# Contents

1 Introduction 3

2 Overview of Interfaces 3

3 Resource Descriptions 4

4 Resource Availability 6
   4.1 Get Possible Schedules 7
   4.2 Get Current Bookings 9

5 Resource Booking/Cancelation/Modification 11

6 Booking Status Monitoring 14

7 Ping Request for Checking RM Availability 15
Figure 1: URLs for accessing functionality through a HARC RM. These are all relative to the top-level URL of the RM, e.g. http://santaka.cct.lsu.edu:9191/santaka-rm

1 Introduction

This document describes the proposed interfaces for the HARM RMs, based upon changes identified during the recent G-Lambda collaboration meetings, plus some things that have needed doing for a while.

Notationally, in this document, information about errors is given in italics.

2 Overview of Interfaces

HARC RMs will presents five interfaces:

1. Resource description information;
2. Resource availability/booking information;
3. Resource booking/cancelation/modification;
4. Booking status monitoring; and
5. A “ping” request for checking RM availability.

Currently, only resource booking goes through the Acceptors. However, in time, all of these requests may be routed through the Acceptors; this would make the network design simpler, enabling greater use of private IP addresses within an organization. (It is my view that this will be necessary to support in the real world.)

Discussion  This change would also be possible to hide the Resource Manager endpoints from the client; RM Endpoint lookup would be function handled by the Acceptors. This change would not be significant to the RM, but would affect the Acceptor design. The HARC Java Client API would not change, however; so clients would be insulated from this change.

The URLs used for accessing these functions are summarized in the table in Figure 1.

Note that all methods now use POST. This is because sending messages from Java using URLConnection will always be sent by HTTP POST if there is any content specified! I think that this is wrong. But I can’t fix it.
states, the URL Components are tacked onto the end of the RM’s URL. So if you were requesting a booking status from a URL located at:

http://santaka.cct.lsu.edu:9191/santaka-rm

You would be using an HTTP GET from the following URL:

http://santaka.cct.lsu.edu:9191/santaka-rm/bookingStatus?bookingID=santaka.12345

3 Resource Descriptions

This interface allows a client to ask for a description of the resource. This description will include a plain language description, and may also include details about the size/shape of the resource. (However, it is also envisaged that some resources may want to conceal all/part of these details.)

<GetResourceDescriptions>
  <Resource>
    <Network>EnLIGHTened</Network>
    <Endpoint type="REST">
      <RESTEndpoint>http://127.0.0.1:9191/nrm</RESTEndpoint>
    </Endpoint>
  </Resource>
  <Resource>
    <Network>EnLIGHTenedII</Network>
    <Endpoint type="REST">
      <RESTEndpoint>http://127.0.0.1:9999/nrm2</RESTEndpoint>
    </Endpoint>
  </Resource>
  <Resource>
    <Compute>mike.cct.lsu.edu</Compute>
    <Endpoint type="REST">
      <RESTEndpoint>http://127.0.0.1:9090/mike-rm</RESTEndpoint>
    </Endpoint>
  </Resource>
</GetResourceDescriptions>

The document above might lead to the following response:

<ResourceDescriptions acceptor="http://127.0.0.1:8080/sample" location="http://127.0.0.1:8080/sample/resourceDescriptions" method="POST">
  <Resource>
    <Network>EnLIGHTened</Network>
    <Endpoint type="REST">
      <RESTEndpoint>http://127.0.0.1:9191/nrm</RESTEndpoint>
    </Endpoint>
  </Resource>
</ResourceDescriptions>
<Nodes>
  <Node>
    <Name>BT1</Name>
    <Text>LSU AIX Cluster</Text>
  </Node>
  <Node>
    <Name>BT2</Name>
    <Text>LSU SGI Altix</Text>
  </Node>
  <Node>
    <Name>BT3</Name>
    <Text>LSU 3rd Floor Frey</Text>
  </Node>
  <Node>
    <Name>CH1</Name>
    <Text>Chicago Compute Cluster</Text>
  </Node>
  <Node>
    <Name>LA1</Name>
    <Text>Caltech Cluster</Text>
  </Node>
  <Node>
    <Name>LA2</Name>
    <Text>Caltech Cluster</Text>
  </Node>
  <Node>
    <Name>RA1</Name>
    <Text>MCNC Compute Cluster</Text>
  </Node>
  <Node>
    <Name>UR3</Name>
    <Text>Caltech 6509</Text>
  </Node>
  <Node>
    <Name>VC1</Name>
    <Text>NCSU VCL Compute Cluster</Text>
  </Node>
  <Node>
    <Name>X1U</Name>
    <Text>Japanese GLambda Network</Text>
  </Node>
</Nodes>

