

HARC: The Highly-Available Resource Co-allocator

Jon MacLaren

Centre for Computation & Technology,
Louisiana State University, Baton Rouge, LA 70803.
`maclaren@cct.lsu.edu`

Abstract. HARC—the Highly-Available Resource Co-allocator—is a system for reserving multiple resources in a coordinated fashion. HARC can handle different types of resource, and has been used to reserve time on supercomputers across a US-wide testbed, together with dedicated lightpaths connecting the machines. At HARC’s core are a distributed set of processes called *Acceptors*, which provide a co-allocation service. HARC functions normally provided a majority of the Acceptors are working; this replication gives HARC its exceptional availability. The Paxos Commit protocol ensures that consistency across all Acceptors is maintained. This paper gives an overview of HARC, and explains both how it works and how it is used. We show that HARC’s design makes it easy for the community to contribute new components for co-allocating different types of resource, while the stability of the overall system is maintained.

1 Introduction

HARC—the Highly-Available Resource Co-allocator—is an extensible system for reserving multiple resources in a coordinated fashion. We call the reservation of multiple resources in a coordinated fashion *co-allocation*,¹ and use this term to cover two scenarios:

1. the reservation of resources for the same time, as required for *meta-computing* applications like TeraGyroid [1] and SPICE [11], and also for distributed visualization experiments of the sort demonstrated at iGrid 2005 [9]; and
2. the reservation of resources for different times, for the scheduling of *workflow* applications, e.g. by frameworks such as Pegasus [2].

The need for co-allocation for meta-computing applications is more obvious; these applications cannot be run unless all the required computational resources

¹ The term *co-scheduling* is sometimes used as a synonym for co-allocation in Grid computing, but has also been used in the High-Performance Computing Scheduling community to mean the scheduling of the multiple processes of a parallel job [17], and so we prefer to avoid this term.

The term *co-reservation* also appears in the literature, e.g. [10], and is synonymous with co-allocation, as defined here.

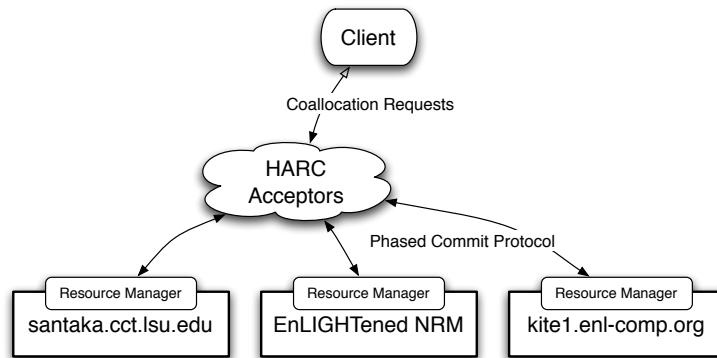


Fig. 1. The HARC architecture, showing the relationship between the Client, the Acceptors, and the Resource Managers (RMs).

are available at the same time. For workflow-based applications, the need for co-allocation arises from a desire to provide guarantees on turnaround. Currently, workflow execution engines submit the individual tasks for execution once all tasks they are dependent upon are completed; the tasks must then queue for resources. Using this method, the execution time for the overall workflow can vary dramatically. For example, if the average queuing time for available resources increases from a few minutes to one hour, then the time taken to execute 3-task workflow (where each task is dependent on the last) would increase by three hours. In addition to providing a certain finish time, the co-allocation of resources in this case would produce a better turnaround time.

The goal of HARC is to provide a co-allocation *service* suitable for both meta-computing and workflow applications. (Although it would be possible instead to provide the co-allocation functionality through a client library or program, this would limit the community’s ability to build higher-level services on top, e.g. resource brokers.) In both scenarios, there is a need to reserve a whole *bundle* of resources as if they were a single, indivisible resource. Using the terminology from the database community, this “all or nothing” behavior is known as *atomicity*.² To this end, HARC treats the allocation process as a Transaction, and uses a *Transaction Commit* protocol to reserve the resources.

The rest of this paper describes HARC in detail, starting with a presentation of the HARC architecture and message protocols in Section 2, which shows why HARC is a highly-available system. This is followed by Section 3 which focuses on the message contents, and shows how the message structure helps make HARC extensible. Section 4 shows how users can interact with HARC from the command line, and through the Java Client API which gives users a simple interface to HARC. Early results with HARC are given in Section 6; other co-allocation systems compared with HARC in Section 5. The paper concludes with Section 7.

² See <http://en.wikipedia.org/wiki/Atomicity>

2 Architecture and Message Protocol

The HARC Architecture, shown in Figure 1, consists of:

- *Clients* who requests resource co-allocations via
- *Acceptors* which make reservations by talking to
- *Resource Managers* which talk to local schedulers on each resource.

HARC uses Paxos Commit [6,7], a transaction commit protocol, to reserve multiple resources in a single, indivisible step. Paxos Commit (which is the application of Lamport’s Paxos Consensus algorithm [12] to the Transaction Commit problem) is a generalization of the classic 2-Phase Commit (2PC) protocol [13],³ replacing 2PC’s single *Transaction Manager* with multiple *Acceptors*.⁴ To understand how Paxos Commit allows us to create a Highly-Available system, we must first look at the exchange of messages in 2PC.

