

# HARC: A Highly-Available Robust Co-scheduler

Jon MacLaren  
Center for Computation and Technology  
Louisiana State University  
Baton Rouge  
Louisiana 70803  
United States  
maclaren@cct.lsu.edu

Mark Mc Keown  
Research Support Services  
University of Manchester  
Oxford Road  
Manchester M13 9PL  
United Kingdom  
Mark.McKeown@manchester.ac.uk

## Abstract

*Distributed High-Performance Computing Applications are commonplace in several scientific domains. To correctly execute such applications requires the allocation of multiple resources; often, these resources will belong to multiple administrative domains, and so will not be under centralized control. Typically, dedicated resources of different types will be required, usually including compute resources and network bandwidth. This paper presents HARC (Highly-Available Robust Co-scheduler): a co-scheduling framework suitable for any resource under the control of a scheduler supporting reservations.*

*At the core of HARC is a replicated co-scheduler process, which gives the system fault-tolerance. Lamport's Paxos Consensus algorithm is used both to ensure consistency, and to ensure progress provided a majority of the replicas continue to function. It is shown that a deployment of seven replicas can be expected to have a Mean-Time-To-Failure measured in years. The design of HARC is presented in detail; specifically, HARC is shown to be well suited to the scheduling of large scientific workflows.*

*Experiences with the first implementation of HARC are presented. This includes the successful use of HARC to co-schedule reservations on ten compute nodes and two Calient Diamondwave optical network switches as part of a demo at iGrid 2005.*

## 1. Introduction and Motivation

Distributed applications such as TeraGyroid [4] and SPICE [25], which have been run across combined resources from the US TeraGrid [19] and UK National Grid Service [18], require these resources to be *co-scheduled*, i.e. to be made available at the same time or at some coordi-

nated set of times. These experiments make use of high-performance computers and networks, storage systems, visualization systems, AccessGrid nodes, haptic devices and people. At the time of the experiments there were only ad-hoc, time consuming procedures for reserving these resources. To make this type of experiment an every-day activity, the scientists will need to be able to easily co-schedule these distributed resources, and to do this through a simple software interface.

Fortunately, a growing number of these systems can be booked in advance using software: advance reservation systems are increasingly common on parallel compute resources; timetable booking systems are available for booking AccessGrid nodes; and several calendar/diary systems are network-accessible.

But making a coordinated booking for a set of these resources still involves interacting with a number of separate and often quite different interfaces. To co-schedule a set of resources, a client can pick a schedule for the resources, and then try to schedule each resource in turn. The key problem with this approach is that one or more of the resources may not be available at the required time. The user must then cancel any already-scheduled resources and try something else, either using a different set of resources, or using a different schedule. From a software engineering perspective, it makes sense to encapsulate this complexity somehow, either in a client library or in an external service. Also, to make either approach scalable, it should be possible to add the ability to schedule new types of resource without altering the client software, save, perhaps, for the updating of some configuration files.

The advantage of using an external service is that it is more readily available for use as part of other more complex or higher-level services, e.g. as part of an application-specific scheduler, or as the back-end of a workflow execution engine. The disadvantage is that an additional distributed component is being added to the big picture, which

is an additional source of failure in the overall system. This paper presents an external co-scheduling system, HARC, which is highly fault-tolerant through its use of Gray and Lamport's Paxos Commit protocol [12], and therefore suitable for use in a distributed environment or Grid.<sup>1</sup> In addition, HARC uses a system of *Resource Manager* services, which wrap the reservation systems of each resource. This allows HARC to provide a framework for co-scheduling any resource with suitable booking functionality; new resources and resource types can be added to the overall system without modification either to the core co-scheduler, or to any client end components. Details of a first implementation of HARC, called HARC/1, are also presented. HARC/1, which was built using HTTP and XML according to the REST [8] architectural style, was demonstrated at iGrid 2005 [22] and is available for download [15].

The rest of this paper is organized as follows. In Section 2, the co-scheduling process is seen to be a distributed transaction; Transaction Commit protocols are discussed, and a sketch of the Paxos Commit protocol—the basis of the HARC system—is given. The next three sections present the design of HARC: the architecture and core of the HARC system is laid out in Section 3; the four co-scheduling actions—Make, Modify, Move and Cancel—are explored in Section 4; and designs for two classes of Resource Manager are sketched out in Section 5. The key points of the HARC/1 implementation, and early experiences with it, are given in Section 6. Related work is covered in Section 7, and the paper closes with Section 8 which presents conclusions and ideas for future work.

## 2. The Co-scheduling Problem

The task of the co-scheduler then, is to take scheduling requests for multiple resources, and to attempt to enact these in a coordinated fashion, achieving one of two possible outcomes:

**Success** All scheduling requests are enacted; or

**Failure** No requests are enacted (and any tentatively made requests must be canceled).

Note that this work does not take the phrase “scheduling requests” to mean simply the construction of a schedule; manipulation and even cancelation of (parts of) a schedule are included. Further, the coordination of such requests is still referred to as co-scheduling.

This problem, where a coordinator must ensure that a number of independent Resource Managers arrive at a consistent state, is essentially a *distributed transaction*. To commit a distributed transaction, and ensure the

<sup>1</sup>A system should be designed to be reliable from the start; retrofitting reliability to an existing design is very difficult [30, 35].

above *atomicity* requirement, a *Transaction Commit* protocol should be used; there is approximately thirty years of work on these protocols in the literature.

The best known *Transaction Commit* protocol is the *Two-Phase Commit* protocol.<sup>2</sup> In this protocol, the coordinator sends a *Prepare* message to each RM, which effectively asks if the RM can commit the transaction (i.e. in our case, it asks if the RM can fulfill the scheduling request). If it can commit, the RM sends back a *prepared* message to the coordinator and moves into the prepared state, otherwise it sends an *aborted* message to the coordinator and moves to the aborted state. If the coordinator receives a *prepared* message from all the RMs it sends a *commit* message to all the RMs and they move into the committed state, i.e. the requests are confirmed. Otherwise, if the coordinator receives an *aborted* message, or if one or more RMs fail to respond within a certain time limit, the coordinator sends an *abort* message, and all RMs move into the aborted state, i.e. the requests are abandoned. There is no response from the RMs to either *commit* or *abort*; the RM *must* perform the stated action. In the case where the RM does not receive the *abort* or *commit* message, the message may be lost due to a network failure, it can query the coordinator to find out the outcome.

The problem with the two-phase commit protocol is that it is a blocking protocol. If the coordinator fails after issuing the prepare message then RMs may be left in the prepared state until the coordinator is repaired. Skeen [36] showed that a non-blocking commit protocol requires at least three phases.