<Description>
The EnLIGHTened Testbed Network
</Description>

<Resource>
  <Network>EnLIGHTenedII</Network>
</Resource>
4 Resource Availability

Previously we had getTimetable, returning a list of the available resources; this was the only resource availability interface. Two key problems have been identified:

1. The timetable format makes it hard to aggregate resources under an RM;
2. The timetable does not allow the resource owner to hide the size of the resource (from G-Lambda team); and
3. After the timetable has been fetched by the client, the calculation of when the work can be scheduled is still non-trivial (this is masked in the API).

We propose to abandon getTimetable entirely, replacing it with two interfaces, one mandatory, one optional:

Get Possible Schedules (mandatory) which allows the user to supply Work elements, in order to discover possible Schedules.

Get Current Bookings (optional) which allows a client to obtain information about all bookings in the RM. This is required to allow the GNS-WSI and GL-CRM wrappers to be written easily.
4.1 Get Possible Schedules

The document supplied to Get Possible Schedules (sent in a GET from possibleSchedules) looks like this:

```xml
<GetPossibleSchedules>
  <Make>
    <Resource>
      <!-- Description of Resource -->
    </Resource>
    <Schedule>
      <!-- Optional initial scheduling constraints -->
    </Schedule>
    <Work>
      <!-- Description of Work -->
    </Work>
  </Make>
</GetPossibleSchedules>
```

There's no need to supply a Schedule element at all, but if supplied, it will be used to restrict the information returned by the RM. Three formats of Schedule are supported, following the AIST schema.

1. **Exact.** Only this one schedule is possible...

   ```xml
   <Schedule><TimeSpecification><Exact>
     <StartTime>2006-08-23T17:54:27Z</StartTime>
     <EndTime>2006-08-23T18:54:27Z</EndTime>
   </Exact></TimeSpecification></Schedule>
   ```

   If successful, you get the same Schedule element back in the PossibleSchedules document.

2. **Deadline.**

   ```xml
   <Schedule><TimeSpecification><Deadline>
     <DeadlineTime>2006-08-03T18:54:27Z</DeadlineTime>
     <Duration>7200</Duration>
   </Deadline></TimeSpecification></Schedule>
   ```

   You get something like this:

   ```xml
   <Schedule>
     <Interval><StartTime>2006-08-02T05:10:44Z</StartTime>
     <EndTime>2006-08-03T18:54:27Z</EndTime></Interval>
   </Schedule>
   ```
Each Interval represents a time when the resources are free. The Intervals reach up to the
deadline. Multiple intervals can be returned. Intervals smaller than the specified Duration are not returned.
(Note that unlike the AIST schema, here the Duration is optional - if not specified, intervals of
any size can be returned.)

3. Duration.

```xml
<Schedule><TimeSpecification><DurationOnly>
  <Duration>7200</Duration>
</DurationOnly></TimeSpecification></Schedule>
```

This is similar to case 2. The Intervals reach forward indefinitely.