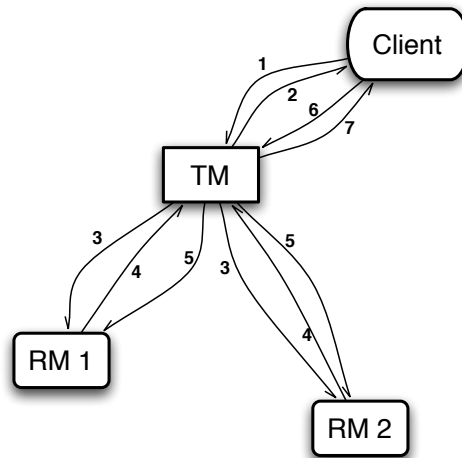


Fig. 2. Message exchanges for Two-Phase Commit.

Figure 2 shows the message exchange that would take place if 2PC were used to co-allocate reservations on two resources whose Resource Managers are RM 1 and RM 2. First, **(1)** the Client sends his/her request to the Transaction Manager (TM); this specifies the resources required, e.g. 16 processors on the two LONI [14] supercomputers *bluedawg* and *ducky*, from noon until 2 pm. **(2)** The TM sends back a Transaction Identifier (TID); the client will use this TID to poll for the results of the co-allocation. Next, **(3)** the TM sends *Prepare* messages to each of the RMs asking for the resources to be made available according to the Client’s request. **(4)** Each RM responds with a *Prepared* messages if they

³ Or see: http://en.wikipedia.org/wiki/Two-phase-commit_protocol

⁴ As noted in [6, Sec. 5], Paxos Commit with a single Acceptor is equivalent to 2PC.

are able to meet the Client's request (including the ID of the reservation) or an *Aborted* message if they cannot (including a message stating why this was not possible). The TM knows the outcome of the transaction once it receives a Prepared message from *all* RMs, i.e. *Commit*, or once it receives an Aborted message from *any* RM,⁵ i.e. *Abort*. (5) Once known, the TM sends the outcome is sent to each RM. (6) The next time that the Client polls for the result, (7) the outcome is returned to them, together with a reservation ID for each resource if the co-allocation succeeded, or with one or more error messages if it did not.

This sequence of messages makes it easy to understand the problem with 2PC; the Transaction Manager is completely central to the process. If the TM fails mid-transaction, or can no longer be reached, then the client may not be able to discover the outcome of the transaction. Worse still, some RMs may be left in the *Prepared* state, with resources put aside for a client, again, not knowing if the transaction was Committed or Aborted.

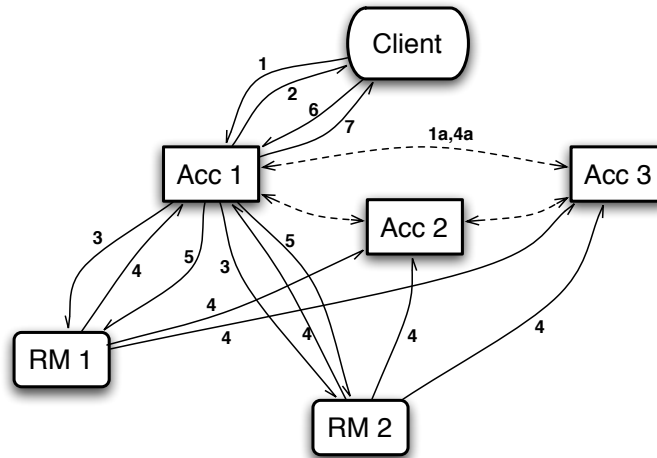


Fig. 3. Message exchanges for Paxos Commit.

Figure 3 shows the message sequence for the same co-allocation, but using Paxos Commit instead of 2PC. Acc 1, 2 and 3 are the three *Acceptors*, which replace the single Transaction Manager. Comparing this diagram with Figure 2, it should be clear that from the client's perspective, the sequence of messages is unchanged (although it may poll any Acceptor for its result). For a Resource Manager, the only change is that its *Prepared* or *Aborted* message must be sent to *all* Acceptors, rather than to the single TM.

The real difference between the two methods is the internal "discussion" between the Acceptors in steps (1a) and (4a). This is where the Paxos Consensus

⁵ In the case where an RM fails to respond within a given period, the RM is assumed to have Aborted.

algorithm is used. In the figure, we see that the Acceptor that the client first contacts with his/her request is the *Leader*. In the case where Acc 1 fails, then one of the other Acceptors will automatically take over as leader for that transaction. In this case, it may be Acc 2 or Acc 3 that sends the *Prepare* message to the RMs. The system functions normally, provided a majority of Acceptors keep working. So, given the Mean-Time To Failure and Mean-Time To Repair for a single Acceptor, you can achieve any required overall Mean-Time To Failure by deploying a sufficient number of Acceptors (see Section 2.3 below, for details).

An additional property of the system is that it still operates correctly in the case when messages are dropped or repeated.

