SNAP: A Protocol for Negotiation of Service Level Agreements and Coordinated Resource Management in Distributed Systems

Karl Czajkowski [*] Ian Foster [†‡] Carl Kesselman [*] Volker Sander [§] Steven Tuecke [†]

[*] Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292 U.S.A.

[†] Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439 U.S.A.

[‡] Department of Computer Science
The University of Chicago
Chicago, IL 60657 U.S.A.

[§] Zentralinstitut für Angewandte Mathematik
Forschungszentrum Jülich
52425 Jülich, Germany

Abstract

A fundamental problem with distributed applications is how to map activities such as computation or data transfer onto a set of resources that will meet the application's requirement for performance, cost, security, or other quality of service metrics. An application or client must engage in a multi-phase negotiation process with resource managers, *as it discovers, reserves, acquires, configures, monitors, and potentially renegotiates resource access. We present a generalized resource management model in which resource interactions are mapped onto a well defined set of symmetric and resource independent service level agreements. We instantiate this model in (the* Service Negotiation and Acquisition Protocol *(SNAP) which provides integrated support for lifetime management and an at-most-once creation semantics for SLAs. The result is a resource management framework for distributed systems that we believe is more powerful and general than current approaches. We explain how SNAP can be deployed within the context of the Globus Toolkit.*

1 Introduction

A common requirement in distributed computing is to negotiate access to, and manage, a resource that exists within another administrative domain. Satisfaction of this requirement is complicated by the competing needs of the *client* and the *resource owner*. The client wants to understand resource behavior while the owner wants to maintain local control and discretion. We require resource management (RM) mechanisms that reconcile these two competing demands by enabling the negotiation of *service-level agreements* (SLAs) by which a resource provider "contracts" with a client to provide a some measurable capability, thus allowing clients to understand what to expect from resource owners. Such mechanisms must also address the inherently volatile nature of the distributed environment, where the nature of policy domains, applications, and communities change dynamically, and faults are a normal occurrence.

The ability to negotiate SLAs is desirable as a means both of ensuring a client's ability to deliver a desired quality of service (QoS) to its user and as a means of permitting objective comparison of alternative resource providers. In addition, if a client can negotiate SLAs with a temporal dimension ("service S will be available at time T"), then they can permit a client to coordinate or co-schedule resources from different administrative policy domains, coordinating resource schedules according to its knowledge of an overall application activity schedule.

Such application-driven coordination requires that resources present a rational behavioral model to clients. A client cannot hope to know everything about a resource's behavior, but they must be able to understand, in some abstracted form, *what the resource will do for them*. This understanding can be expressed in the form of SLAs that describe a provisioning of resource capacity or service quality by the resource for the client. This approach enables a separation of concerns between resource and client: the resource is not directly concerned with meeting application goals, but merely with satisfying client provisioning requirements, leaving the client to be responsible for meeting its application scheduling goals by coordinating its service agreements with one or more resources. Higher-level brokers can also make provisioning agreements, but these services in turn are clients to the underlying distributed resources.

We propose here a resource management architecture and supporting protocol based on these ideas. Building on a

rich and extensible language for describing the form, initiation time and duration of resource assignments, the *Service Negotiation and Acquisition Protocol* (SNAP) defines message exchanges for negotiating three types of SLAs:

- *Resource acquisition agreements* in which one negotiates for the right to use (i.e. consume) a resource. For example, an advance reservation takes the form of an acquisition agreement.
- *Task submission agreements* inform a resource of the existence of a task or activity. Such an agreement may be associated, for example, with submitting a job to a queuing system.
- *Task/resource binding agreements* associate an acquired resource with a submitted activity, enabling the activity empowered by the user to consume the agreed upon resource capacity granted by the resource provider.

To ensure reliable RM operation independent of transport reliability, the protocol consists of idempotent operations that maintain the service-side agreement state using a soft-state semantics.

The RM approach proposed here extends techniques first developed within the Globus Toolkit's GRAM service [9] and then extended in the experimental GARA system [21, 22, 34]. An implementation of this architecture and protocol can leverage a variety of existing infrastructure, including the Globus Toolkit's Grid Security Infrastructure [19] and Monitoring and Discovery Service [8]. We expect SNAP protocol to be easily implemented within the Open Grid Services Architecture (OGSA) [18, 38], which provides request transport, security, discovery, and monitoring.

2 Motivating Scenarios

The design of SNAP is motivated by our experiences with Grid applications [16], which place numerous and often complex demands on RM services [22] due to the need to deliver performance guarantees to users while executing on heterogeneous mixes of resources.

These scenarios fall into two major categories, representing the perspective of application builders versus that of resource owners:

- *Application scheduling* attempts to optimize performance from the perspective of goals, ideally by permitting application control of any planning decision.
- *Aggregate scheduling* attempt to optimize performance from the perspective of the resource provider (the owner) or, in the case of multiple resources, from the perspective of the community, ideally by permitting a scheduler with complete knowledge of the resource to control all planning decisions.

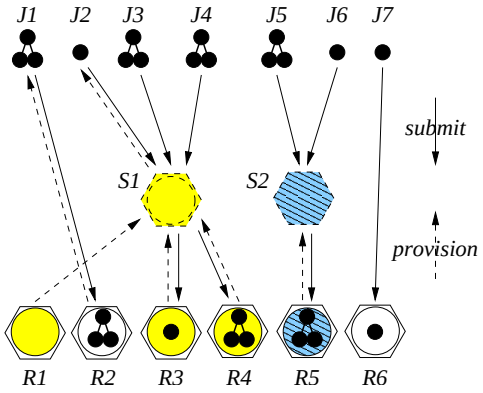

Figure 1. Community scheduler scenario. Multiple users (J1–J7) gain access to shared resources (R1–R7). Community schedulers (S1–S2) mediate access to the resources by obtaining *promises* of capacity from the individual resources and in turn make *submission* agreements with the users.

However, the Grid is ripe with conflicting policies. Neither of these scheduling ideals is reached in practice, and instead RM systems must focus on achieving a dynamic balance between these ideals. Our RM architecture is structured around the negotiation of SLAs to solve this balancing problem at run-time.

There are many familiar examples of resource capacity assignment agreements: space-sharing computer reservations, storage reservation or pre-allocation, and communication or input/output quality of service (QoS) can all be considered as capacity provisioning. The client obtains a promise of resource availability that they may then exploit in further application scheduling and resource control interactions.