One such transaction commit protocol is Gray and Lamport's Paxos Commit Protocol [12], where the coordinator process is replaced with a set of replicated processes, called *Acceptors*. This protocol is based upon Lamport's Paxos Consensus Algorithm.<sup>3</sup>

In the Paxos Consensus algorithm a leader process tries to get a number of Acceptor processes to agree a value. Any one of the Acceptors can act as the leader and any of the Acceptors can replace the leader if it fails. Paxos can still reach consensus if there is more than one leader or if any of the Acceptors or leader processes fail and recover. Messages can be lost, delayed or duplicated and the leader and

<sup>2</sup>There are many descriptions of Two-Phase Commit in the literature. These date back to Gray's paper from 1978 [11, Sec. 5.8.3.3], which cites an otherwise-unpublished 1976 Technical Report by Lamport and Sturgis [31] as the original “elaboration” of the protocol; only the 1979 revision of the latter is available on-line, which circularly cites Gray's 1978 paper. Much later, Gray and Lamport provide a description plus TLA<sup>+</sup> Specifications in [12]. For a concise overview, the reader is referred to the Wikipedia definition [21].

<sup>3</sup>The originally published description of Paxos [28] is notoriously hard to understand; a far simpler formulation can be found in [29]. Both of these papers, and the paper on Paxos Commit [12], may be found at Leslie Lamport's website: <http://research.microsoft.com/users/lamport/pubs/pubs.html>

Acceptor processes can all run at different speeds. Given a set  $2F + 1$  of Acceptors, Paxos continues to make progress as long as  $F + 1$  Acceptors are working; if more than  $F$  Acceptors fail, Paxos will block, but never becomes inconsistent.

In Paxos Commit, the Acceptors engage in a number of separate instances of the Paxos Consensus Algorithm, one instance for the prepared/aborted decision of each Resource Manager. Once the result of the transaction is known—i.e. after all RMs have prepared, after any RM has aborted, or some RM has failed to respond in time—the current leader sends the outcome to the RMs; if these messages fail to arrive, the RMs can query any of the Acceptors for the outcome.

Due to its excellent fault-tolerant capabilities, and also due to the elegance of the algorithm, Paxos Commit was chosen to be the basis for the HARC co-scheduler.

### 3. HARC System Description

The architecture of the HARC co-scheduling system is shown in Figure 1, following the terminology explained in the previous section; to simplify the diagram, the individual Acceptors are not shown. Any odd number of Acceptors can be used; increasing the number of Acceptors increases the reliability of the system, but also increases the number of message needed to complete each transaction.

Co-scheduling with HARC consists of two basic steps. First, the client contacts the RMs of the resources for timetable information, whose representation is described below.<sup>4</sup> Based on an analysis of this information, a set of resources is selected and, in the second step, scheduling requests are presented to an Acceptor for co-scheduling. The timetable information may be approximate, and will certainly be slightly out-of-date (as in a distributed system, messages take a non-negligible time to arrive [27]), and so there is a possibility that the transaction will be aborted. The user polls the Acceptors to discover the outcome of the attempted co-scheduling. The message sequence diagram for both steps is shown in Figure 2. As will be seen, *all* messages in the system, including the user’s co-scheduling request, are idempotent; if, at any point, the user does not receive a reply to a request, it is *always* safe to send the request again.

#### 3.1. Transparent Acceptors

By using the Paxos Commit protocol, the contents of the messages describing the reservations requested by the user are only of concern to the RMs; the Acceptors only need to

<sup>4</sup>It is assumed that there is a separate step prior to this, where the user discovers and selects suitable resources, by the use of registries and/or resource brokers.

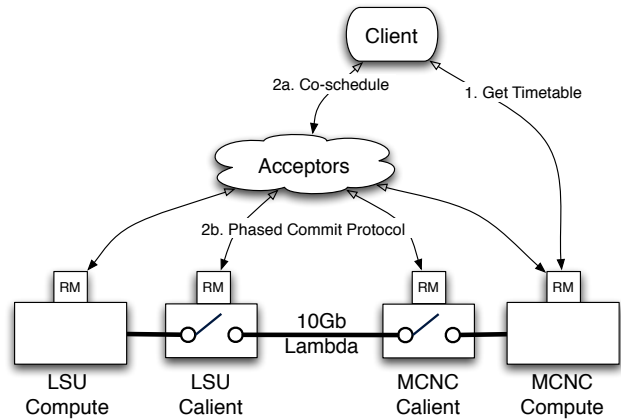


Figure 1. The HARC architecture, showing the relationship between the client, the Acceptors, and the Resource Managers (RMs).

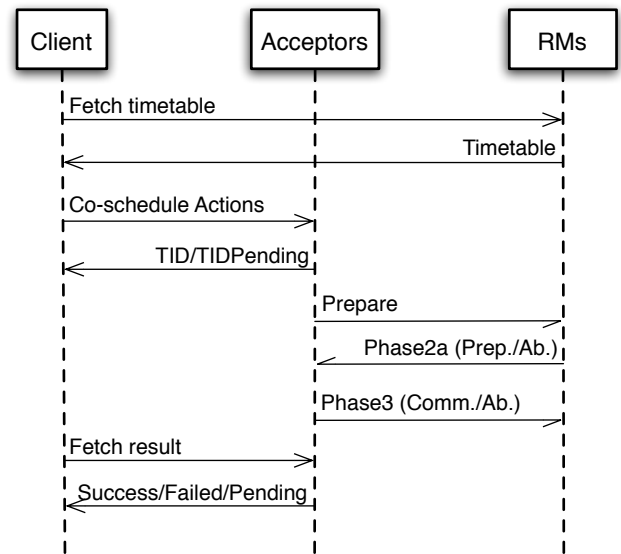


Figure 2. Simplified HARC Message Sequence Diagram (client polling and Acceptor-to-Acceptor Paxos messages are omitted).

know the endpoints of the RMs. All other message content is regarded as opaque by the Acceptors; it could even be encrypted if desired, e.g. using XML Encryption [23].

The obvious corollary of this is that to allow the co-scheduling of a new type of resource only requires that a new Resource Manager is written; absolutely no modification of Acceptor code is required.<sup>5</sup>

The detailed design of HARC is presented below, as a coherent whole. The authors indicate which elements of the design are integral to the Acceptors, and which are integral to the Resource Managers; apart from the choice of Transfer Layer, rarely is an element integral to both.

### 3.2. Message Format and Transfer Layer

All messages sent between the client and HARC, are written in XML, as are all messages sent inside HARC between Acceptors and RMs. Beyond this, the Transfer Protocol used to transmit the messages is the concern of the implementation. The purpose of all XML messages in the system can be distinguished by the tag name of the root element of the message; no additional discriminator, e.g. method name, is required. The transfer layer must be capable of supporting the secure transfer of messages; encryption need not be supported, but it must be possible to authenticate the source of messages, and to ensure that messages cannot be undetectably tampered with.<sup>6</sup> For identification purposes, it is preferable that a security system that can map users to a reasonably unique string is used, e.g. signing by X509 Certificates, which can map users to a Distinguished Name.

Acceptors and RMs both need to pass service endpoints around. In all messages, this is done using an `Endpoint` element. The chosen Transfer Layer should use an attribute, `type`, to distinguish the Endpoint's type. A transfer-specific element (of any type) precisely specifying the endpoint should be embedded within the Endpoint element. An example endpoint for a REST-ful implementation is shown in Figure 3. There is no requirement to further describe the endpoints, e.g. how additional parameters are communicated; this is left to convention within the implementation.

Clearly, it is a matter for the implementation as to what Transfer Layer(s) it wishes to support; no protocols are mandatory.