- In the case where the client’s initial message does not reach the Acceptor, or where the Acceptor’s response does not get back to the client, the client can simply send the request again, to any Acceptor. The design of the messages ensures that HARC will only attempt to reserve the requested resources once.
- In the case where the *Prepare* message does not reach an RM, then the Acceptors will time-out the RM’s result, providing an Aborted response.
- Again, if *all* the *Prepared* messages from an RM to the Acceptors were dropped, then again, the Acceptors will assume that the RM aborted (although this is far less likely);
- In the case where the Commit/Abort message from the Acceptor to the RM is dropped, the RMs discover the outcome of the transaction by asking the Acceptors for the result.
- As the client is already polling for the outcome of the transaction, it does not matter if this request/response is dropped and resent.

The use of Paxos guarantees that clients and RMs will always get a consistent response, regardless of which Acceptor they talk to.

2.1 Non-Co-allocation Messages

In addition to co-allocation, HARC also allows clients to ask when their jobs could be timetabled, i.e. to discover start times when their co-allocation is likely to succeed; it is also possible to check on the status of previously-made reservations. The message protocol for these functions is far simpler, because the Acceptors do not hold any state concerning these messages, or need to agree upon an RM’s response, and hence there is no need to involve more than a single Acceptor.

The protocol is as follows. First, the Client formulates their enquiry, and **(1)** sends this to any Acceptor. This Acceptor separates the enquiry by RM, and **(2)** sends these to the RMs. **(3)** The RMs send the Acceptor their response, which the Acceptor collates before **(4)** sending this combined response back to the Client. If the client does not receive a response because of a dropped message, they can simply repeat their request to the same, or any other, Acceptor.

2.2 Security Model

In HARC, all messages are written in XML, and are sent directly over HTTPS (i.e. HTTP over an SSL connection). In the Web Services community, this approach is sometimes referred to as *Transport Level Security*, as opposed to *Message Level Security*, where parts of the XML messages are signed (and possibly encrypted), before transmission over a non-secure transport, like HTTP. The key advantages to Transport Level Security are speed (the cryptography is more likely to be done in C, rather than Java) and simplicity (when using message level security, you must be careful not to leave the system open to replay attacks). The disadvantage of this approach is that it is less flexible; in particular, it is not possible for a user to formulate a request to HARC where different credentials must be used at different sites (although this is not required in general).

All HTTPS connections are initiated using X.509 Certificates, meaning that each Acceptor and each HARC RM must have its own X.509 credential; these typically have a distinguished name that looks like this, for an acceptor:

```
/C=UK/O=eScience/OU=Manchester/L=MC/CN=harcacceptor/man4.nw-grid.ac.uk
```

Or like this, for an RM:

```
/C=UK/O=eScience/OU=Manchester/L=MC/CN=harccrm/man2.nw-grid.ac.uk
```

The HARC Acceptors authenticate all users, verifying the certificate chain that is presented to them, based on a set of trusted Certificate Authority (CA) certificates. In addition to accepting the use of standard X.509 Certificates, Acceptors are typically configured to accept GSI (Grid Security Infrastructure) Proxy Certificates [19], as used with the Globus Toolkit [5]. Having identified the Distinguished Name (DN) of the client, the Acceptors then embed this into the messages that are sent to the Resource Managers.

Each RM first authenticates the Acceptor, again using a list of CA certificates. Next, the Distinguished Name of the Acceptor is checked against a list of pre-configured Acceptor DNs; only if the Acceptor's DN is recognized, will the message be processed further. Next, the RM attempts to authorize the client's DN, as passed to it by the Acceptor, by checking it against a file that maps DNs to local usernames. This file has the same format as a Globus "gridmap" file and, for Compute Resources, this file will typically be a symbolic link to the Globus gridmap file for that machine.

In addition to the authentication of user requests, HARC Acceptors also authenticate the *Prepared/Aborted* responses from RMs. Paxos Consensus messages from other Acceptors are also authenticated, and then authorized against a list of Acceptor DNs.

2.3 HARC Mean-Time To Failure

In order to calculate the Mean-Time-To-Failure (MTTF) for HARC, i.e. the average time before an installation of HARC will cease to function, the formulae and terminology from [8, Sec. 3] are used. The calculation is made in terms

of the Mean-Time-To-Failure (MTTF) and Mean-Time-To-Repair (MTTR) of a single Acceptor. It is assumed that the Acceptors are writing their state to stable storage, so that they may be easily repaired by being restarted. Non-repairable faults, such as disk crashes, are not modeled.

A deployment of HARC with $2F + 1$ Acceptors will fail if any F Acceptors are unavailable, and a further Acceptor fails. From [8, Eqn. 3.9], the probability that a Specific Acceptor, n , fails is:

$$P_n \approx \frac{1}{MTTF}$$

The system will fail if Acceptor n fails, and any F of the other $2F$ Acceptors are unavailable. From [8, Eqn. 3.7], the probability that a particular Acceptor is unavailable is:

$$P_1 \approx \left(\frac{MTTR}{MTTF + MTTR} \right) \approx \left(\frac{MTTR}{MTTF} \right)$$

since $MTTR \ll MTTF$. So the probability that any F of the other $2F$ Acceptors are unavailable is:

$$P_{any-F} \approx \binom{2F}{F} \cdot \left(\frac{MTTR}{MTTF} \right)^F$$

The probability that both events occur together is:

$$P_n \cdot P_{any-F} \approx \binom{2F}{F} \cdot \left(\frac{1}{MTTF} \right) \cdot \left(\frac{MTTR}{MTTF} \right)^F$$