Similarly, there are existing scenarios for activity submission SLAs in which the client delegates the activity to the resource with the agreement that it will accomplish the assigned task: job execution, reliable file transfer, and other batch control interfaces all provide more abstract guarantees of delivery. The scheduler accepts responsibility for performing an activity with some quality metrics such as reliability or completion time. The complexity of real-world scenarios comes from the combination of these simple SLAs. In the remainder of this section we explore two such combined scenarios to introduce the basic structure of our architecture.

2.1 Community Scheduler Scenario

In this scenario, a community scheduler negotiates capacity guarantees with a pool of underlying resources. Users can then submit jobs to this community scheduler,

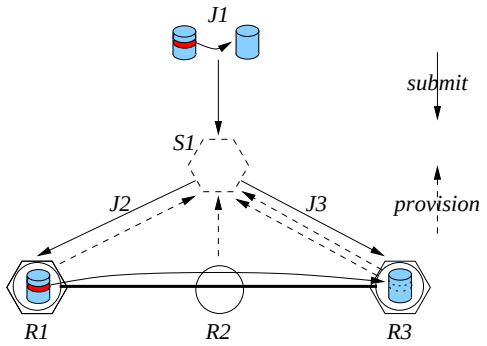

Figure 2. File transfer scenario. File transfer scheduler obtains disk and network reservations before submitting transfer endpoint jobs.

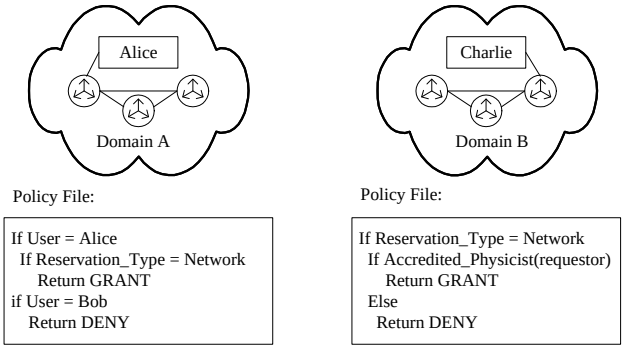

Figure 3. End-to-end resource requirements involve coordination of resources from complex heterogeneous policy environments.

perhaps with deadline requirements. Users in this environment interact with community and resource-level schedulers as appropriate for their goals and privileges. A batch job such as J7 in Figure 1 may not need resource promises, nor the help of a community scheduler, because the goals are expressed directly in the submission to resource R6. The interactive job J1 needs a resource promise to better control its performance. Jobs J2 to J6 are submitted to community schedulers S1 and S2 which might utilize special privileges or domain-specific knowledge to efficiently implement their community jobs. Note that not all users require resource promises from the community scheduler, but S1 does act as a promise "reseller" between J2 and resource R3. Scheduler S1 also maintains a resource (R1) in an idle state to more rapidly serve future high-priority job requests.

The agreement model abstracts away the impact of other community schedulers as well as any "non-Grid" local workloads, provided the resource manager enforces guarantees at the resource. As depicted in Figure 1, a Grid environment may contain many resources (R1–R6), all presenting an assignment interface as well as a submission interface; multiple community scheduler services (S1 and S2) present a submission interface to users, and some may also broker resource promises.

Due to the complex Grid policy environment, the community schedulers usually are not authoritative over physical resources. Thus, the existence of the capacity agreements simplifies the scheduler's resource planning problem in the face of competing workloads by other clients of the same resource pool. The SLA negotiation protocol allows them to coordinate their scheduling of resources shared with other community and application schedulers, *without necessarily trusting those other schedulers*. The only trust implied in this scenario is the vertical trust encoded in the pairwise SLAs between the scheduler and the resource provider.

2.2 File Transfer Service Scenario

In this scenario, users submit a file transfer job to a community scheduler. The scheduler understands that a transfer requires substantial storage space on the destination resource, and substantial network and endpoint I/O bandwidth during the transfer. This situation extends the Community Scheduler scenario with the ability to manage multiple resource types and perform co-scheduling of these resources.

As depicted in Figure 2, the file transfer scheduler S1 presents a scheduler interface, and a network resource manager R2 presents a resource-assignment interface. A user submits a transfer job such as J1 to the scheduler with a deadline. The scheduler obtains a storage pre-allocation on the destination resource R3 to be sure that there will be enough space for the data before attempting the transfer (in essence, a storage pre-allocation is an open-ended reservation that occurs in many first-come, first-served storage policy environments). Once space is allocated, the scheduler obtains bandwidth reservations from the network and the storage devices, giving the scheduler confidence that the transfer can be completed within the user-specified deadline. Finally, the scheduler submits transfer endpoint jobs J2 and J3 to implement the transfer J1 using the space and bandwidth promises.

The distributed applications common in Grid environments exacerbate the coordination problems described for community schedulers. Not only do SLAs coordinate use of resources by mutually distrustful schedulers, they also coordinate the use of distrustful resources for a single application goal. The file transfer scenario emphasizes such distributed goals by requiring real-time coordination of significant endpoint and network capability.

2.3 Policy Complications

As mentioned above *end-to-end* services often require the *co-scheduling* of several distinct resources. A number

of technical issues complicate the co-scheduling process. Co-scheduling can require negotiation with resource owners in each of several distinct administrative domains. Each domain may have different policies governing who can use its resources and for what purposes, and different trust relationships with individual users. For example, in Figure 3, domain A's policy might state that "Alice can use the network, Bob cannot," while domain B's policy is that "only accredited physicists can use the network."

A policy expressed in one domain can also be dependent on conditions in other domains. For example, domain A may wish to enforce the policy "I will only authorize a reservation if reservations have also been approved for all other resources in the end-to-end path." Or, domain B might only authorize bandwidth greater than 10 Mb/s if domain A has a valid assignment agreement for a consumer. As stated above, the resource description language is compositional and thus automatically capable of expressing these dependencies.

2.4 Scalability Concerns

Scalability demands that every resource should not have a direct trust relationship with every user. While some domains know about individuals (e.g., domain A), others must be able to delegate responsibility for personal trust relationships to third parties. (For example, domain B agrees to provide resources to anyone whom a third party accredits as a "physicist."). By virtualizing a set of resources from other managers for the benefit of its user community we can eliminate the direct relationship between end-users and resource providers. Following the concept of the Community Authorization Service (CAS) [31], the intermediate Community Scheduler interacts with the resource providers on behalf of the whole community and is thus the entity of interest.