In any implementation of HARC, it should be possible to keep the number of modules which are transfer-layer aware to a minimum; the vast majority of the software will just need to deal with XML messages, and should not care how they are received and sent.

<sup>5</sup>This was observed "in the field" when adding the facility to co-schedule the Calient switches; the Acceptor code was not touched.

<sup>6</sup>The requirement that message cannot be undetectably tampered with comes directly from the Paxos Consensus Algorithm [29, 12]. This can be easily achieved with signing and encryption techniques, e.g. by using either HTTPS, or WS-Security [32].

### 3.3. Timetable Retrieval

Timetables are used in HARC to communicate about the availability of resources. Timetables are requested from RMs by clients; the Acceptors are not involved in the timetable system. Timetables show what resource is free, rather than providing information about other jobs in the system which are consuming resources; this allows a higher degree of privacy for the resource owner, and simplifies the task of the client. On the RM side, the protocol is simpler to implement than one where a client asks what time a particular amount of resource is free.

For timetabling purposes, a resource is composed of a number of subresources, each of which has its own timetable. For example, a single-system image compute resource could publish two subresource timetables representing available processors and memory.<sup>7</sup> HARC currently supports two separate classes of subresource: quantitative subresources, where the timetable specifies the amount of the subresource which is available at a given time; and atomic subresources, where the timetable specifies whether the subresource is available or not. The returned timetable information consists of

- a description of the resource,
- a specification of the range which the timetable information covers, then, for each subresource:
  - a brief description; and
  - the timetable information.

Figure 4 shows the XML returned from a timetable request for CCT's SGI Altix machine. The machine has 32 processors, and 32 GB of main memory. The timetable returned covers an eight hour period from the time that the request was made. Initially, 4 processors and 8 GB are free; from 5pm, 12 processors and 12 GB of memory are available.<sup>8</sup> Figure 5 shows the simplified XML returned from a timetable request for CCT's Calient Diamondwave Switch; only two of the 16 actual ports are shown. Each port has an input and output which are considered to be separate subresources (currently, there is no way to show that these are linked in any way, other than by the convention used in the name). A port is either free or it is not. Here, both ports are free (input and output) for the eight hour period covered; unavailability would be indicated by the absence of the `Available` element in a timetable entry.

<sup>7</sup>In a single system image resource, the relationship between available memory and available processors is not necessarily obvious. A 32-processor machine might have only two processors available, but 90% of the machine's memory.

<sup>8</sup>It should be noted that the *content* of the message is dependent on not only the resource being described, but also upon conventions used by the implementation, e.g. the use of the strings "processors" and "memory".

```

<Endpoint type="REST">
  <RESTEndpoint>
    http://up.cct.lsu.edu:9292/lsu-calient-rm
  </RESTEndpoint>
</Endpoint>

```

**Figure 3. XML Snippet describing the endpoint of a RESTful service.**

```

<?xml version="1.0" encoding="utf-8" ?>
<Resource>
  <Name>santaka.cct.lsu.edu</Name>
  <Type>compute</Type>
  <Description>LSU's SGI Altix Machine</Description>
  <Timetable>
    <Start>2006-01-09T16:56:37-06:00</Start>
    <End>2006-01-10T00:56:37-06:00</End>
  </Timetable>
  <SubResources>
    <Quantitative type="processors">
      <Name>processors</Name>
      <Units>CPU</Units>
      <Quantity>32</Quantity>
      <Timetable>
        <Entry>
          <From>2006-01-09T16:56:37-06:00</From>
          <Quantity>4</Quantity>
        </Entry>
        <Entry>
          <From>2006-01-09T17:00:00-06:00</From>
          <Quantity>12</Quantity>
        </Entry>
      </Timetable>
    </Quantitative>
    <Quantitative type="memory">
      <Name>memory</Name>
      <Units>GB</Units>
      <Quantity>32</Quantity>
      <Timetable>
        <Entry>
          <From>2006-01-09T16:56:37-06:00</From>
          <Quantity>8</Quantity>
        </Entry>
        <Entry>
          <From>2006-01-09T17:00:00-06:00</From>
          <Quantity>21</Quantity>
        </Entry>
      </Timetable>
    </Quantitative>
  </SubResources>
</Resource>

```

**Figure 4. Example timetable for a Compute Resource.**

```

<?xml version="1.0" encoding="utf-8" ?>
<Resource>
  <Name>calient.ocs.lsu.edu</Name>
  <Type>network</Type>
  <Description>LSU's Calient Switch</Description>
  <Timetable>
    <Start>2006-01-12T00:00:00-06:00</Start>
    <End>2006-01-12T08:00:00-06:00</End>
  </Timetable>
  <SubResources>
    <Atomic type="port">
      <Name>0.11a.1.OUT</Name>
      <Alias>nlr-10gb.OUT</Alias>
      <Timetable>
        <Entry>
          <From>2006-01-12T00:00:00-06:00</From>
          <Available />
        </Entry>
      </Timetable>
    </Atomic>
    <Atomic type="port">
      <Name>0.11a.1.IN</Name>
      <Alias>nlr-10gb.IN</Alias>
      <Timetable>
        <Entry>
          <From>2006-01-12T00:00:00-06:00</From>
          <Available />
        </Entry>
      </Timetable>
    </Atomic>
    <Atomic type="port">
      <Name>0.11b.1.IN</Name>
      <Alias>santaka.IN</Alias>
      <Timetable>
        <Entry>
          <From>2006-01-12T00:00:00-06:00</From>
          <Available />
        </Entry>
      </Timetable>
    </Atomic>
    <Atomic type="port">
      <Name>0.11b.1.OUT</Name>
      <Alias>santaka.OUT</Alias>
      <Timetable>
        <Entry>
          <From>2006-01-12T00:00:00-06:00</From>
          <Available />
        </Entry>
      </Timetable>
    </Atomic>
  </SubResources>
</Resource>

```

**Figure 5. Example timetable for a Calient Diamondwave Switch.**

Requesting timetable information is very simple; often, no parameters will be required. Indeed, the request could be implemented by a number of transfer protocols without the need for an XML document, and so it is left to implementation to determine the nature of this request. However, in terms of querying, the following requirements must be satisfied.

- Clients **MUST** be able to request timetables for specific intervals, by supplying an optional start-time and/or end-time; RMs are under no obligation to return information for the requested interval (the client may request timetable information further into the future than the service is willing to expose and/or take bookings for) but **MUST** accurately indicate the start and end times of the information being returned. The RM can provide as complete or simplified a picture of the available resources as it chooses.
- An implementation **MUST** provide a mechanism for the the user to obtain the descriptions of the subresources, without any timetable information.
- An implementation **MAY** support the querying of timetable information for specific subresource(s).

### 3.4. Co-scheduling the Requests

To co-schedule a set of resources, the user sends an XML document containing a list of `Make` elements; it will be seen later that `Make` is just one of a number of supported *Actions*. The set of *Actions* is given a User Identifier string, or `UID`, which should be unique to that user. One `Make` element is sent for each required resource, specifying three things:

**What** – a `Work` element, describing the resources which are required;

**Where** – a `Resource` element, pointing to the Resource Manager of the resource; and

**When** – a `Schedule` element, describing when the resources are needed.

An example message is shown in Figure 6, requesting a single CPU and 20GB of memory on Santaka (specified using JSDL 1.0 [2]), plus the connection of Santaka’s 10Gb network port to the NLR through LSU’s Calient Diamondwave Switch. Both resources are requested for a duration of four hours starting from 21:45 (specified in UTC).

