GGF SchedulingWorking Group
SchedWD 12.1

K. Czajkowski
V. Sander
University of Southern California
John von Neumann Institut für Computing
September 11, 2001

Grid Resource Management Protocol: Requirements

1 Introduction

This document outlines the main considerations in defining a standard Grid Resource Management (RM) Protocol to connect agents or clients to resource managers. This generic protocol is intended to support interoperability between Grid resource consumers (clients) and Grid resource managers, independent of resource type or native resource access mode(s). This protocol specifically targets known resource types such as computers, storage systems, and communication networks, but is thought to be general enough to support additional resource types as the need arises. This protocol specifically targets known access modes such as batch job submission, advance reservation, deadline schediling, and service guarantess.

In addition to providing consistent access to individual remote resources, the Grid RM protocol must support the simultaneous use, or co-scheduling, of such resources.

2 Required Features

The generic Grid Resource Management Protocol, as defined by this document, satisfies a number of requirements identified by the Scheduling Working Group. For reference in later discussions, these requirements are enumerated below.

The first two requirements address the core problem of resource management, and are discussed at length in Section 4 of this document:

Requirement 1 (Control) *The core feature of the protocol is to allow clients to control their consumption of potentially remote Grid resources. In addition to individual resource access, this control must support coordinated use of resources in different administrative domains.*

Requirement 2 (Extensibility) *The protocol must provide dynamic access to new resource types and/or access modes not necessarily recognized by all clients or managers.*

The following five requirements address practical issues of Grid resource management that have been identified in previous experimental systems. These requirements are explained further in this section:

Requirement 3 (Notification) *The ability to monitor the status of resources and resource acquisition attempts is essential to the use of resources in dynamic environments. This requires asynchronous notification of important status changes, in addition to the usual synchronous status results.*

Requirement 4 (Reliability) *A reliable semantics must be provided to satisfy clients ranging from interactive users to fully-automated task brokers.*

Requirement 5 (Protocol Timers) *Negotiated soft-state or keep-alive features allow better error-recovery behavior for clients and managers, e.g. providing a timeout semantics to automatically reclaim or cancel orphaned resource assignments.*

Requirement 6 (Negotiation) *Version and feature negotiation is important to facilitate interoperability of large-scale infrastructure while permitting incremental extension and revision of components in the field.*

Requirement 7 (Hierarchy) *Many resource acquisitions are managed hierarchically, e.g. a single reservation may be subdivided and applied to multiple simultaneous or sequential resource allocations. The protocol must provide uniform access and control for such hierarchical management scenarios.*

To simplify this protocol specification, a few complex requirements are isolated as embedded specifications. The definition of the Grid Resource Management Protocol addresses these issues by adopting external standards:

Requirement 8 (Secure Messaging) *The protocol must provide authenticated and integrity-checked delivery of all messages over untrusted networks.*

Requirement 9 (Security Language) *The protocol must embed a policy language to allow the expression and exchange of complex security requirements and policy, e.g. to represent* delegation *of agent rights with limited trust.*

Requirement 10 (Resource Language) *The protocol must embed a structured language to allow the expression and exchange of complex resource requirements and configuration data.*

These embedded problems represent areas of external research. The Grid Resource Management Protocol specification identifies workable solutions for immediate deployment, and is designed to accomodate future advances in these areas.

2.1 Monitoring and Notification

Transient clients must be able to establish the status of a request, and long-running clients should be informed of important asynchronous status changes. Notification interfaces do not affect the status of a request, though they may be used by clients to recover transactional state information, i.e. in the event of client failure. The ability to affect request state is part of the control interface discussed in Section 4.

2.2 Reliable Protocol Semantics

For practical Grid deployment and use, the resource management protocol must provide a strict reliability semantics in the face of network, client, and server failures. At a minimum, the *state* of the resource manager with regard to an ongoing request must be well-defined for all points in the client's interaction. A sufficiently robust client must always know how to determine (or infer) the status of their request, in order to respond appropriately.

To facilitate error recovery, each request must be *persistently named* so that clients may identify and resume an interaction following failure in the network or client. In the face of resource or manager failures, persistent naming also allows the client to determine whether or not a request survived the failure. It is also desireable for manager implementations to use persistent logging techniques to provide better reliability in the face of manager or resource failures, but most of these implementation properties are outside the scope of the protocol specification.

In order to support redundant and.or failover services, the protocol definition must consider any explicit or implicit linkage between protocol naming and network addressing.