If a set of applications creates many parallel flows between the same two end-domains, it is infeasible to negotiate agreements for each one. Not only do agreements require non-trivial storage and computational resources to be implemented in some types of manager, but the shear number of SNAP negotiation messages could be impractical for some flows. We address agreement scalability with aggregate scheduling in Section 4, permitting multiple claims against the same acquisition agreement as well as composite tasks in submission and task/resource binding agreements.

2.5 Resource Virtualization

In the scenarios above, the Community Scheduler virtualizes a set of resources from other managers for the benefit of its community of users. This type of resource virtualization is important as it helps implement the trust relationships that are exploited in Grid applications. The user community trusts their scheduler to form agreements providing resources or performing tasks, and the scheduler has its own trust model for determining what resources are acceptable targets for the community workload.

Another type of virtualization in dynamic service environments like OGSA is the *factory* service. A manager in such an environment is a factory, providing a long-lived contact point to initiate RM agreements. The RM factory exposes the agreements as set of short-lived, stateful services which can be manipulated to control one agreement. Resource virtualization is particularly interesting when the agreement runs a job which can itself provide Grid services. This process is described for active Grid storage systems in [7], where data extraction jobs convert a compute cluster with parallel storage into a high-performance data server. Such an action can be thought of as the dynamic deployment of new services "on demand," a critical property for a permanent, but adaptive, global Grid [20].

3 A Model of Resource Management

In this section, we present an abstract model of resource management that captures the process from the perspective of the resource consumer (e.g. in terms of *resource requirements*) and the resource provider (e.g. in terms of promised *resource capabilities*). We describe the connections between these abstractions in more detail below in Section 3.1, before formalizing an *agreement language* in Section 3.2.

3.1 Abstracting Resource Management

The root of resource management is orchestrating the consumption of some resource by an activity. However, as implied by scenarios from Section 2 the process of getting to this end result is different when from the perspective of the resource consumer or the resource provider. The relationship between these different perspectives is captured in Figure 4. The consumer's perspective is shown on the right, and is concerned with discovering, acquiring and ultimately using resources. The left-hand side of the figure shows the resource providers perspective, being concerned with how to accept tasks or activities and how to map them onto the resources capabilities, again, resulting in resource consumption.

3.1.1 Concrete Utilization

At the core of the resource management model (Figure 4) is the *concrete utilization* of resources, where real work is done by expending resources on activities. All other resource management concepts serve to abstract some aspect of this utilization for the purpose of understanding or communication. An example of resource utilization would be the control of a particular executable program or procedure with appropriate parameters for the activity at hand. As implied in the figure, concrete utilizations occur on discrete

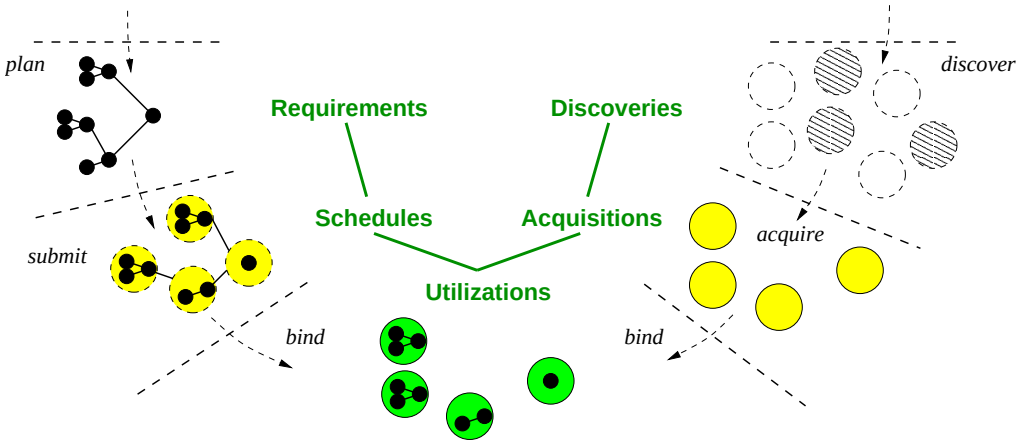

Figure 4. Grid RM abstraction domain. Resource utilization can be interpreted variously from a task-centric and resource-centric perspective; both are useful for understanding the task planning process.

resources over finite time intervals. Any required orderings are enforced by sequencing of request and completion events.

3.1.2 Submission and Acquisition Abstraction

Resource capability rights are obtained prior to, or simultaneously with, concrete utilization. Thus in our model, resource utilization is achieved by the merging of two separate abstract resource management paths: one representing the abstraction of a resource into the schedule of activities or tasks for which the resource will be used, and the second representing the assignment of capability for unspecified use. As illustrated in Figure 4, activities are manipulated by engaging in a *submission* operation, while capability is manipulated via *acquisition*. The two are bound together to create a concrete utilization.

Thus withing the proposed model, a schedule has quantifiable requirements, and the acquired resource capability *satisfies* those requirements. A broker tracking both abstractions can thus implement the schedule by matching the requirements with the capabilities of the controlled resources.

3.1.3 Requirements-driven Discovery Abstraction

An application planner is imbued with a task plan involving work-flow dependencies and requirements. Resource discovery informs the task planner of *managed resource pools* such that evaluation and acquisition may commence. This level is distinguished from the lower scheduling and acquisition level by its imprecision. The activity requirements may involve application deadlines or other run-time constraints but they do not yet identify a specific resource-bound execution plan. The discovery pool consists of all known resources that might satisfy these activity requirements. The scheduler evaluates the members of the pool to select candidate resources; but until the scheduler successfully acquires a target subset from these candidates, there is no uniquely identified task plan.

3.2 Agreement Language

The previously introduced abstractions scope the meaning of an expressive *resource language*. We formalize the structure of such a language later in Section 5, but let us imagine for the moment an activity (job) language J capable of expressing activity plans and requirements. Let us further assume a resource language R⊆J rich enough to express resource provisioning metrics, whether for requirements in instances of J or as stand-alone provisioning statements. We can then begin to formalize the language of agreement which is parameterized to include capability or quality instances $r \in$ R, or activity instances $j \in$ J.