The format of the *Actions* message is partly the concern of the *Acceptors*, which split the message into the individual *Actions* for use in the *Prepare* messages that are sent to the *RMs*. The *Acceptors* also use the `Resource` elements, which provide them with the location of the *RMs*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Actions uid="main1136928630068">
  <Make filename="net22">
    <Resource>
      <Name>calient.ocs.lsu.edu</Name>
      <Endpoint type="REST">
        <RESTEndpoint>http://up.cct.lsu.edu:9292/1
su-calient-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule>
      <StartAt>2006-01-10T21:45:00Z</StartAt>
    </Schedule>
    <Work>
      <Duration>14400</Duration>
      <ConnectionSet>
        <Connection bidirectional="true">
          <From>nlr-10gb</From><To>santaka</To>
        </Connection>
      </ConnectionSet>
    </Work>
  </Make>
  <Make filename="input.santaka.igrid">
    <Resource>
      <Name>santaka.cct.lsu.edu</Name>
      <Endpoint type="REST">
        <RESTEndpoint>http://santaka.cct.lsu.edu:9
191/santaka-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule>
      <StartAt>2006-01-10T21:45:00Z</StartAt>
    </Schedule>
    <Work>
      <JobDefinition xmlns="http://schemas.ggf.org
g/jsdl/2005/06/jsdl">
        <JobDescription>
          <Application>
            <POSIXApplication xmlns="http://schema
s.ggf.org/jsdl/2005/06/jsdl-posix">
              <WallTimeLimit>14400</WallTimeLimit>
            </POSIXApplication>
          </Application>
          <Resources>
            <TotalCPUCount>
              <Exact>1</Exact>
            </TotalCPUCount>
            <TotalPhysicalMemory>
              <LowerBoundedRange>21474836480</Lowe
rBoundedRange>
            </TotalPhysicalMemory>
          </Resources>
        </JobDescription>
      </JobDefinition>
    </Work>
  </Make>
</Actions>
```

**Figure 6. Example co-scheduling request for a compute resource and a network switch.**

For extensibility sake, any implementation of the Acceptors must copy all elements inside the Action elements into the Prepare message, except for the Resource element. One possible use of this feature will be seen in Section 3.5.

The user can send this message to any Acceptor. The Acceptors conduct an instance of the Paxos Consensus algorithm, to agree a Transaction ID (or TID) (and also the associated Actions, which form part of the value of the consensus's outcome value), with the Acceptor which the user contacted acting as leader and proposing the value of the TID. If this instance of Paxos is resolved quickly, the TID can be sent straight back to the user, in an XML document with a single TID element, containing the TID as a string; otherwise a TIDPending element is sent back. In the latter case, or if for some reason the user gets no reply, the same message is sent again, either to the same Acceptor, or to another. In the case that the original message was lost, the instance of Paxos will be started. In the case that the instance of Paxos was started, the result from that first instance will be found (the Acceptors use the identity of the user's identity plus the UID as a key for looking up the Paxos instance). The user retries until a TID is obtained.

Following this initial round of Paxos, the co-scheduling begins. Whichever Acceptor is acting as leader dispatches the Prepare messages to the RMs.<sup>9</sup> One Prepare message is sent for each Make element received, containing a list of Acceptors, the TID, the identity of the user, and the What and When of the proposed work (plus any additional elements, as stated above).

The RMs consider the work, and respond by sending their Prepared/Aborted response to all of the Acceptors (in Paxos terminology, this is a *Phase 2a* message). The Acceptors agree the Prepared/Aborted decision for each RM, each RM's decision having its own instance of Paxos. In the case where an RM's decision only reached a subset of the Acceptors, the Paxos algorithm will propagate this to the other Acceptors, and it will be agreed upon. In the case where an RM's decision was not received by any Acceptor (or if it was never sent for some other reason, e.g. if the RM was down), then some Acceptor will (after a delay) propose a new value of "Aborted" for that instance of Paxos, which will be agreed upon.

Once all instances of Paxos have resolved to "Prepared", or after any one instance produces "Aborted", then the Acceptors will know the result of the transaction. The final decision is sent to all RMs; if they do not receive this—e.g. if the message is lost or an RM is temporarily unavailable—then they may query the result from any Acceptor, using the TID. The user polls the Acceptors for the result of the transaction; if the result is not ready, this will contain an empty ActionsPending element, and the user should retry. As

<sup>9</sup>For the sake of brevity, the structure of this message is not described, nor are of any of the other Paxos messages.

with the timetable request, the details of this request are left to the implementation.

In the case of success, an XML document with a root ActionsSucceeded element is returned, like the example in Figure 7. For each Make element sent, there will be a corresponding Make element in the reply (these are in the same order as in the request, and are also numbered by actionId attribute), containing an Ident element containing the identifier of the reservation, and also copies of the Resource and Schedule elements sent; future RMs may support the refinement of the Schedule element, where a possible range of times is specified, and where the RM selects a more precise value.

In the failure case, an ActionsFailed element is sent, having one Make element for each failed Action, containing the Resource element, and also a Reason element containing either a string returned by the RM stating why it aborted, or a string stating that the RM failed to respond, if this is the case. The number in the actionId attribute can be used to map this back to the corresponding Make element.

In both the success and failure case, the format of this message is a combination of information from the RMs and the Acceptors. In the success case, the Ident elements or, in the failure case, the Reason elements are simply copied by the Acceptors (they form part of the outcome value of the instances of Paxos).

It must be pointed out that the Paxos Consensus algorithm requires that, upon failing, Acceptors can be repaired to the state they were in before, implying that the state of all ongoing instances of Paxos must be written to stable storage, e.g. to a database, as the protocol proceeds. By doing this, upon restart, the Acceptor still knows the state of each Paxos instance.

### 3.5. Security Model

As stated in the requirements for the Transfer Protocol, it must be possible for entities to identify each other: RMs and Acceptors need to authenticate (and authorize) users, and RMs need to authenticate and authorize Acceptors. As Acceptors use the user's identity plus the UID to uniquely identify instances of Paxos, username and password schemes are unlikely to be appropriate. (Indeed the system was designed with the use of PKI X509 Certificates in mind.)

In some environments, Resource Managers will be able to trust the Acceptors (which they can authenticate and authorize) to relay the user's identity correctly. In situations where this is inadequate, the user could supply an XML signature [3] for the Work element, and place this inside a new Signature element in the Make element. As the Acceptors simply copy all the elements in the Action elements

except for the `Endpoint` element, this would not require any modification to the Acceptors. Of course the RMs needing the enhance security would have the additional requirement of verifying the signature.

### 3.6. Reliability of HARC

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 [13, Sec. 3] are used. 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 [13, 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 [13, 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:

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

This gives us a MTTF of the whole system of:

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

Lets conservatively assume that the MTTF of a single Acceptor is 48 hours, and that following a failure, the MTTR is 2 hours; this should be sufficient to include unavailability due to network outages. Then, in a deployment with seven Acceptors (i.e.  $F = 3$ ), results in a MTTF of over 650 days.

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 just five, or even three Acceptors.

While these numbers are impressive, the authors 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. Although these failures are not modeled in the above equations, the value of being able to co-schedule across a distributed system during such failures is highly questionable.

Finally, the authors note that as HARC is based upon the Paxos Consensus algorithm, it will *never* become inconsistent. In the case where too many Acceptors fail, the system will simply block until sufficient Acceptors are repaired to provide a majority.

## 4. Make, Modify, Move and Cancel

Above, it was shown how a user would use HARC to co-schedule a compute job along with a network connection by co-scheduling two Make Actions. HARC also supports three other Actions for controlling the reservations created with Make: Modify, Move and Cancel.