Finally, there are $2F + 1$ Acceptors. The chance that any one can cause the failure is:

$$P_{HARC} \approx \binom{2F}{F} \cdot \binom{2F+1}{2F+1} \cdot \left(\frac{MTTR}{MTTF} \right)^F$$

This gives us a MTTF of the whole system of:

$$MTTF_{2F+1} \approx \left(\frac{1}{\binom{2F}{F}} \right) \cdot \left(\frac{MTTF}{2F+1} \right) \cdot \left(\frac{MTTF}{MTTR} \right)^F$$

Let's conservatively assume that the MTTF of a single Acceptor is 168 hours (1 week), and that following a failure, the MTTR is 4 hours. Then, a deployment with seven Acceptors (i.e. $F = 3$), would have a MTTF of 88,905.6 hours, which is over 10 years. Note that in a production environment, where the Acceptors were deployed in a reliable environment, such as a web farm, the MTTF to MTTR ratio would be greatly improved, and a high level of availability attained with less Acceptors.

Although these numbers are impressive, we are aware that there are certain failures, e.g. wide-area network failures, that might partition the Acceptors in such a way that they cannot continue to make progress, i.e. so that there is no majority of acceptors which can still communicate with each other. Although these failures are not modeled in the above equations, the value of being able to co-allocate across a distributed system during such failures is highly questionable.

3 Message Structure and Content

HARC co-allocation request messages consist of a set of *Actions*, each of which expresses the user's desire to create, manipulate or cancel a reservation. Specifically:

Make actions are used to create reservations;

Modify actions are used to change reservations (start time, end time, or resource quantity); and

Cancel actions are used to remove reservations.

So, to co-allocate the LONI machines **bluedawg** and **ducky**, a request containing two *Make* actions is sent; the message in Figure 4 would try to schedule 16 CPUs on both machines for two hours, starting from 12 noon on the 25th of June 2007 UTC. As can be seen from the example message, each *Make* action contains three things:

- a *Resource* element which specifies **where** the reservation is to be made;
- a *Schedule* element which says **when** the reservation is for; and
- a *Work* element which specifies **what** is to be reserved.

If the co-allocation was successful, the response message would look like the message shown in Figure 5. The **Ident** elements contain the reservation IDs for each resource. (These identifiers will be used to refer to the reservations in the future.)

The return message includes a **Schedule** and a **Work** element for each of the booked resources. These represent the final schedule for the job, and the actual resources reserved, which, in certain situations, can differ from those requested. For example, the returned **Work** element can be different if you request four CPUs on a SMP Cluster with 8 CPUs per node; here, the scheduler may allocate 8 CPUs. Similarly, an RM may accept an approximate **Schedule** element to book against, containing only a deadline, but can then return an exact schedule once the booking has been made.

If the booking did not succeed, a message with a top-level **ActionsFailed** element is returned. The structure of this is similar, but contains one or more **Error** elements, stating why specific resources could not be reserved.

The other actions have similarly simple message structures.

- *Cancel* actions contain a **Resource** element, and an **Ident** element to refer to the reservation being canceled.
- *Modify* actions contain a **Resource** element and an **Ident** element, plus new **Schedule** and/or **Work** elements, depending on what is to be changed.

3.1 Combining Actions

Even after a set of resources have been co-allocated, the resulting reservations can be manipulated separately; if the user wished to cancel only two of four

```

<?xml version="1.0" encoding="UTF-8"?>
<Actions uid="D96290E2-4651-47E1-8091-AFF83B406E6A">
  <Make actionCount="0">
    <Resource>
      <Compute>bluedawg.loni.org</Compute>
      <Endpoint type="REST">
        <RESTEndpoint>https://bluedawg.loni.org:9393/bluedawg-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule><TimeSpecification><Exact>
      <StartTime>2007-06-25T12:00:00Z</StartTime>
      <EndTime>2007-06-25T14:00:00Z</EndTime>
    </Exact></TimeSpecification></Schedule>
    <Work>
      <SimpleCompute>
        <CPUCount>16</CPUCount>
      </SimpleCompute>
    </Work>
  </Make>
  <Make actionCount="1">
    <Resource>
      <Compute>ducky.loni.org</Compute>
      <Endpoint type="REST">
        <RESTEndpoint>https://ducky.loni.org:9393/ducky-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule> ... </Schedule>
    <Work> ... </Work>
  </Make>
</Actions>

```

Fig. 4. Example HARC co-allocation request (2nd Schedule and Work elements omitted).

```

<?xml version="1.0" encoding="UTF-8"?>
<ActionsSucceeded
  tid="TID135_d0c5e26b-693f-42d1-8b71-f4dda775a3d7"
  uid="84d15fc4-b800-49a7-988c-0cd6ba5cdbe1">
  <Make actionCount="0">
    <Resource>
      <Compute>bluedawg.loni.org</Compute>
      <Endpoint type="REST">
        <RESTEndpoint>https://bluedawg.loni.org:9393/bluedawg-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule> ... </Schedule>
    <Work> ... </Work>
    <Ident>l1f1n01.103.r</Ident>
  </Make>
  <Make actionCount="1">
    <Resource>
      <Compute>ducky.loni.org</Compute>
      <Endpoint type="REST">
        <RESTEndpoint>https://ducky.loni.org:9393/ducky-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule> ... </Schedule>
    <Work> ... <SimpleCompute>
      <Ident>l2f1n01.106.r</Ident>
    </Work>
  </Make>
</ActionsSucceeded>

```

Fig. 5. HARC response following a successful co-allocation.

co-allocated resources, they can do so. (Similarly, it is possible to manipulate previous unconnected reservations in the same, new request.) Also, it is possible to combine Actions of different *types* in the same co-allocation request, e.g. two Makes, plus a Cancel. This could be used by a workflow execution engine such as Pegasus [2], when amending the schedule of a long-running workflow.