In the following formalization, the resource language R is expected to model capability or quality over time. We annotate an abstract capability r to fix it to a particular time interval, e.g. $r^{[t_0,t_1]}$ restricts the described capability r to times between t_0 and t_1. Furthermore, an agreement can only pertain to the future (since anything in the past is no longer subject to agreement but rather to auditing). Therefore at wall-clock time t', any agreement regarding $r^{[t_0,t_1]}$, where $t_0 < t' < t_1$, can be normalized to the current time as $r^{[t',t_1]}$. As a short-hand we will use non-normalized expressions such as $r^{[0,t_1]}$, rather than constantly having to formally define the current wall-clock time.

3.3 Limited Trust in Agreements

In the remainder of this section we define an agreement language within which promises of resource availability, job execution, and resource utilization are expressed. These promises model the expected behavior of the RM service with which the client establishes the agreement, but we do not attempt to model the quality or veracity of an RM service.

Ideally, managers should not make agreements expressed with provisioning requirements that the manager cannot or will not observe. However, due to the possibility of faults and the possibility of preemption by higher-priority requests, it is possible that an agreement will be violated after acceptance. It is assumed that a monitoring infrastructure will be in place to augment this RM service protocol, such that clients can monitor the health and status of the manager and their agreements. We assume the activity language J would be expressive enough to capture varying degrees of commitment in a requirements description. We also assume a language for describing the status of agreements, or agreement elements—a minimal model would capture the agreement states depicted later in Section 4.1 and Figure 5, but a better model would allow extension with application-specific states to better communicate application progress.

It is beyond the scope of the Grid RM system to define how much a client can rely on the promises of a scheduler, as this is essentially a trust policy between the user and scheduler. The more trustworthy a scheduler or manager, the more confidence the user might have in agreements made with the manager. If not in a completely ad-hoc manner, this sort of trust must be established via security and information services; for example by inquiring with a trusted accrediting service [26]. One can also imagine imposing penalties if an agreement is not satisfied.

3.3.1 Assignment Agreements

We represent an assignment of resource capability or service quality r from manager m to client c over time period $[t_0, t_1]$ by a tuple with the form:

$$\text{assign}\langle m, c, r^{[t_0,t_1]}\rangle$$

For example, the configuration of a local RM system M by an organization O could include a delegation of the computing resources R for all time:

$$\text{assign}\langle \text{O}, \text{M}, \text{R}^{[0,\infty)}\rangle$$

and this delegation record could be used by the manager to model what delegations it is capable of issuing.

Using predictive techniques [12, 41, 35], it is possible for a manager to provision resources that are not yet available to the manager. Therefore, the Grid RM agreement protocol only models the pair-wise assignment promises $\mathcal{A}_\text{M}$ made by a manager M to clients c:

$$\mathcal{A}_\text{M} = \{\text{assign}\langle \text{M}, c, r\rangle\}$$

where the universe of assignment tuples has of course been restricted to successfully negotiated agreements.

3.3.2 Submission Agreements

Symmetric to assignment, the submission state $\mathcal{Q}_\text{S}$ of a scheduling agent models the commitments the scheduler S has made to clients c, that an activity j will be performed according to its self-expressed requirements:

$$\mathcal{Q}_\text{S} = \{\text{queue}\langle c, \text{S}, j\rangle\}$$

and the creation and management of this state is defined below in the protocol definition.

3.3.3 Utilization Agreements

The utilization of a resource r in implementing an activity j may be also represented as an agreement. This representation is useful as it captures the association of pre-existing resource promises and activities, whether initiated by clients or established internally by the scheduling algorithm. As will be explained in Section 4.5, activities may be referenced that are not part of a submission agreement and we exploit the activity language J assuming that it allows reference to external activities. These agreements also might serve as a language for describing scheduler state through a monitoring interface.

Similar to assignment and submission, the utilization state $\mathcal{U}_\text{M}$ of a manager models the commitments the manager M has made to clients c:

$$\mathcal{U}_\text{M} = \{\text{apply}\langle \text{M}, c, r, j\rangle\}$$

This agreement indicates the manager M's commitment to clients c that an existing resource capability referenced by r be applied to the activity referenced by j.

4 The SNAP Agreement Protocol

The core of our RM architecture is a client-service interaction used to negotiate service-level agreements. These agreements can be viewed as establishing transitory additions to the policy environment of a resource manager, expressed using the agreement language from Section 3.2. The protocol applies equivalently when talking to authoritative, localized resource owners or to intervening brokers.

There are two levels to this interaction. First, there is a set of abstract message-based *operations* that atomically change the policy state of the remote endpoint. Second,

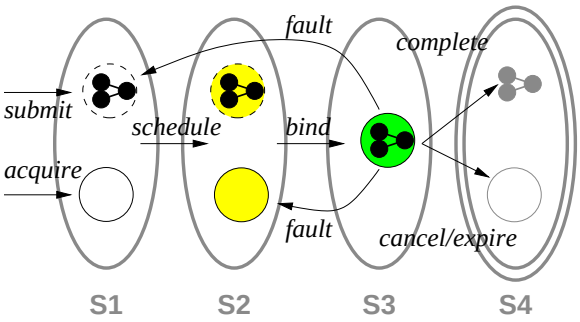

Figure 5. Agreement state transitions. Assignment and submission agreements get associated by utilization agreements and may eventually terminate due to client operations or expiration.

short-term connection or messaging state is used to transmit and reassemble the more abstract messages. In this article we focus on the pattern and meaning of the abstract messages and leave details of the transport state to be determined by a suitable low-level specification.

We describe each operation in terms of unidirectional messages sent from client to service or service to client. All of these operations follow a client-server remote procedure-call (RPC) pattern, so we assume the underlying transport will provide correlation of the initiating and response messages. One way of interpreting the following descriptions is that the client to service message corresponds to the RPC call, and the return messages represent the possible result values of the call. This interpretation is consistent with how such a protocol would be deployed in a Web Services or OGSA environment, using WSDL to model the RPC messages [6, 1].

4.1 Agreement State Transitions

The association of submitted activities with acquired resources in a utilization agreement is not always a protocol operation, but is nonetheless an observable transition of the service state, as depicted in Figure 5. In essence there are four states through which planning progresses:

S1: Submitted activities or acquired resources are not matched with each other.