As with Make, the specification of the other Actions is simple. As shown above, an `Ident` element, containing a Reservation ID, is returned for every successful Make Action; a copy of this element is used in Move, Modify and Cancel to identify the reservation that is to be manipulated. To use these other Actions, the following must be specified:

**Move** – the `Ident` element plus **When** – a new `Schedule` element;

**Modify** – the `Ident` element plus **What** – a new `Work` element;

**Cancel** – just the `Ident` element.

With HARC, any combination of the Make, Modify, Move and Cancel Actions can be combined into an atomic co-scheduling transaction. This simple, yet powerful idea makes HARC well suited to the scheduling of large scientific workflows.

### 4.1. Co-scheduling Workflows with HARC

The use of workflows to express and automate complex scientific processes is a growing research topic in scientific computing. These workflows are often modeled as a Directed Acyclic Graph, or DAG, where the nodes of the graph represent computational tasks, and the edges of the graph are dependences between the tasks, which constrain the order in which the tasks may be executed. Two large projects

dealing with workflows, Pegasus [7] and LEAD [10] represent workflows in this form.

The common approach when scheduling workflows is to submit tasks for execution once all tasks they are dependent upon are completed. At the point they are submitted, the tasks must wait for resources to become available, typically waiting in some sort of work queue. Due to the unpredictability of the wait time in the queue, it is not possible to safely submit jobs ahead of their possible start time.<sup>10</sup> This approach to workflow execution is ideal in the situation where resources are readily available to execute the components of the workflow, or where the user does not care about the overall turnaround time for the workflow execution. However, the execution time for the overall workflow can vary dramatically. For example, if the average queuing time for the resources available to a user increases by one hour, then the time taken to execute a three-stage workflow, where each step is dependent on the last, would increase by three hours. Such variations in resource usage are common in reality, as are workflows consisting of many more parts.

The scheduling of workflows with advance reservation should provide better turnaround time for the entire workflow, as the later tasks—the ones with dependences—can be scheduled much earlier. The client should be able to get a better placement of the components. In addition, scheduling in this way will provide the end user with a far more accurate indication of how long the workflow will take to execute. Because HARC allows clients to atomically combine different Actions on arbitrary reservations, it is trivial to schedule large workflows in a piecewise fashion; there is no grouping of reservations into a single indivisible entity. It is also easy to cancel or re-schedule part of a workflow.

Figure 8 shows the scheduling and re-scheduling of a simple example workflow 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. The client creates an initial schedule for the workflow, the first part running on Machine A for 4 hours, the second part on Machine B for 8 hours, and the third part on Machine C for 4 hours. While the second part is running, a monitoring tool discovers that this computation is running slower than expected, and will take between 10 and 11 hours to complete; the 8 allocated hours are insufficient. First, the client tries to extend the allocated time for the second part and delay the execution of the third part, by co-scheduling a Modify for Machine B and a Move for Machine C. Unfortunately, the RM for Machine C does not support Move, causing the transaction to be aborted; the initial schedule remains. The client then

<sup>10</sup>This is the case with most batch queue systems running on supercomputers. There is always a chance that your job will start far quicker than expected, based on an inspection of the work queue: the job may be prioritised; may fill a gap in the schedule better than other work in the queue; or the work ahead of the job in the queue may complete ahead of time or be withdrawn.

tries a new strategy: instead of delaying the third part, a Cancel for Machine C is co-scheduled with the same Modify for Machine B, together with a Make Action for Machine X, which will be now be used for executing the third part.<sup>11</sup>

## 4.2. Caveat

It is only possible to support Move and Modify if:

1. the underlying scheduler supports the modification of existing reservations;
2. two-phase commit on these actions is also supported.

This is not the case with Make and Cancel, which can be supported without 2-phase commit, in the following way. When Preparing a Make, the RM tries to make the reservation for the user, the result determining its Prepared/Aborted response; if the transaction is then Aborted, the reservation will be deleted. When preparing a Cancel, the RM does nothing, and returns “Prepared”;<sup>12</sup> if the transaction is committed, then the reservation will be deleted. But there is no analog for Move or Modify; if the change is enacted by the RM during the first phase, then there is no guarantee that it can reverse the change in the second phase if the transaction is canceled; if the RM waits until receiving a Commit, there is no guarantee that the change will be possible at all.

The authors currently know of no scheduler which supports two-phase commit on the modification of existing reservations. The most promising scheduler for future investigation is Moab from Cluster Resources [14, 24], which allows the modification of existing and even current reservations, and supports two-phase commit for the creation of a reservation.

## 5. Resource Managers for HARC

To date, two classes of Resource Manager have been designed for HARC; one for parallel compute resources, the other for network switches. These are described briefly below.

Both classes of RMs expect a single `xsd:datetime` to be present in the `StartAt` element contained in the `Schedule` element.

<sup>11</sup>If this second re-scheduling strategy succeeds, the client will need to make other adjustments to ensure that any file transfers to/from Machine C are redirected to Machine X, and ensure that the job that is to run on Machine X gets submitted to the new reservation. These processes are outside the scope of HARC.

<sup>12</sup>The authors know of no circumstances where canceling a reservation is not possible.

## 5.1. Compute Resource RMs

A class of RMs is specified which can be used to create and manipulate reservations on high performance computers that are under the control of a batch queue system supporting advance reservations. The RMs are concerned with the description of the resource requirements and the creation of the reservation, but not with the submission of jobs to the reservations; this is left for the client to arrange with external mechanisms.<sup>13</sup>

These RMs will expect the supplied `Work` element to contain a single `JobDefinition` element as defined in the JSDL 1.0 Standard [2], which describes the resources required by the client. It is further mandated that:

- The `POSIX WallTimeLimit` element [2, Sec. 8.1.9], specifying the length of the reservation in seconds, **MUST** be present;
- The `TotalCPUCount` element [2, Sec. 6.4.24], specifying the number of processors required, **MUST** be present and be specified using `Exact`; and
- The `TotalPhysicalMemory` element [2, Sec. 6.4.25], specifying the memory required by the job, **MAY** be present, and if present should be specified as a `LowerBoundedRange`.
- All other elements are optional, and **MAY** be ignored.

An example is shown in the second `Make` element in Figure 6.

## 5.2. Network Switch Resource Managers

Network Switches are modeled as devices consisting of a number of *Ports*, to which devices are attached; the Switch can then make a number of *Connections* between the various Ports. Clients request a set of connections they wish to be made for a certain length of time (starting from the `Schedule` element's `StartAt` time). The RM checks that the request does not interfere with any other timetabled connections.

The RMs will accept a `Work` element containing: a `ConnectionSet` element, which will contain a list of `Connection` elements, specifying the connections; and a `Duration` element, which specifies the length of time that these are required for, in seconds. The `Connection` elements are further specified as follows:

- The ports to be connected are named in `From` and `To` elements (a list of the ports is available through the timetable interface);

<sup>13</sup>It would be possible to design another class of RMs which took a more complete specification of the work, and which then submitted the work to the reservation, in a single atomic step.

- Whether a connection is bi- or uni-directional can be specified by the `bidirectional` attribute on the `Connection` element, and should be set to a valid `xsd:boolean` value; and
- Different implementations may support switch-specific extensions.

A simple example can be seen in the first `Make` element in Figure 6.

## 6. HARC/1: A First Implementation

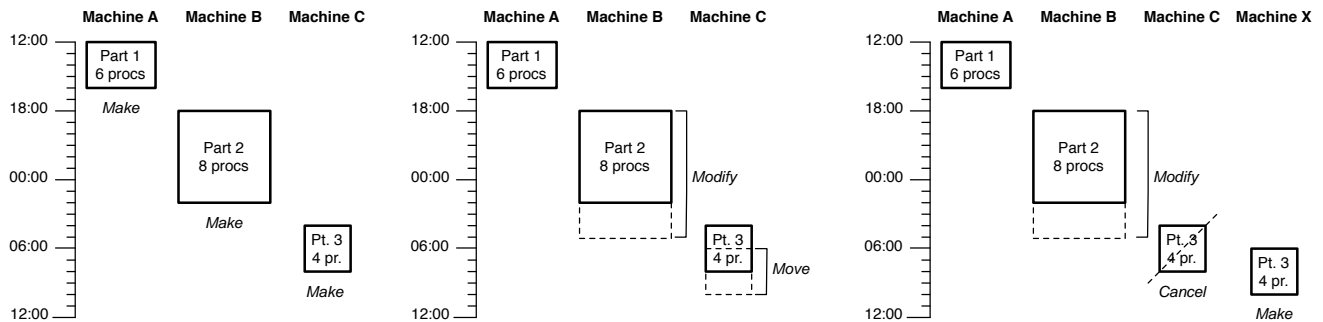