Note that in all cases, the RM may choose to limit the information returned, more so than the user
requested.

```xml
<PossibleSchedules>
  <Make>
    <Resource>
      <!-- Description of Resource -->
    </Resource>
    <Schedule>
      <Interval>
        <StartTime>2006-06-21T00:00:00Z</StartTime>
        <EndTime>2006-06-21T08:00:00Z</EndTime>
      </Interval>
      <Interval>
        <StartTime>2006-06-21T16:00:00Z</StartTime>
        <EndTime>2006-06-22T00:00:00Z</EndTime>
      </Interval>
    </Schedule>
  </Make>
  <Make>
    <Resource>
      <!-- Description of Resource -->
    </Resource>
    <!-- No schedule possible -->
    <Error>Resources overbooked</Error>
    <Work>
      <!-- Description of Work -->
    </Work>
  </Make>
</PossibleSchedules>
```

There is an open question about how an RM should return information about multiple overlapping
requests to the same resource, e.g. where the client wishes to make two simultaneous allocations on
the same resource. (We did this in iGrid, with ten simultaneous reservations on loni1n01, each eight processors/one node.)

It is up to the RM implementation whether or not it shows tentative bookings. My feeling is that bookings should not be shown until they have been confirmed. Although tentative bookings made by the GL-CRM and GNS-WSI wrappers will look confirmed to HARC from the time they are Prepared in the G-Lambda world, the G-Lambda RNDS monitoring system will show the correct status, as this information is retrieved through the wrappers, which will be aware of the G-Lambda status of the bookings.

4.2 Get Current Bookings

This call can be simple. If you want the information on all the RM's bookings then it's just a fetch from the URL given in the table. If you want to be more specific, then you can specify these using the following document:

```xml
<GetCurrentBookings>
  <Resource>
    <Compute>santaka.cct.lsu.edu</Compute>
    <Endpoint/>
    <!-- Description of Endpoint -->
  </Resource>
  <Resource>
    <Network>enlightened</Network>
    <Endpoint/>
    <!-- Description of Endpoint -->
  </Resource>
</GetCurrentBookings>
```

Of course an RM can only return information on the resources which it knows about. It returns a `NoBookingSet` element containing the resource description, plus an `Error` for anything which it doesn't understand.

The main reason for supporting this is to allow integration with the G-Lambda framework, as it's needed in the GNS-WSI and GL-CRM wrappers. It would be possible to support this in a more long-term fashion, but it would have to be limited to certain authorized clients; I'm not comfortable with all clients being able to see all jobs by querying an RM directly. However, it would be possible to trust a portal, which you trusted to remove any "sensitive" information about bookings which did not belong to the user.

The document returned from the HTTP GET would look like this:

```xml
<GetCurrentBookings>
  <BookingSet>
    <Resource>
      <Compute>santaka.cct.lsu.edu</Compute>
      <Endpoint>
        ...
      </Endpoint>
    </Resource>
  </BookingSet>
</GetCurrentBookings>
```
For the SimpleComputeRM, I am now returning a very simple version of the information in the booking, rather than the full JSDL-compliant version. The SimpleComputeRM also accepts requests in this format too, in addition to supporting JSDL.

Here is an example for an NRM:

```xml
<CurrentBookings>
  <BookingSet>
    <Resource>
      <Compute>mike4.cct.lsu.edu</Compute>
      <Endpoint>
        <!-- Description of Endpoint -->
      </Endpoint>
    </Resource>
    <Error>No such resource at this RM</Error>
  </BookingSet>
</CurrentBookings>
```
I'm not sure about the format of the information above. I don't want the RMs to have to go to the
effort of constructing a full JSDL-compliant description of the bookings, but the above, which simple,
isn't very pleasing to me. I am open to suggestions regarding the format of the Work element (and the
Schedule).

5 Resource Booking/Cancelation/Modification

This is the most complicated interface, because it involves the Paxos Commit protocol. And the client
never sees these messages. To give a simple overview...

1. The RM receives a proposed set of Actions, which it considers. This is a "Prepare" message, and
it contains the Endpoints of all the acceptors.