S2: Submitted activities are matched with appropriate resource acquisitions, and this grouping represents a resource utilization agreement meant to resolve the activity.

S3: Acquired resources are being utilized for a submitted activity and can still be controlled or changed.

S4: The agreements have been resolved either by successful completion of the activity, or by expiration or cancellation of the agreements.

These states have a relationship to protocol messages in that client messages can only create or operate on active agreements and inactive or finalized agreements can only occur as a result of client messages and the passing of time. However, the figure simplifies the situation in that the state changes are not actually synchronized between submission and assignment agreements. Related submission or acquisition agreements may in fact move through these planning states at different rates, because activities may share the same resource capability but also may consume different capabilities sequentially.

4.2 Idempotence and Lifetime

Resource management has important fault-tolerance requirements. In many situations, it can be more important that a recovery model be *predictable* rather than optimized for some particular behavior. On the one hand, it is important that agreement creation be unambiguous: many scheduling and planning activities must be able to determine with high confidence that an agreement either has, or has not, been initiated. On the other hand, agreements are sometimes "orphaned" due to unexpected failures or even just sloppy users—it is desirable to be able to reclaim resources provisioned to these orphans if such reclamation can be done while maintaining the guarantees of unambiguous agreement.

We believe that *idempotence* (i.e. at most once semantics) combined with *expiration* is well-suited to achieving both goals for fault-tolerant agreement. We define our operations as atomic and idempotent interactions that create agreement statements in the manager. Each agreement statement has a termination time, after which a well-defined reclamation effect occurs. This termination parameter can be exploited at runtime to implement a spectrum of negotiation strategies: a stream of short-term expiration updates could implement a heart-beat monitoring system [36] to force reclamation in the absence of positive signals, while a long-term expiration date guarantees agreement state will persist long enough to survive transient outages.

A client is free to re-issue requests, and a manager is required to treat duplicate requests received after a successful agreement as being equivalent to a request for acknowledgment on the existing agreement. In the case of failure, the client cannot distinguish whether a manager recognizes duplication. This idempotence is achieved in part by introducing a mechanism for establishing a unique name for each agreement.

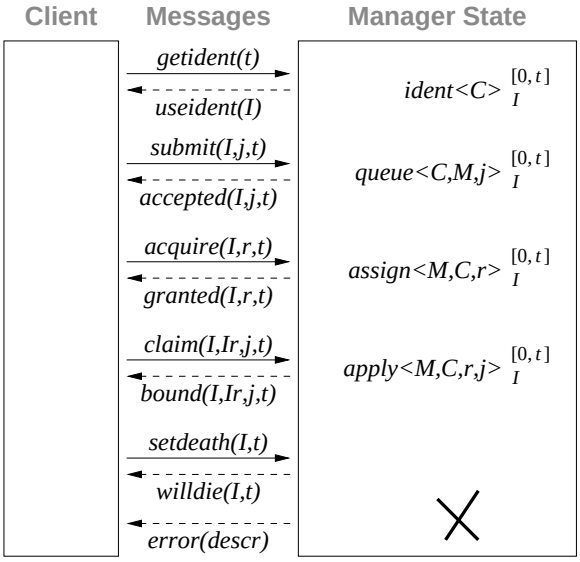

Figure 6. RM protocol messages. The protocol messages initiate *operations* that change the agreement state of the manager.

4.3 Support Operations

4.3.1 Allocate Identifier Operation

There are multiple approaches to obtaining unique identifiers suitable for naming agreements. To avoid describing a security infrastructure-dependent approach, we suggest a special light-weight agreement to allocate identifiers from a manager. This operation is analogous to opening a timed transaction in a database system. The client sends:

$$\mathbf{getident}(t_{\mathrm{dead}})$$

asking the manager to allocate a new identifier that will be valid until time t_{dead}. On success, the manager will respond:

$$\mathbf{useident}(I, t_{\mathrm{dead}})$$

and the client can then attempt to create reliable RM agreements using this identifier as long as the identifier is valid. The main agreement operations, described below, atomically reassign I from a self-labeled identifier agreement to label their respective RM agreement.

4.3.2 Set Termination Operation

To support a wide range of fault-recovery techniques, the lifetime of an existing agreement can be updated. With this operation, a client can set a new termination time for the state. The client changes the lifetime by sending a message of the form:

$$\mathbf{setdeath}(I, t_{\mathrm{dead}})$$

where t_{dead} is the new wall-clock termination time for the existing agreement state labeled by I. On success the manager will respond with the new termination time:

$$\mathbf{willdie}(I, t_{\mathrm{dead}})$$

and the client may reissue the $\mathbf{setdeath}(\ldots)$ message if some failure blocks the initial response. Throughout this protocol proposal, agreements can be abandoned with a simple request of $\mathbf{setdeath}(I, 0)$ which forces expiration of the agreement.

The lifetime represented by t_{dead} is the lifetime of the agreement named by I. If the agreement makes promises about times in the future beyond its current lifetime, those *promises expire with the agreement statement, no matter the details of the promise*. Thus, it is a client's responsibility to extend or renew a future promise for the full duration required.

4.4 Agreement Operations

The protocol provides a way for clients to enter into agreements with managers as described in Section 3.2: assignment agreements are promises of resource availability; submission agreements are promises of activity scheduling; and utilization agreements are promises of resource application to an activity. Submission agreements are established explicitly using the submission operation described below. Assignment agreements can be established explicitly using the acquisition operation described below, and they may also be established implicitly when an activity scheduler plans the completion of a submitted activity.

There are three ways that utilization agreements are established in our architecture. A submitted activity description j may refer to an existing assignment agreement in the manager. Likewise, an assignment agreement description r may refer to an existing submission agreement. In these cases, the scheduler knows to apply the resource promise to facilitate the completion of the activity. The final form of utilization is through an explicit *claim* operation that informs the local RM system to *bind* resources to an ongoing activity.

All managers need not present all three interfaces in our architecture. A goal-oriented job scheduler might only accept jobs with performance goals such as completion time or real-time responsiveness. A computer reservation system might provide assignments of processing nodes and job submission "into" those processing nodes without permitting reversed binding of nodes to existing jobs. When this architecture is mapped into an implementation model and service environment, these interface policy preferences must be represented clearly to enable client discovery of manager capability.