The design presented in Sections 3 has been implemented, together with Resource Managers that follow the designs in Section 5. All parts of the implementation are open source under a BSD-style license, and are available for download [15]. Some of the key features of the implementation—and its current limitations—are described, as are the authors' preliminary experiences using the software in the field.

### 6.1. Transfer Layer

Plain XML over HTTP/HTTPS has been opted for, rather than SOAP-based Web Services. Messages are sent using HTTP POST, and information is requested with HTTP GET. This reduces the complexity of the Acceptor software stack, and is adequate for implementing HARC securely.<sup>14</sup> This approach is often characterized as the Representational State Transfer (REST) approach to building services [20], which evolved from the architectural principals proposed by Fielding [8, Sec. 5]; however, we do not currently provide any of the self-description and linking which are typically part of a REST-ful service. In particular, convention is used to determine the URLs to GET and POST to, i.e. a schema, rather than have the service provide links to these URLs which the Acceptors/RMs would follow.

Table 1 shows the URL extensions that the HARC/1 RMs and Acceptors use for communicating; these extensions should be appended to the component's endpoint URL. So, for example, the XML document in Figure 5 was obtained from an HTTP GET on `http://up.cct.lsu.edu:9292/lsu-calient-rm/timetable?start=2006-01-12`. The result of co-scheduling a set of Actions can be obtained by performing an HTTP GET on a URL such as: `http://vizws00.cct.lsu.edu:8080/accl/result?tid=paxos-ident-4`.

<sup>14</sup>This decision provided an additional advantage that during development, in that specific clients to the services in HARC were rarely required; Web browsers, `curl` and `wget` were usually sufficient.



**Figure 8. The scheduling and re-scheduling of a simple workflow. The creation of an initial schedule is shown on the left; the center picture shows the first attempt to re-schedule, extending the reservation on Machine B, and moving the reservation on Machine C; the final picture shows a second attempt to re-schedule, again extending the reservation on Machine B, but canceling the reservation on Machine C, and replacing this with a new reservation on Machine X.**

Component	Operation	URL Extension	Description
RMs	GET	/description	Fetches the subresource descriptions without timetables
	GET	/timetable	Fetches the timetable information for the default range
	GET	/timetable?start=2006-01-11&end=2006-01-13	Fetches the timetable from the 11th to 13th of January 2006
	POST	/paxosRequest*	“Prepare” some Actions for the user
Acceptors	POST	/doActions	Request the co-scheduling of some Actions
	POST	/paxos*	Send a Phase <i>X</i> Paxos message
	GET	/result?tid=paxos-ident-4	Fetch the result of a given TID
	GET	/result?uid=jons-user-ident	Fetch the result of a given UID
	GET	/getAcceptors	Fetch the list of Acceptors

**Table 1. Supported URL extensions in HARC/1; asterisked entries (\*) are for internal use only by the HARC components.**

## 6.2. Acceptors

The Acceptors were coded in Java, using Version 1.5 in order to get the new timeout facility on `java.net.HttpURLConnection`; no other Java 1.5 specific features were used. The Acceptors are packaged as a servlet, for deployment in Jakarta Tomcat.

The code is clean and extensible. The Protocol-specific code is kept in the class `RESTEndpoint` (a subclass of `Endpoint`). `REST` Endpoints are also assumed in `AcceptorServlet`, where all the servlet-specific code resides, and where the URL schema shown in Table 1 is encoded. It should be very easy to provide other wrappers for the code, e.g. a SOAP-based Web Service, by defining a new `Endpoint` subclass, and providing a new implementation of the `PaxosCommunicator` interface (which `AcceptorServlet` implements).

## 6.3. Resource Managers

Two Resource Managers have been written so far, using Perl 5.8. The first RM is a compute resource Resource Manager, specialized for PBSPro scheduler. This follows the design presented in Section 5.1. Due to the issues mentioned in Section 4.2, only Make and Cancel are supported. In addition, the current implementation does not obtain realistic timetable information from the scheduler.

The second RM is for Calient Diamondwave optical network switches and follows the design described in Section 5.2. As the switch does not have a native scheduler, a simple, reusable timetable-based scheduler was written for resources consisting of a set of Atomic subresources. The RM should be run on a workstation near to the switch. The switch is controlled by sending TL1 commands<sup>15</sup> from this machine to a port on an IP address hosted by the switch's control plane. The TL1 commands are scheduled on the machine running the RM using the standard UNIX `at` command. Currently only Make and Cancel are supported; the timetable facility needs to be extended to support Move and Modify. It should be noted that the timetable-based scheduler will only function properly if no other clients "go around it" and interact with the resource directly (all schedulers rely on this). If desired, the scheduler could be run on a subset or partition of the resource; this might be more practical.

The two RMs have been written in a modular fashion. The common core of the RMs use `XML::Xerces` for parsing and constructing XML messages; `POE` is used to provide a `HTTPD` listener. Creating additional RMs should be

<sup>15</sup>TL1, or Transaction Language 1, is a telecommunications management protocol. It is a standard, and is used by multiple vendors. Those interested in learning more should visit <http://www.tl1.com/>

a simple process. For example, to create a new RM supporting Make/Cancel Actions for batch queue system X (which supports reservations), and which accepts Work definitions in JSDL 1.0, should only require that a new module is created to replace `PBS.Batch.pm`; this only contains 100 lines of code.

## 6.4. Client APIs

When planning the construction of a portal interface to the co-scheduler, the authors realized that much of the existing Java client code used in the iGrid demo would need to be copied and modified in the portlet. To avoid this, the client code was completely refactored to create APIs for accessing the Timetable and Co-scheduling functionality through reusable code. The command line interface and portal shown at Supercomputing '05 both used this API. These APIs, which are available for download [15], are explained here briefly.

The Timetable API allows the user to:

- discover the resource description and subresources available from an RM;
- compose resource requirements simply, and query the timetable to see when these are free;
- overlay the results from multiple RMs to see when *all* resource requirements are met.

The timetable API handles all network communications. Timetables from particular RMs are cached automatically. The API refreshes its copy after a given (configurable) time, or when instructed to directly through the API.

The co-scheduling API accepts simple descriptions of work, specified as attribute-value pairs. These are checked and converted into the appropriate XML form, according to what the HARC/1 RMs accept. The addresses of the Acceptors are specified in a properties file, which is loaded when the API is initialized (not shown). The co-scheduling process is enacted by a single call, which encapsulates the complexity of querying the multiple Acceptors. An example is shown in Figure 9.

## 6.5. Early Experiences

HARC/1 was used to co-schedule ten compute jobs, and two Calient DiamondWave switches as part of iGrid 2005 demonstration US127, entitled "Interactive Visualization across LONI" [22]. The experiment was repeated during the Supercomputing'05 exhibition. During rehearsals and the demonstrations, the co-scheduling process, involving twelve simultaneous instances of Paxos, took between 10 and 20 seconds, using a configuration of three Acceptors deployed on workstations on the same subnet.

## 6.6. Caveats

There are two main features still to be implemented in HARC/1 before it can be regarded as a complete implementation of HARC. First, the Acceptors do not store the state of the rounds of Paxos in stable storage. Second, the security model presented in Section 3.5 has not been implemented; plain HTTP is being used.

## 7. Related Work

Architecturally, the closest system to HARC described in the literature is GARA [9, 34], which can interface to a broad class of Resource Managers, and so can reserve network bandwidth in addition to compute resources. GARA does not provide a co-scheduling facility, but gives a good framework for building one; Roy sketches a design for one in [33, Sec. 7.3.2]. GARA was built for Globus Toolkit 1 and 2 and used RSL to describe the resource requirements. GARA is no longer being developed.

Generic Universal Remote, or GUR [37] 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 less up-front deployment of software than HARC, and as such will be capable of running in more environments. However, all configuration details for each resource (e.g. the type of batch queue software being used, etc.) are held on the client; with HARC, these configuration details are kept in the Resource Managers, and the clients need make no changes to use new resources. It is hard to see how GUR can be extended to cover resources that the user does not directly log into, e.g. network devices.