2. As this is Paxos, the builds its decision in a Phase2A message, which contains a Prepared element
that contains an xsd:boolean that is true when the RM goes to the Prepared state, or false if it
goes to the Aborted state.
3. The RM posts this Phase2A message to all the acceptors.

4. The RM then receives either a Commit or Abort message containing the outcome of the overall transaction.

5. In the case that the RM does not receive a timely Commit or Abort message, it can discover the outcome of the transaction by querying any acceptor in the same way that the client does. (This was never implemented, but will be before the GLIF demo. It will be in the common code for all RMs.)

The messages look like this.

Here is a “Prepare” message.

```
<Prepare tid="paxos-ident-7">
  <Acceptors>
    <Endpoint isLeader="true"> <!-- Endpoint of first acceptor -->
    <Endpoint isLeader="false"> <!-- Endpoint of second acceptor -->
    <Endpoint isLeader="false"> <!-- Endpoint of third acceptor -->
  </Acceptors>
  <Instance>
    <SetPrepared>
      <TID>paxos-ident-7</TID>
      <Endpoint> <!-- Description of the Endpoint -->
    </SetPrepared>
  </Instance>
  <Make actionId="1">
    <!-- The action as specified by the client -->
  </Make>
  <Make actionId="2">
    <!-- The action as specified by the client -->
  </Make>
  <!-- more action elements could be here -->
</Prepare>
```

The rational behind the SetPrepared element is to do with Paxos. In terms of the Paxos Consensus algorithm, the Acceptors are trying to decide whether or not the RM went to the Prepared state. So the RM and the proposed action IDs form part of the identifier of that instance of Paxos Consensus. (There is one instance of Paxos Consensus for each RM.)

Here is a “Prepared” Phase2A message.

```
<Phase2A>
  <Endpoint> <!-- Endpoint of leader acceptor -->
  <Instance>
    <SetPrepared>
      <TID>paxos-ident-7</TID>
      <Endpoint> <!-- Description of the Endpoint -->
    </SetPrepared>
  </Instance>
</Phase2A>
```
Here is a “Aborted” Phase2A message.

In addition to returning the outcome, the Ident elements in the success case, and the Error elements in the failure case, form part of the “Value” of what is being decided in that RM’s instance of the Paxos Consensus.

The Commit and Abort messages are identical to the Prepare message, but with the Prepare element tag name changed to Commit or Abort respectively.
6 Booking Status Monitoring

A document like this is sent in the HTTP GET request:

```xml
<GetBookingStatus>
  <Booking>
    <Resource>
      <Compute>santaka.cct.lsu.edu</Compute>
      <Endpoint>
        <!-- Description of Endpoint -->
      </Endpoint>
    </Resource>
    <Ident>Reservation ID</Ident>
  </Booking>
  <Booking>
    <Resource>
      <Network>enlightened</Network>
      <Endpoint>
        <!-- Description of Endpoint -->
      </Endpoint>
    </Resource>
    <Ident>Reservation ID</Ident>
  </Booking>
</GetBookingStatus>
```

A document like this is returned:

```xml
<BookingStatus>
  <Booking>
    <Resource>
      <Compute>santaka.cct.lsu.edu</Compute>
      <Endpoint>
        <!-- Description of Endpoint -->
      </Endpoint>
    </Resource>
    <Ident>Reservation ID</Ident>
    <Status>Activated/Reserved</Status>
  </Booking>
  <Booking>
    <Resource>
      <Network>enlightened</Network>
      <Endpoint>
        <!-- Description of Endpoint -->
      </Endpoint>
    </Resource>
    <Ident>Reservation ID</Ident>
    <Error>Unknown Booking</Error>
  </Booking>
</BookingStatus>
```
7 Ping Request for Checking RM Availability

A document like this is sent in the HTTP GET request:

```xml
<Ping>
  xsd:any
</Ping>
```

The following is returned:

```xml
<Pinged>
  <!-- This is a copy of what was sent -->
  xsd:any
</Pinged>
```

The code to handle this will be implemented in the common RM code.