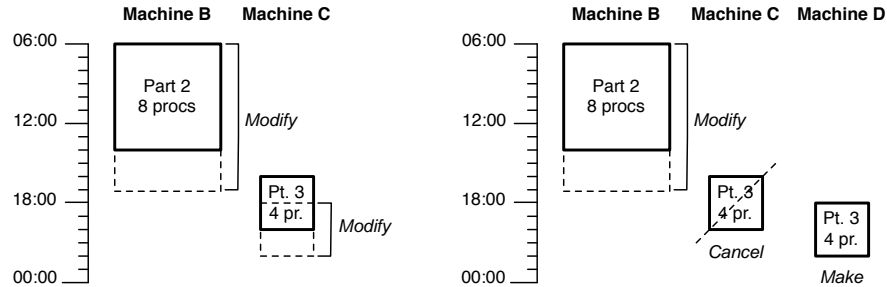


Fig. 6. Two attempts to re-scheduling a simple workflow.

To see how HARC could be used to help schedule a workflow, consider a simple example consisting of three tasks, which are to be run one after the next, with gaps of at least one hour in-between for file transfer purposes. Initially, the workflow execution engine co-allocates resources for all three parts of the job: Part 1 running on Machine A for 4 hours, Part 2 on Machine B for 8 hours, and Part 3 on Machine C for 4 hours. While the second part is running, a monitoring system discovers that this computation is running slower than expected, and will take an additional between 10 and 11 hours to complete; the 8 allocated hours are insufficient.

Figure 6 shows two attempts to amend the schedule. First, the engine tries to extend the allocated time for Part 2 and change the schedule of Part 3, by sending a request with two Modify actions. Unfortunately, the RM for Machine C does not support Modify, causing the transaction to be aborted; the initial schedule remains. The engine then tries a new strategy: instead of delaying the third part, a Cancel for Part 3 is combined with the same Modify for Part 2, together with a Make for Machine D, which will now be used for executing Part 3.

Note, that if this second re-scheduling strategy succeeds, the client will still need to make other adjustments to ensure that any file transfers to/from Machine C are redirected to Machine D, and ensure that the job that is to run on Machine D gets submitted to the new reservation. These processes are outside the scope of HARC.

3.2 Processing the Messages

Looking again at the example message shown Figure 4, in the context of the message exchange pattern shown in Figure 3, it should become clear that the only part of the Make action that the Acceptors need be concerned with is the **Resource** element; the **Schedule** and **Work** elements are the concern of the Resource Manager alone. This means that the Acceptor code does not require modification or adaptation when new Resource Managers are added to the system, even if these are for completely new **types** of resource, e.g. a Grid Storage service, or Google Calendar.

This makes HARC easy to extend, and also makes it ideal for a community that produces open-source software, such as the Grid Computing community. Anyone can contribute new or adapted Resource Managers back to the community, without compromising the stability of the overall system, which stems from the stability of the Acceptors. The code for the Acceptors can be tightly controlled by a small group of trusted experts. As this code seldom changes, its stability will increase with time, as fixes are required less often, and as fewer and fewer bugs remain.

In addition to this, the Resource Manager code, which is written Object-Oriented Perl, has been designed to ease the production of new Resource Managers. All the message protocol code has been divided off into separate modules, which again, should not need to change.

This section concludes with a quick look at the Compute and Network Resource Managers which have been written for HARC.

Compute Resource Manager The HARC Compute Resource Manager can work with any batch scheduler that supports advance reservations. The modules currently in CVS support LoadLeveler, Torque with Maui and Torque with Moab; modules that support LSF and PBSPro will be added shortly. Typically, it is possible to avoid running HARC Compute Resource Managers (CRMs) as root, because most schedulers allow the specification of an Access Control List (ACL) at the time the reservation is made, controlling which users and groups may submit jobs to the reservation. The CRM can simply look up the client's username in a mapfile, and then make a reservation that the client can access, e.g. in Torque with Maui:

```
setres -u maclaren -s 16:00 -d 1:00 TASKS==1
```

When an RM receives a *Prepare* message from the Acceptor, it considers whether or not the client's request can be met or not (after rejecting any unauthorized requests—see Section 2.2). In the Paxos Commit protocol (as with classic 2PC), responding with a *Prepared* message means that the RM is committed to meeting the request should the transaction be committed (it is not possible to say no to a *Commit* message). Therefore, when processing the *Prepare* message, the RM must ensure that the underlying scheduler sets aside the required resources. Where the scheduler supports two-phase commit, the RM can perform

the first phase of the resource booking at this point. Currently, only the Moab batch scheduler supports phased commit. In all other cases, the RM must in fact try to book the resources on behalf of the user; if (and only if) this succeeds, *Prepared* is returned. Later, if *Commit* is received, the booking is left in place; if *Abort* is received, the booking is canceled.