The Grid Resource Allocation and Agreement Protocol Working Group (GRAAP-WG) within the Global Grid Forum are also working on the problem of reservations and co-allocation in Grids. At the time of writing, they are completing work on WS-Agreement [1]. WS-Agreement contains several components: a document format, an agreement template discovery system, a protocol for creating agreements, and a monitoring system, some of which require the use of the emerging Web-Services Resource Framework. [17] Like its predecessor, SNAP [6], the protocol component of WS-Agreement is a one-phase protocol. It has been proposed that such protocols can be applied to the co-scheduling problem. However, even if the lack of phased-commit model is accepted, there is still the problem that the RM cannot tell the difference between a real reservation and a reservation which is being tentatively made as part of a co-scheduling attempt.

An alternative protocol for co-scheduling with advance reservations has been proposed in [26]. This protocol is not a transaction commit protocol, and makes no guaran-

tee about atomicity when co-scheduling; resource providers are permitted to back out from a booking at any point. This gives more flexibility to the resource providers, while giving less assurances to the user. It is not clear how suitable this system is for distributed applications requiring co-scheduling, where components are typically closely coupled; one resource defaulting on a reservation would not only result in a rebooking, but would possibly also involve the other co-scheduled jobs being altered to know about the replaced resource.

WS GRAM in Globus Toolkit 4 [16] contains support for multi-site jobs (replacing the Globus Toolkit 2 DUROC component [5]). However, this is not intended to address co-scheduling as defined here; it simply allows the user to submit multiple jobs to multiple sites as part of a single work unit. This separate, but linked process is referred to as *co-allocation* in the literature.

## 8. Conclusions and Future Work

This paper has described HARC, a readily extensible co-scheduling system which can work with any type of resource controlled by a scheduler permitting advance reservations. Unlike existing systems, HARC is fault-tolerant. HARC supports four Actions for manipulating reservations: Make, Modify, Move and Cancel. The ability for HARC to combine arbitrary combinations of the Actions in a single transaction makes HARC an excellent candidate back-end system for the iterative (re-)scheduling of large, complex scientific workflows. An early implementation HARC/1 has been built, and successfully tested during live experiments at iGrid 2005 and Supercomputing 2005.

In addition to completing and properly documenting the HARC/1 implementation, the authors are seeking to establish and support a number of deployments of the software. Ideally, the construction of additional Resource Manager components could be undertaken by the wider community, thus building a rich set of resources which HARC could co-schedule. In addition to compute and network resources, the authors wish to expand HARC's capability into booking rooms, and also entries in people's diaries. The authors are particularly interested in the development of RMs than can support the Move and Modify Actions.

## 9. Acknowledgments

This work is supported in part through NSF Award #0509465, "EnLIGHTened Computing".

## References

- [1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke,

- and M. Xu. Web Services Agreement Specification (WS-Agreement). DRAFT GGF Recommendation in public comment period, September 2005. <https://forge.gridforum.org/projects/graap-wg>.
- [2] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, and D. Pulsipher. Global Grid Forum Recommendation GFD-R-P.056: Job Submission Description Language (JSDL) Specification, Version 1.0, November 2005.
- [3] M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. W3C Recommendation: XML-Signature Syntax and Processing, February 2002. <http://www.w3.org/TR/xmlsig-core/>.
- [4] R. J. Blake, P. V. Coveney, P. Clarke, and S. M. Pickles. The teragryoid experiment—supercomputing 2003. *Scientific Computing*, 13(1):1–17, 2005.
- [5] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *The Eighth International Symposium on High Performance Distributed Computing*, pages 219–228. IEEE Computer Society Press, 1999.
- [6] K. Czajkowski, I. T. Foster, C. Kesselman, V. Sander, and S. Tuecke. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In D. G. Feitelson, L. Rudolph, and U. Schwiiegelshohn, editors, *8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2002), Revised Papers*, volume 2537 of *Lecture Notes in Computer Science*, pages 153–183. Springer Verlag, 2002.
- [7] 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 Journal*, to appear 2006.
- [8] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University Of California, Irvine, 2000.
- [9] 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.
- [10] D. Gannon, J. Alameda, O. Chipara, M. Christie, V. Dukle, L. Fang, M. Farrellee, G. Kandaswamy, D. Kodeboyina, S. Krishnan, C. Moad, M. Pierce, B. Plale, A. Rossi, Y. Simmhan, A. Sarangi, A. Slominski, S. Shirasuna, and T. Thomas. Building grid portal applications from a web service component architecture. *Proceedings of the IEEE*, 93(3):551–563, 2005.
- [11] J. Gray. Notes on data base operating systems. In M. J. Flynn, J. N. Gray, A. K. Jones, K. L. H. Opderbeck, G. J. Popek, B. Randell, J. H. Saltzer, and H. R. Wehle, editors, *Operating Systems: An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. Springer-Verlag, 1978.
- [12] 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](http://research.microsoft.com/research/pubs/view.aspx?tr_id=701).
- [13] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.
- [14] <http://clusterresources.com/moabdocs>. Moab workload manager administrator's guide [online].
- [15] <http://www.cct.lsu.edu/personal/maclaren/CoSched/>. HARC: A Highly-Available Robust Co-scheduler [Online].
- [16] <http://www.globus.org/toolkit/docs/4.0/execution/wsgam/admin-index.html>. Globus GT 4.0 WS GRAM system administrator's guide [online].
- [17] <http://www.globus.org/wsrf/>. WS-RF: The WS-Resource Framework [Online].
- [18] <http://www.ngs.ac.uk/>. Uk national grid service [online].
- [19] <http://www.teragrid.org/>. Teragrid [online].
- [20] <http://www.wikipedia.org/wiki/REST>. Representational state transfer [online].
- [21] [http://www.wikipedia.org/wiki/Two-phase-commit\\_protocol](http://www.wikipedia.org/wiki/Two-phase-commit_protocol). Two-phase-commit protocol [online].
- [22] 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*, Submitted for special issue on iGrid 2005, awaiting review.
- [23] T. Imamura, B. Dillaway, and E. Simon. W3C Recommendation: XML Encryption Syntax and Processing, December 2002. <http://www.w3.org/TR/xmlenc-core/>.
- [24] D. B. Jackson. Grid scheduling with Maui/Silver. In J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors, *Grid Resource Management: State of the Art and Future Trends*, chapter 11, pages 161–170. Kluwer, 2003.
- [25] 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.
- [26] D. Kuo and M. Mc Keown. Advance reservation and co-allocation protocol for grid computing. In *First International Conference on e-Science and Grid Computing (e-Science'05)*, volume e-science, pages 164–171. IEEE Computer Society Press, 2005.
- [27] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- [28] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.
- [29] L. Lamport. Paxos made simple. In ACM SIGACT news distributed computing column 5. *SIGACT News*, 32(4):18–25, 2001.
- [30] B. W. Lampson. Hints for computer system design. *ACM SIGOPS Operating Systems Review*, 17(5):33–48, 1983.
- [31] 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>.
- [32] A. Nadalin, C. Kaler, P. Hallam-Baker, and R. Monzillo. Web services security: Soap message security 1.0 (ws-security 2004), oasis standard 200401, March 2004. <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>.
- [33] A. Roy. *End-to-End Quality of Service for High-end Applications*. PhD thesis, University Of Chicago, Illinois, 2001. <http://www.cs.wisc.edu/~roy/publications>.
- [34] A. Roy and V. Sander. GARA: A uniform quality of service architecture. In J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors, *Grid Resource Management: State of the Art and Future Trends*, chapter 23, pages 377–394. Kluwer, 2003.
- [35] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.
- [36] D. Skeen. Nonblocking commit protocols. In *SIGMOD'81: Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pages 133–142. ACM Press, 1981.
- [37] 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.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ActionsSucceeded Acceptor="http://vizws00.cct.lsu.edu:8080/accl" location="http://vizws00.cct.lsu.edu:8080/accl/result" method="GET" tid="paxos-ident-16" uid="main1136928630068">
  <Make actionId="1" filename="net22">
    <Resource>
      <Name>calient.ocs.lsu.edu</Name>
      <Endpoint type="REST">
        <RESTEndpoint>http://up.cct.lsu.edu:9292/1su-calient-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule>
      <StartAt>2006-01-10T21:45:00Z</StartAt>
    </Schedule>
    <Ident>GRP_1</Ident>
  </Make>
  <Make actionId="2" filename="input.santaka.igrid">
    <Resource>
      <Name>santaka.cct.lsu.edu</Name>
      <Endpoint type="REST">
        <RESTEndpoint>http://santaka.cct.lsu.edu:9191/santaka-rm</RESTEndpoint>
      </Endpoint>
    </Resource>
    <Schedule>
      <StartAt>2006-01-10T21:45:00Z</StartAt>
    </Schedule>
    <Ident>R1178</Ident>
  </Make>
</ActionsSucceeded>
```