2.3 Soft-state/Keep-alive Support

The addition of soft-state or keep-alive semantics improves the behavior of the protocol with respect to severe failures. A client can use keep-alive messages to inform a resource manager of continued interest in interactive allocations, such that in the absence of messages a resource manager could automatically reclaim a resource. Conversely, a client participating in such a keep-alive exchange knows that an allocation is really "gone" (and subsequently the user will not be charged for unexpected resource use) some time-out period after the last keep-alive message is sent.

Due to the wide range of client requirements, any such keep-alive mechanism must be negotiated on a per-request basis. Additionally, such messages must be properly secured to prevent malicious interference that could be detrimental to resource or client.

3 Generalizing Resource Types

The RM protocol must cover acquisition of storage, processing, and communication resources.

Editing note 3.1 *This section will introduce a base set of resource types that must be possible to support via this protocol: compute (CPU, GPU); storage (memory, disk, tape); network (HPC switch, LAN, WAN); misc. (guest account, quota pool)*

This section will also broach topic of resource "sub-type" composites, e.g. storage bandwidth and storage space. Issue: is there a viable abstraction of CPU space versus bandwidth, e.g. thread pools running on dedicated CPU partitions?

4 Generalizing Acquisition Modes

There is a space of acquisition modes we want to support. There are several dimensions to this space and the protocol must express all permutations, though a particular implementation may provide only partial support:

1. service guarantees;
2. access schedule;
3. independent vs. combined allocation;
4. delayed vs. immediate commitment; and
5. dynamic vs. immediate binding.

Different systems provide different names for the points in this acquisition space that they support. With this protocol we should make no attempt to derive compatible names—instead, we should specify a parametric space of concepts and expect implementations to map the protocol concepts to available local implementation points.

4.1 Service Guarantees

Service guarantees formalize the notion of *what* is being acquired or assigned, e.g. *how much* or with what *priority or quality* a resource is allotted to the requesting client. As such, these QoS guarantees identify points in a continuum from best-effort to dedicated access.

Editing note 4.1 *This subsection should give some examples of capacity, quality, priority, etc.*

4.2 Access Schedule

The access schedule determines when the resource is assigned to the client. Clients can request access schedules ranging from best-effort "as soon as possible" allocation, to partially constrained *deadline* based scheduling, all the way to advance definition of the required access period.

Editing note 4.2 *This section should show how simple constraints can cover the range of immediate, deadline, and advance scheduling scenarios.*

How about more complex dependancy graphs? However, "access Schedule" is more relevant for services which are build on top of the protocol. Within the protocol, the characterization seems to be good, but we have to mention this more explicitly.

4.3 Delayed Commitment

Nearly every RM system provides an operation to *cancel* a request. There are often different costs associated with cancellation depending on when the request is canceled. Expenses may include resource *unit* consumption in an accounting system, computational overhead associated with cleanup operations, etc. There is a fuzzy notion of *reversibility* that is often used in connection with cancellation: the more reversible the original request, the cheaper the cancellation.

To simplify cost models for cancellation, systems may employ various *commitment* protocols. In fully asynchronous distributed systems, commitment is often encoded in a *two-phase commit* protocol. A request may be easily canceled anytime until a commit message is exchanged. In partially synchronous systems, commitment is usually mixed with *protocol timers*. A request may be canceled or committed via message exchange anytime prior to a timeout deadline, and at the deadline the request is either automatically canceled or automatically committed depending on the protocol employed.

Advance requests—those requests constraining access start to a determinate future time—always provide a *delayed commitment* semantics. Depending on the request, commitment may require an additional message or may be triggered automatically at the request start time. The availability of, or requirement for, a commitment message must be negotiated between a particular client/manager pair.

4.4 Dynamic Binding

Every RM interface allows the *binding* of configuration parameters to a request. Some RM systems allow (or require) dynamic binding of configuration parameters to an existing request. There can be a complex relationship between binding and commitment, because in some cases a request cannot proceed to an "active" stage until a full set of configuration data is bound. However, some systems allow the changing or *rebinding* of configuration data on an active, committed request.

5 Embedded Resource Language

The ability to express complex and extensible requirements suggests embedding a structured resource description language inside the protocol. Until a resource description language is standardized by the Grid Forum, we recommend embedding the existing Globus RSL language which is a declarative expression language with well-understood capabilities and limitations.

Editing note 5.1 *This section should elaborate content requirements, including interaction with the Security Model and Policy language.*

6 Communication Model

The RM protocol standard should adopt the Grid community best-practices for secure and reliable remote interactions. This adoption will be expressed in a follow-on document explaining the binding of RM protocol elements to transport-level mechanisms.