4.4.1 Submission Operation

The submission operation is initiated by a client and results in a new activity agreement being accepted within the scheduler S if the operation is successful. On failure to accept an activity agreement, S's state is unchanged. The client C initiates an agreement attempt by sending:

$$\mathbf{submit}(\mathrm{C}, I, j, t_{\mathrm{dead}})$$

The resource description j captures all the temporal and other requirements of the client. On success, S will respond with a message of the form:

$$\mathbf{accepted}(\mathrm{queue}\langle \mathrm{C}, \mathrm{S}, j\rangle_I^{[0,t_{\mathrm{dead}}]})$$

After the accept, the client C knows that the new activity agreement exists in the scheduler S:

$$\mathrm{queue}\langle \mathrm{C}, \mathrm{S}, j\rangle_I^{[0,t_{\mathrm{dead}}]} \in \mathcal{Q}_{\mathrm{S}}.$$

In other words, the scheduler S agrees to run job j on behalf of the client C. The agreement does not imply that the job is running, but merely that S has agreed to fit the job into its schedule while satisfying the requirements stated in j. The agreement will remain in the scheduler's state until time t_{dead} unless the client performs a $\mathbf{setdeath}(I, t)$ operation to change its scheduled lifetime. Any temporal requirements of the job are captured in the description j, but the scheduler may abandon the agreement and reclaim resources if the agreement expires before the job has been completed.

4.4.2 Acquisition Operation

The acquisition operation is initiated by a client and results in a new resource provisioning agreement within the manager M if the operation is successful. On failure, M's state is unchanged. The client C initiates the operation by sending a message of the form:

$$\mathbf{acquire}(\mathrm{C}, I, r, t_{\mathrm{dead}})$$

The resource description r captures all the temporal and other requirements of the client. On success, M will respond with a success message of the form:

$$\mathbf{granted}(\mathrm{assign}\langle \mathrm{M}, \mathrm{C}, r\rangle_I^{[0,t_{\mathrm{dead}}]})$$

After the grant, the client C knows that a new provisioning agreement exists in the manager M:

$$\mathrm{assign}\langle \mathrm{M}, \mathrm{C}, r\rangle_I^{[0,t_{\mathrm{dead}}]} \in \mathcal{A}_{\mathrm{M}}.$$

In other words, the manager M agrees to make resource r available to client C. The agreement does not imply that the resource is idle or actively assigned to C, but that C should expect success when it tries to claim the resource. The agreement will remain in the manager's state until time t_{dead} unless the client performs a $\mathbf{setdeath}(I, t)$ operation to change its scheduled lifetime.

4.5 Claim Operation

The claim operation is initiated by a client and results in a new utilization agreement with the manager M if the operation is successful. On failure, M's state is unchanged. The client C initiates by sending:

$$\mathbf{claim}(\mathrm{C}, I, r, j, t_{\mathrm{dead}})$$

The description r references an existing assignment agreement between M and C, and the activity description j references an existing activity. On success, M will respond to C with a message of the form:

$$\mathbf{bound}(\mathrm{apply}\langle \mathrm{M}, \mathrm{C}, r, j\rangle_I^{[0,t_{\mathrm{dead}}]})$$

After the bind, the client C knows that a new utilization agreement exists in the manager M:

$$\mathrm{apply}\langle \mathrm{M}, \mathrm{C}, r, j\rangle_I^{[0,t_{\mathrm{dead}}]} \in \mathcal{U}_{\mathrm{M}}.$$

The claim operation allows referencing of activities not known at the time of the resource assignment, which are not necessarily initiated through the activity submission interface. A typical example is the specification of the client port-number for network reservation. This information is typically not available in advance. The description r may often reduce to I_r referencing an assignment already in place. In the degenerate case of an existing activity agreement between M and C, j reduces to I_j. For external activities such as a network communication channel, the extensible aspects of the language J are exploited to identify the activity.

4.6 Change

Finally, we support the common idiom of atomic *change* by allowing a client to resend the same initiating message identifier and format, but with modified requirement content. The service will respond as for an initial request, or with an error if the given change is not possible from the existing RM state. When the response indicates a successful state, the client knows that the original agreement named by I has been replaced by the new one depicted in the response. When the response indicates failure, the client knows that the state is unchanged from before the request.

In essence, the service compares the incoming initiation request with its internal policy state to determine whether to treat it as a *create*, *change*, or *lookup*. The purpose of change semantics are to preserve state in the underlying resource behavior where that is useful, e.g. it is often possible to preserve an I/O channel or compute task when QoS levels are adjusted. Whether such a change is possible may depend both on the resource type, implementation, and local policy. If the change is refused, the client will have to

initiate a new request and deal with the loss of state through other means such as check-pointing.

Change is also useful to adjust the degree of commitment in an agreement. An expected use is to monotonically increase the level of commitment in a promise as a client converges on an application schedule involving multiple resource managers, essentially implementing an arbitrary-length, timed, multi-phase commit protocol across the managers which may be in different administrative domains. However, there is no requirement for this monotonic increase—a client may also want to decrease the level of commitment if they lose confidence in their application plan and want to communicate a relaxation to the manager.

5 Resource and Schedule Language

The resource and scheduling language assumed in Section 3.2 plays an important role in our architecture. Because resources are often shared or virtualized in a Grid environment, resource description is more important than resource naming. Clients in general must request resources by property, e.g. by capability, quality, or configuration. Similarly, clients must understand their assignments by property so that they can have any expectation of delivery in an environment where other clients' assignments and activities may be hidden from view.

We believe that resource description must be dynamically extensible. Sets of clients and resources must be able to define new resource semantics to capture novel devices and services, so the language should support these extensions in a structured way. A complex new semantics can be captured by composing existing primitives, and hopefully large communities will be able to standardize a relatively small set of such primitives.

5.1 Capacity Metrics

Many resources have parameterized attributes, i.e. a metric describing a particular property of the resource such as *bandwidth*, *latency*, or *space*. Resource assignments will scope these metrics to a window of time $[t_0, t_1]$ in which the client desires access to a resource with the given qualities. We use a generic scalar metric and suggest below how they can be composed to model conventional resources.

A scalar metric can exactly specify resource capacity. Often requirements are partially constraining, i.e. they identify ranges of capacity. We extend scalar metrics as unary inequalities to use the scalar metrics as a *limit*. The limit syntax can also be applied to time values, e.g. to specify a start time of "$\leq t$" for a provisioning interval that starts "on or before" the exact time t.

