Design document for Grid Services Performance Measurement Point

P.D.Mealor

May 13, 2003

1 Version history

2 May 2003 Updated to match test implementation

4 March 2003 Initial version

2 Introduction

3 Overview of requirements

In "Design document for performance measurement point" a list of tasks that a PMP must perform was presented:

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).

However, for an OGSA PMP with a possibly wider use than E2EpiPES we must modify these requirements. A general PMP must be able to:

1. Advertise its capabilities;
2. Accept a schedule of measurements to make;
3. Perform those measurements at the requested times;
4. Store the results of those commands in a database (locally or remotely).

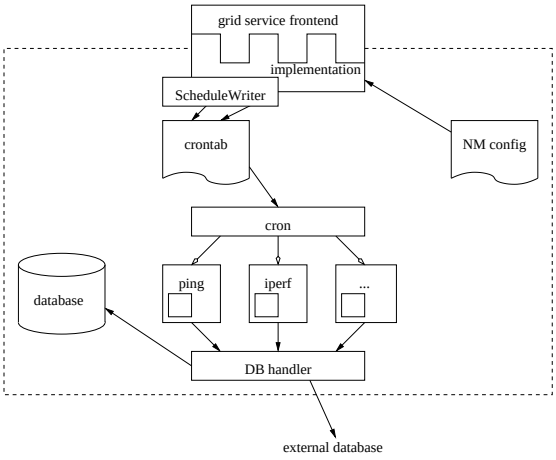

Figure 1: The control interface and workings

4 Design overview

Figure 1 shows the structure of a possibly implementation of an OGSA PMP. This is the structure that we will use as a first attempt.

The Grid Service front end presents functions and service data to the outside world. Behind it sits the specific implementation.

The PMP configuration is located in a file, probably in XML (as we likely have the tools for XML parsing readily at hand); at present the a Java "properties" files is used. This configuration contains the names of the tools available for the PMP, the characteristics those tools can measure, the mappings between measurement parameters and command-line options passed to the tools, and whether the tool needs a server and the name and versioning information required to match to a server tool.

The grid service uses implementations of the ScheduleWriter interface to write out the schedules. Different ScheduleWriters can be used for different tools. The default ScheduleWriter creates crontab files, the cron program is used to instantiate measurements.

Tools are run via a wrapper script, which allows for considerable simplification of the PMP frontend implementation configuration. The wrapper scripts also collect up the output of the tools to pass to pass to the database handler. The database handler is separately configured, and can store measurement results in a local database for retrieval later or in some remotely located database.

5 Accessing a PMP service

5.1 Finding a PMP

This is not implemented yet. A OGSA client would perform a search on an OGSA registry to find the PMP it requires.

5.1.1 Tools which require a server

A tool that requires a server on the target host is identified by having at least one serverRequirements element in its availableCharacteristics service

```
<availableCharacteristic>
  <name>bandwidth.availableBandwidth</name>
  <parameter>
    <name>client.sendBufferSize</name>
    <minValue>24</minValue>
  </parameter>
  <serverRequirements>
    <toolName>bandmeter</toolName>
    <version>1.5.5</version>
  </serverRequirements>
</characteristicSchedule>
<availableServer>
  <toolName>bandmeter</toolName>
  <parameter>
    <name>server.receiveBufferSize</name>
  </parameter>
  <version>1.8.0</version>
  <minClientVersion>1</minClientVersion>
  <maxClientVersion>1</maxClientVersion>
</availableServier>
<availableServer>
  <toolName>bandmeter</toolName>
  <parameter>
    <name>server.receiveBufferSize</name>
  </parameter>
  <version>2.0.1</version>
  <minClientVersion>2</minClientVersion>
</availableServer>
```

Figure 2: An example of the service data required to publish that the PMP can measure available bandwidth using the fictional bandmeter tool. The PMP has a client version 1.5.5, and two server versions: 1.8.0, which can handle any version 1.x.x client; 2.0.1 which can handle any version 2 or greater client.

```
<schedule>
<characteristicSchedule>
  <name>bandwidth.availableBandwidth</name>
  <source>192.168.0.5</source>
  <target>192.168.0.7</source>
  <parameter>
    <name>client.sendBufferSize</name>
    <value>1024</value>
  </parameter>
  <when>
    <minute>12</minute>
    <hour>1-23/2</hour>
    <dayOfMonth>*</dayOfMonth>
    <month>*</month>
    <dayOfWeek>mon,tue,wed,thu,fri</dayOfWeek>
  </when>
  <tool>
    <toolName>bandmeter</toolName>
    <version>1.5.5</version>
  </tool>
</characteristicSchedule>

</schedule>
```

Figure 3: An example of a schedule. The characteristicSchedule indicates that a measurement of available throughput should be made from 192.168.0.5 to 192.168.0.7, with a client buffer size of 1024. Although unnecessary in this case, the PMP is also instructed to use bandmeter version 1.5.5. Measurements are made on weekdays at 12 minutes past odd-numbered hours.

```
<schedule>

<serverSchedule>
  <toolName>bandmeter</toolName>
  <serverVersion>2.0.1</serverVersion>
  <clientVersion>2.2.0</clientVersion>
  <source>192.168.0.12</source>
  <source>192.168.0.11</source>
  <target>192.168.0.5</target>
  <when>
    <minute>12</minute>
    <hour>0-23/2</hour>
    <dayOfMonth>*</dayOfMonth>
    <month>*</month>
    <dayOfWeek>1-5</dayOfWeek>
  </when>
  <duration>25 minutes</duration>
</serverSchedule>

</schedule>
```

Figure 4: An example of a schedule. The serverSchedule indicates that a version 2.0.1 bandmeter server accepting messages from 192.168.0.11 and 12 should be started on 192.168.0.5 at 12 minutes past even-numbered hours on weekdays, and that it should remain available for 25 minutes.

data. Each serverRequirements element indicates the "name" of the server tool required, the version of the client-side version of the tool and optionally bounds on the server-side version number.

PMPs running a server tool advertise this with availableService elements in its service data. This element indicates the available server tools, their versions and any bounds on the client-side version numbers that the server tool can talk to.

The OGSA client would have to match the serverRequirements to the availableService, and then talk to the two PMPs required.

5.2 Submitting a schedule

To submit a schedule, the client calls one of two functions. Reportdfdj

6 Issues

6.1 Negotiation of tool server/client

Do we make the OGSA client work out which PMPs have compatible tools? Might we, perhaps, make the PMP search for a PMP with a compatible server tool and return a failure if it can't?

6.2 Distributed scheduling

Can we make the PMPs themselves perform scheduling? Perhaps when we receive a schedule, we can negotiate with other PMPs to ensure that we don't have crossing paths and so on, and adjust the schedule that actually gets used. The schedule would therefore just be the OGSA client's preferred option, while the PMPs (fairly continuously) negotiate on what to actually do.

We then have problems of how to guarantee that once a schedule has been accepted all the measurements will be carried out. Either that or we have to be able to signal retrospectively that a measurement has been dropped.

6.3 Identifying schedules and measurements, plus authorisation

With a more intelligent PMP, perhaps we want to be able to allow OGSA clients to be able to update their schedules. This means that schedules must be kept in full (in some form or another), and must contain some sort of identifying mark. This also raises the question of authorisation: perhaps we can use some sort of authentication/authorisation library to ensure that OGSA clients can only modify their own schedules.