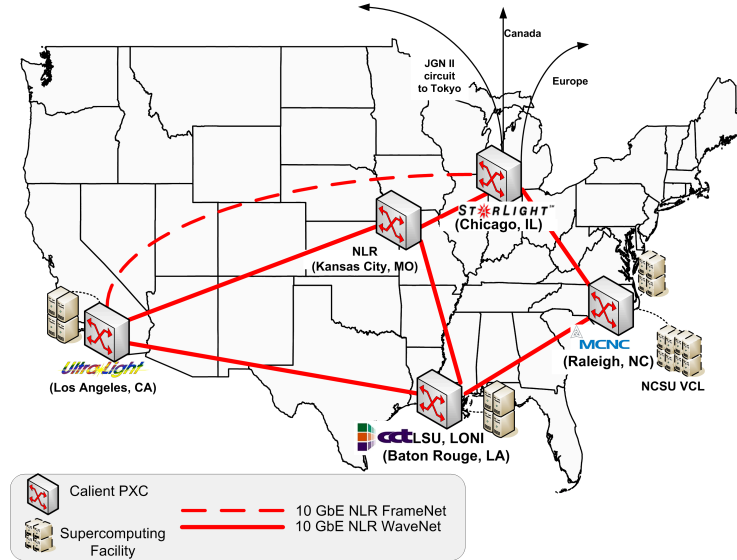


Fig. 7. EnLIGHTened computing testbed.

Network Resource Manager The EnLIGHTened project [3] testbed, as shown in Fig. 7, is a US-wide optical (Layer 1) network that has been deployed to facilitate the middleware and application goals as well as to support investigations into new network and control plane architectural models. The core of the testbed is built using Calient Networks Diamond Wave photonic cross-connect (PXC) switches interconnected by Ten Gigabit Ethernet (10 GbE) circuits provided by Cisco Systems and National Lambda Rail (NLR). GMPLS (Generalized Multi-Protocol Label Switching) [16] is used as the control plane protocol; this allows dynamic instantiation of end-to-end paths across the testbed. A prototype Network Resource Manager (NRM) was designed for HARC that could communicate with the Calient PXCs; to set up a lightpath, a TL1 command⁶ is sent to the switch at the one end of the lightpath. The HARC NRM is described in more detail in [15].

⁶ TL1, or *Transaction Language 1*, is a widely-used protocol for managing network elements such as switches and routers. For more details, see [18].

4 Using HARC to Run Meta-Computing Jobs

To save clients from needing to construct XML messages, there is a command-line interface to HARC. The commands are written in Java, using the Java Client API. This section briefly shows how these commands would be used to help run a meta-computing job—the most common use for HARC—using Globus to submit the work to the reservations. (An example of Client API code is given at the end of the section.) The basic steps of this are:

1. Book the resources using HARC;
2. Submit the jobs to the reservations using Globus;
3. Monitor the state of the reservations using HARC; and
4. Cancel the reservations (if time remains) using HARC

There is a function in HARC for discovering possible slots for booking the work. However, in general it is not possible to obtain good information from batch schedulers about when a reservation would succeed; only Moab supports this kind of query. Consequently, the information returned by the Resource Managers is not accurate. Instead, users simply attempt different co-allocations for different times.⁷

In the following sections, it is assumed that the user has already created a valid GSI proxy credential.

4.1 Booking the Resources

The following command will attempt to reserve 16 CPUs on both `bluedawg` and `ducky`, from 1pm (in the client's local timezone) for 2 hours.⁸

```
harc-reserve -c bluedawg.loni.org/16 \  
             -c ducky.loni.org/16 -s 13:00 -d 2:00
```

Upon success, the client is given the reservation IDs for the request:

```
bluedawg.loni.org/11f1n01.103.r  
ducky.loni.org/12f1n01.106.r
```

If the reservation is unsuccessful, the client will be told which node(s) refused the request and why, e.g.:

```
ducky.loni.org: llmkres: 2512-876 The reservation has not been created.  
ll_make_reservation() returns RESERVATION_TOO_CLOSE.
```

In this case, the reservation failed because the reservation time was too near. The client should change the start time for the request and retry.

⁷ As HARC returns a reason for the failure of a particular co-allocation, it is trivial for users to work out whether or not their co-allocation will work if retried with a different start time.

⁸ This is the same request as shown in Figure 4, assuming that the client is in the UK, during British Summer Time.

```

+
( &(resourceManagerContact="bluedawg.loni.org/jobmanager-loadleveler")
  (count=16)
  (reservation_id=11f1n01.103.r)
  (jobtype=mpi)
  (label="subjob 0")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0))
  ...
)
( &(resourceManagerContact="ducky.loni.org/jobmanager-loadleveler")
  (count=16)
  (reservation_id=12f1n01.106.r)
  (jobtype=mpi)
  (label="subjob 1")
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1))
  ...
)

```

Fig. 8. Example RSL for submitting an MPICH-G2 job to HARC reservations.