**Figure 7. Example response for a successful co-scheduling request.**

```

Date start_time=...

Map reservations=new HashMap();
Map santaka=new HashMap();

santaka.put (AV.OPERATION,AV.OP_MAKE);
santaka.put (AV.TYPE,AV.TYPE_COMPUTE);
santaka.put (AV.MACHINE,"santaka.cct.lsu.edu");
santaka.put (AV.START_TIME,XMLUtils.
    getDateAsString(start_time));
santaka.put (AV.DURATION,"01:00:00"); // HH:MM:SS
santaka.put (AV.COMP_MEMORY_GB,"5"); // a string
santaka.put (AV.COMP_PROCESSORS,"8"); // a string

reservations.put ("santaka",santaka);

Map lsu_calient=new HashMap();

lsu_calient.put (AV.OPERATION,AV.OP_MAKE);
lsu_calient.put (AV.TYPE,AV.TYPE_NETWORK);
lsu_calient.put (AV.MACHINE,"calient.ocs.lsu.edu");
lsu_calient.put (AV.START_TIME,XMLUtils.
    getDateAsString(start_time));
lsu_calient.put (AV.DURATION,"01:00:00");
lsu_calient.put
    (AV.NETWORK_CONNECTIONS,"santaka:nlr-10gb");
reservations.put ("lsu_calient",lsu_calient);

coscheduler.setInputPairLists (reservations);

try {
    boolean succeeded=coscheduler.
        coscheduleActions();

    if(succeeded) {
        coscheduler.updatePairLists();
        String res_id=(String)santaka.get
            (AV.RESERVATION_ID);
        System.out.println
            ("Resv. id on santaka: "+res_id);
        String res_id2=(String)lsu_calient.get
            (AV.RESERVATION_ID);
        System.out.println
            ("Resv. id on lsu_calient: "+res_id2);
        ...
    }
}

```

**Figure 9. Java co-scheduling client code, with error and exception handling omitted.**