allocation/deallocation errors.

The PMP daemon will wait for a new connection to its port. When a connection is made, input lines will be parsed until the termination sequence is met or the connection is closed. If the connection is closed before the termination sequence is met, all input from this connection will be discard. When the termination sequence is met, the daemon will close the connection and rewrite its control crontab files.

Each line will consist of a crontab time specification (i.e. five space-separated fields specifying the time or times to run), followed by a tool *name* and finally

any parameters that will be passed to the wrapper script.

Each line of the input document will be check to ensure that it is valid. Any errors will result in the whole document being discarded. The lines are checked for the following items:

- The date and time formats must be correct
- The tool must be in a list of acceptable tools

Any shell meta-characters will be escaped if possible (or required).

Acceptable tools will be ones which have a wrapper program available in some well-defined place which the daemon can see. The crontab file will consist of a standard header which sets the system search path so so that the wrapper program directory is searched first. The incoming crontab file can then be appended to the crontab header once control characters have been escaped. The complete crontab file will be written to a file for the standard cron daemon to pick up.

5.1 Available tools

5.1.1 Iperf

The Iperf tool system is based on work by Yee-Ting Li to perform on-demand tests with Iperf[1]. This work is based on the EU-Datagrid's IperfER system[2].

The wrapper script accepts the same parameters as the version of Iperf it runs (version 1.7.0 as of 6 May 2003).

I could use YTL's tools-server program to launch the iperf server on the remote machine. This would have the advantage of not requiring the scheduler to worry about sceduling the server, however it might interfere with the normal running of the client.

6 Points for discussion

Explicitly stop overrunning programs? The schedule written to crontab could start a general wrapper program that can stop previously running sub-programs and generally clear up before starting the current program. We could have an rc.d-style hierarchy of start/stop programs for the various tools.

```
wrappers
 +-start.d
 |  +-iperf
 |  +-udpmon
 |  +-ping
 +-stop.d
    +-iperf
    +-udpmon
```

Or perhaps we could achieve the same result just having the general wrapper simply killing all other general wrappers, and hence any children of those wrappers (i.e. tool-specific wrappers, tools).

References

[1] Mark Gates, Ajay Tirumala, Jim Ferguson, Jon Dugan, Feng Qin, and Kevin Gibbs. Iperf.

[2] Yee-Ting Li. Iperfer. Based on PingER by Robin Tasker.