4.2 Submitting the Jobs to the Reservations

To allow the submission of jobs to the reservations using Globus, modifications to the pre-WS (i.e. pre-Web Services) jobmanagers are required.⁹ These modifications are easy to make, and documented on the HARC website. They extend the RSL of the job manager to allow the new RSL term `reservation_id` to be used to place the job into a reservation, rather than onto a queue. This is necessary only because the Globus Toolkit does not currently support job submission to reservations. The RSL required to submit an MPICH-G2 job to Globus using the MPICH `mpirun` (with the `-globusrs1` flag) is shown in Figure 8.

4.3 Monitoring the Reservations

HARC provides a command for enquiring after the status of reservations. A reservation is considered to be in one of three states:

RESERVED if the reservation has been made but not started;

ACTIVE if the reservation is currently working; or

UNKNOWN if the reservation has been canceled, or has passed (or if the given ID is not recognized at all).

The following command will discover the status of the reservations made earlier:

```

harc-status -r bluedawg.loni.org/11f1n01.103.r \
            -r ducky.loni.org/12f1n01.106.r

```

⁹ Similar extensions to WS-GRAM jobmanagers can also be made. It is also anticipated that future versions of the Globus Toolkit will support this directly.

During the time of the reservation, this should produce:

```
bluedawg.loni.org/11f1n01.103.r ACTIVE
ducky.loni.org/12f1n01.106.r ACTIVE
```

4.4 Canceling the Reservations

Once you are done with the reservations you have made, and if they still have some time remaining, you can cancel them.¹⁰ The following command would cancel the reservations created above.

```
harc-cancel -r bluedawg.loni.org/11f1n01.103.r \
            -r ducky.loni.org/12f1n01.106.r
```

4.5 The Client API

Although new commands will continue to be developed and added to HARC, there will be situations where it makes more sense for groups to write their own client programs, using the Java Client API, perhaps integrating the co-allocation process together with application-specific job submission code. HARC is distributed with a Java Client API which makes the construction of such clients easy. (The HARC Command Line tools are written using this API.) The example code in Figure 9 could be used to make two reservations on `bluedawg` and `ducky`.

5 Related Work

GARA, the General-purpose Architecture for Reservation and Allocation [4,10], is architecturally similar to HARC. The system allows clients to interface to a broad class of Resource Managers through a single mechanism, and so can reserve network bandwidth in addition to compute resources. GARA does not provide a co-allocation service, but rather provides a library which could be used to construct one; reliability would need to be addressed separately. GARA is no longer being developed.

Generic Universal Remote, or GUR [20] is a system for co-scheduling compute resources. GUR uses `ssh` (or `gsissh`) to log into the resources being co-scheduled, and then attempts to make the reservations directly. This approach requires no server-side software, and as such is capable of co-allocating any computational resources where the user has permission to make reservations. However, all configuration details for the resources (e.g. the type of batch scheduler, where the reservation commands are located, etc.) are held on the client, resulting in a more complex client installation than with HARC. GUR is a client-based tool, and it is not easy to see how it could be used to construct a co-allocation service. Unlike HARC, GUR is not extensible to non-computational resources.

¹⁰ Batch systems are often configured to release reservations once they are unused for a short time (e.g. five minutes), but in any case, it is good practice to release resources that you are not using.

```
CoallocatorFactory.loadProperties();
Coallocator co=CoallocatorFactory.getCoallocator();

Calendar cal=Calendar.getInstance();
cal.add(Calendar.MINUTE,30); Date startTime=cal.getTime();
cal.add(Calendar.MINUTE,120); Date endTime=cal.getTime();

Vector<MakeAction> makes=new Vector<MakeAction>();

ExactSchedule sched=new ExactSchedule(startTime,endTime);
SimpleComputeWork work=new SimpleComputeWork(16);

SimpleComputeResource bluedawg=
    SimpleComputeResource.forName("bluedawg.loni.org");
makes.add(new MakeAction(bluedawg,sched,work));

SimpleComputeResource ducky=
    SimpleComputeResource.forName("ducky.loni.org");
makes.add(new MakeAction(ducky,sched,work));

Coallocator.CoallocationResponse resp=co.coallocateActions(makes);
```

Fig. 9. Code using Client API to reserve 16 CPUs on bluedawg and ducky, for two hours, starting in 30 minutes.

6 Current Status and Early Results

There are three deployments of HARC in use today: those on the EnLIGHTened testbed [3] and the Louisiana Optical Network Initiative (LONI) infrastructure [14] in the United States; and a third on the NorthWest Grid (NW-GRID),¹¹ a regional Grid in the United Kingdom. A trial deployment is underway on part of the TeraGrid,¹² and HARC is also being evaluated for deployment on the UK National Grid Service.¹³

HARC was used in the high-profile EnLIGHTened/G-lambda experiments at GLIF 2006 and SC'06, where compute resources across the US and Japan were co-allocated together with end-to-end optical network connections;¹⁴ these are believed to be the largest scale co-allocations to date. HARC has also been used on a more regular basis, to schedule a subset of the optical network connections

¹¹ <http://www.nw-grid.ac.uk/>

¹² <http://www.teragrid.org/>

¹³ <http://www.ngs.ac.uk/>

¹⁴ See <http://www.gridtoday.com/grid/884756.html>

being used to broadcast Thomas Sterling's HPC Class from Louisiana State University.¹⁵

7 Conclusions

HARC is a reliable mechanism for co-allocating multiple resources of different types. As we have shown in this paper, it is of particular use to those who wish to run meta-computing jobs across multiple computers simultaneously, and also to those who wish to reserve resources for running scientific workflows.