Time metric t expressed in wall-clock time, e.g. "Wed Apr 24 20:52:36 UTC 2002."

Scalar metric $x\,u$ expressed in *real* valued *units*, e.g. $512\,\mathrm{Mbytes}$, $12\,\mathrm{Mbytes/s}$, or $10\times10^{-3}\,\mathrm{s/seek}$.

Max limit $<m$ and $\leq m$ specify an exclusive or inclusive upper limit on the given metric m, respectively.

Min limit $>m$ and $\geq m$ specify an exclusive or inclusive lower limit on the given metric m, respectively.

These primitives are "leaf" constructs in a structural resource description. They define a syntax, but some of their meaning is defined by the context in which they appear.

5.2 Resource Composites

The resource description language is compositional. Realistic resources can be modeled as composites of simpler resource primitives. Assuming a representation of resources r_1, r_2 etc. we can aggregate them using various typed constructs. For convenience in naming and identifying resources, we use an optional notation $\langle r \rangle_I$ where the identifier I is an arbitrary name for the resource r. Identifiers for siblings in a composite must be unique in order to be meaningful.

Typed Group $\left[\langle r_1\rangle_{I_1}, \langle r_2\rangle_{I_2}, \ldots\right]_{type}$ combining arbitrary resources. Groups are marked with a *type* to convey the meaning of the collection of resources, e.g. $[x_1\ \mathrm{bytes}, x_2\ \mathrm{bytes/s}]_{\mathrm{disk}}$ might collect space and bandwidth metrics for a "file-system" resource.

Array $n \times r$ is an abbreviation for the group of identical resource instances $[\langle r\rangle_1, \langle r\rangle_2, \ldots, \langle r\rangle_n]_{\mathrm{array}}$, e.g. for convenient expression of symmetric parallelism.

The requirements expressed in R may capture complicated temporal constraints. For example, an important metric of a resource might denote an instantaneous property $f(t)$ restricted to a range $x_1 \leq f(t) \leq x_2$. Such a property can also be abstracted over the time domain to denote an accumulation $\int_{t_0}^{t_1} f(t)\,dt$ or an average value $(\int_{t_0}^{t_1} f(t)\,dt)/(t_1 - t_0)$, each of which could be constrained as an equality or inequality. The purpose of typed groups is to provide this denotational context to the metric values inside—in practice the meaning is denoted only in an external specification of the type, and the computer system interrogating instances of R will be implemented to recognize and process the typed composite.

Resources are required over periods of time, i.e. from a start time t_0 to an end time t_1, and we denote this as $r^{[t_0,t_1]}$. A complex time-varying description can be composed of a sequence of descriptions with consecutive time intervals:

$$r = \left[[r_1]^{[t_0,t_1]}, [r_2]^{[t_1,t_2]}, \ldots, [r_n]^{[t_{n-1},t_n]}\right]^{[t_0,t_n]}$$

Each subgroup within a composite must have a lifetime wholly included within the lifetime of the parent group.

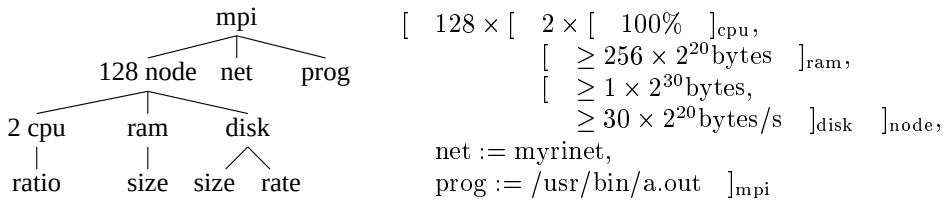

Figure 7. Hypothetical resource description. A parallel computer with 128 dedicated dual-processor nodes, each providing at least 256 MB of memory and 1 GB disk with disk performance of 30 MB/s, connected by Myrinet-enabled MPI. A parse tree is provided to help illustrate the nested expression.

5.3 Resource Alternatives

We define disjunctive *alternatives* to complement the conjunctive composites from section 5.2.

Alternative $\vee(r_1, r_2, ...)$ differs from a resource group in that only one element r_i must be satisfied, rather than all.

As indicated in the descriptions above, limit modifiers are only applicable to scalar metrics, while the alternative concept applies to all resource description elements. Alternatives can be used to express alternate solution spaces for the application requirements within distinct planning regimes, or to phrase similar requirements using basic and specialized metrics in the event that a client could benefit from unconventional extensions to R that may or may not be supported by a given manager.

5.4 Resource Configuration

The final feature present in our description language is the ability to intermingle control or *configuration* information within the resource statement. In a trusting environment, this intermingling is merely a notational convenience to avoid presenting two isomorphic statements—one modeling the requirements of the structured resource and one providing control data to the resource manager for the structured resource. Task configuration details are what are added to the language R to define the activity language J.

Configure $a := v$ specifies an arbitrary configuration attribute a should have value v.

Utilize use $:= M.I_r$ references an existing resource assignment required to implement the activity.

Apply bind $:= S.I_j$ references an existing activity submission for which an additional resource is being requested.

In an environment with limited trust and permissions, some resources may be unavailable for certain configurations due to arbitrary policy. We therefore treat them as primitive metrics when considering the meaning of the description for resource selection, while also considering them as control data when considering the meaning of the description as an activity configuration.

6 Implementing SNAP

The RM protocol architecture described in this article is general and follows a minimalist design principle in that the protocol captures only the behavior that is essential to the process of agreement negotiation. We envision that SNAP would not be implemented as a stand alone protocol, but in practice would be layered on top of more primitive protocols and services providing functions such as communication, authentication, naming, discovery, etc. For example, the Open Grid Services Architecture [18] defines basic mechanisms for creating, naming, and controlling the lifetime of services and in the following, we explore how SNAP could be implemented on top of the OGSA service model.

6.1 Authentication and Authorization

Because Grid resources are both scarce and shared, a system of rules for resource use, or *policy*, is often associated with a resource to regulate its use [39]. We assume a wide-area security environment such as GSI [19] will be integrated with the OGSA to provide mutually-authenticated identity information to SNAP managers such that they may securely implement policy decisions. Both upward information flow and downward agreement policy flow in Figure 8 are likely subject to policy evaluation that distinguishes between clients and/or requests.