HARC is designed to be extensible, and so new types of Resource Manager can be developed without requiring changes to the Acceptor code. This differentiates HARC from other co-allocation solutions. The most important next step for HARC is to increase its availability to end-users, and to try and make it part of the everyday Grid infrastructure. By doing this, we hope to get the Grid community using HARC, and encourage others to develop new Resource Managers for different resources, which can then be contributed back to the community.

Acknowledgements

The implementation of HARC took place while the author was employed at the Center of Computation & Technology at Louisiana State University. During this time, the work was supported in part by the National Science Foundation "EnLIGHTened Computing" project [3], NSF Award #0509465.

The original design for HARC was produced by Mark Mc Keown, while at the University of Manchester.

References

1. R. J. Blake, P. V. Coveney, P. Clarke, and S. M. Pickles. The teragyroid experiment—supercomputing 2003. *Scientific Computing*, 13(1):1–17, 2005.
2. E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
3. EnLIGHTened Computing: Highly-dynamic Applications Driving Adaptive Grid Resources. <http://www.enlightenedcomputing.org>.
4. I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *The Seventh IEEE/IFIP International Workshop on Quality of Service (IWQoS'99)*, pages 27–36. IEEE Computer Society Press, 1999.
5. Globus toolkit. <http://www.globus.org/toolkit/>.

¹⁵ This class is the First Distance Learning Course ever offered in Hi-Def Video. Participating locations include other sites in Louisiana, and Masaryk University the Czech Republic. See <http://www.cct.lsu.edu/news/news/201>

6. J. Gray and L. Lamport. Consensus on transaction commit. Technical Report MSR-TR-2003-96, Microsoft Research, January 2004. http://research.microsoft.com/research/pubs/view.aspx?tr_id=701.
7. J. Gray and L. Lamport. Consensus on transaction commit. *ACM TODS*, 31(1):130–160, March 2006.
8. J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.
9. A. Hutanu, G. Allen, S. D. Beck, P. Holub, H. Kaiser, A. Kulshrestha, M. Liška, J. MacLaren, L. Matyska, R. Paruchuri, S. Prohaska, E. Seidel, B. Ullmer, and S. Venkataraman. Distributed and collaborative visualization of large data sets using high-speed networks. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 22(8):1004–1010, 2006.
10. I. Foster and M. Fidler and A. Roy and V. Sander and L. Winkler. End-to-End Quality of Service for High-end Applications. *Computer Communications*, 27(14):1375–1388, 2004. <http://www.globus.org/alliance/publications/papers/e2e.pdf>.
11. S. Jha, M. J. Harvey, P. V. Coveney, N. Pezzi, S. Pickles, R. Pinning, and P. Clarke. Spice: Simulated pore interactive computing environment - using grid computing to understand dna translocation across protein nanopores embedded in lipid membranes. In *UK e-Science All Hands Meeting*. <http://www.allhands.org.uk/2005/proceedings>, 2005.
12. L. Lamport. Paxos Made Simple. In ACM SIGACT news distributed computing column 5. *SIGACT News*, 32(4):18–25, 2001. <http://research.microsoft.com/users/lamport/pubs/paxos-simple.pdf>.
13. B. W. Lampson and H. E. Sturgis. Crash recovery in a distributed data storage system. Technical Report (Unpublished), Xerox Palo Alto Research Center, 1976, 1979. <http://research.microsoft.com/Lampson/21-CrashRecovery/Acrobat.pdf>.
14. The Louisiana Optical Network Initiative (LONI). <http://www.loni.org/>.
15. J. MacLaren. Co-allocation of Compute and Network resources using HARC. In *Proceedings of "Lighting the Blue Touchpaper for UK e-Science: Closing Conference of the ESLEA Project*. PoS(ESLEA)016, 2007. http://pos.sissa.it/archive/conferences/041/016/ESLEA_016.pdf.
16. E. Mannie. Generalized Multi-Protocol Label Switching (GMPLS) Architecture. RFC 3945, The Internet Engineering Task Force (IETF), Oct. 2004. <http://www.ietf.org/rfc/rfc3945.txt>.
17. C. McCann and J. Zahorjan. Scheduling Memory Constrained Jobs on Distributed Memory Parallel Computers. In *Proceedings of ACM SIGMETRICS'95/PERFORMANCE'95 Joint International Conference on Measurement and Modeling of Computer Systems*, pages 208–219, May 1995.
18. Beginners Guide to TL1. http://netcoolusers.org/TL1/Beginners_Guide_to_TL1.
19. S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820, Internet Engineering Task Force, June 2004. <http://www.ietf.org/rfc/rfc3820.txt>.
20. K. Yoshimoto, P. A. Kovatch, and P. Andrews. Co-scheduling with user-settable reservations. In D. G. Feitelson, E. Frachtenberg, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing (JSSPP)*, volume 3834 of *Lecture Notes in Computer Science*, pages 146–156. Springer, 2005.