6.2 Resource Heterogeneity

The SNAP protocol agreements can be mapped onto a range of existing local resource managers, to deploy its beneficial capabilities without requiring wholesale replacement of existing infrastructure. Results from GRAM testbeds have shown the feasibility of mapping submission agreements onto a range of local job schedulers, as well as simple

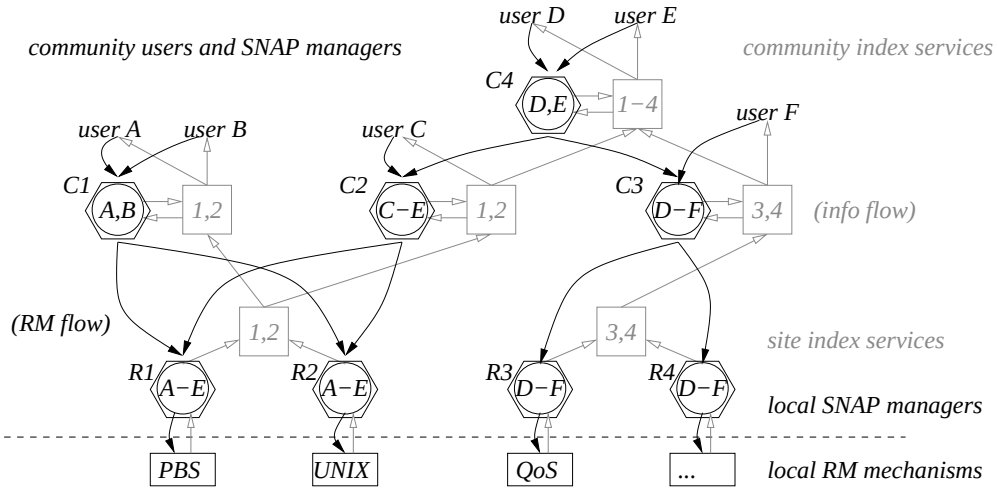

Figure 8. An integrated RM system. Discovery services provide indexed views of the resources to users, and RM services provide complementary access to resources.

time-sharing computers [15, 4, 33]. The GARA prototype has shown how resource capacity assignments and resource claiming can be mapped down to contemporary network QoS systems [21, 22, 34]. Following this model, SNAP manager services represent *adaptation* points between the SNAP protocol domain and local RM mechanisms.

6.3 Monitoring

A fundamental function for RM systems is the ability to monitor the health and status of individual services and requests. Existing Grid RM services such as GRAM and GARA include native protocol features to signal asynchronous state changes from a service to a client. In addition to these native features, some RM state information is available from a more generalized information service, e.g. GRAM job listings are published via the MDS in the Globus Toolkit [9, 21, 8].

We expect the OGSA to integrate asynchronous subscription/notification features. Therefore, we have omitted this function from the RM architecture presented here. An RM service implementation is expected to leverage this common infrastructure for its monitoring data path. We believe the agreement formalisms presented in Section 3.2 suggest the proper structure for exposing RM service state to information clients, propagating through the upward arrows in Figure 8. Information *index services* can cache and propagate this information because life-cycle of the agreement state records is well defined in the RM protocol semantics, and the nested request language allows detailed description of agreement properties.

6.4 Resource and Service Discovery

As described in Section 3.1, Grid RM relies on the ability for clients to discover RM services. We expect SNAP services to be discovered via a combination of general discovery and registry services such as the index capabilities of MDS-2 and OGSA, client configuration via service registries such as UDDI, and static knowledge about the community (Virtual Organization) under which the client is operating. The discovery information flow is exactly as for monitoring in Figure 8, with information propagating from resources upward through community schedulers and into clients. In fact, discovery is one of the purposes for a general monitoring infrastructure.

Due to the potential for virtualized resources described in Section 2.5, we consider "available resources" to be a secondary capability of "available services." While service environments provide methods to map from abstract service names to protocol-level service addresses, it is also critical that services be discoverable in terms of their capabilities. The primary capability of an RM service is the set of agreements it *offers*, i.e. that it is willing to establish with clients. As mentioned in Section 4.4, this set of agreements can be statically limited in the RM interface depending on what subset of submission, acquisition, or claiming negotiations are supported.

6.5 Multi-phase Negotiation

There are dynamic capabilities that also restrict the agreement space, including resource load and RM policy. Some load information may be published to help guide clients with their resource selection. However, proprietary policy including prioritization and provider SLAs may ef-

fect availability to specific classes of client.

The agreement negotiation itself is a discovery process by which the client determines the willingness of the manager to serve the client. By formulating future agreements with weak commitment and changing them to stronger agreements, a client is able to perform a multi-phase commit process to discover more information in an unstructured environment. Resource virtualization helps discovery by aggregating policy knowledge into a private discovery service—a community scheduler can form provisioning SLAs with application service providers and then expose this virtual resource pool through community-specific agreement offers.

6.6 Resource Language

In Section 5 we present the abstract requirements of an expressive resource language J. These requirements include unambiguous encoding of provisioning metrics, job configuration, and composites. We also identify above the propagation of resource and agreement state through monitoring and discovery data paths as important applications of the resource language. For integration with the OGSA, we envision this language J being defined by an XML-Schema [13] permitting extension with new composite element types and leaf types. The name-space features of XML-Schema permit unambiguous extension of the language with new globally-defined types.

This language serves the same purpose as RSL in GRAM/GARA [9, 10, 21, 22] or Class Ads in Condor [32, 27]. We suggest a more extensible model for novel resource composites than RSL and a more rigorously typed extension model than Class Ads, both of which we believe are necessary for large-scale, inter-operable deployments.

7 Agreements Represent Delegation

In the preceding protocol description, mechanisms are proposed to negotiate agreement regarding activity implementation or resource provisioning. These agreements capture a *delegation* of resource or responsibility between the negotiating parties. However, it is important to note that the delegation concept goes beyond these explicit agreements. There are analogous implicit delegations that also occur during typical RM scenarios.

The submission agreement delegates specific responsibilities to the scheduler that are held by the user. The scheduler becomes responsible for reliably planning and enacting the requested activity, tracking the status of the request, and perhaps notifying the user of progress or terminal conditions. The acquisition agreement delegates specific resource capacity to the user that are held by the manager. Depending on the implementation of the manager, this delegation might be mapped down into one or more hidden operational policy statements that enforce the conditions necessary to deliver on the guarantee. For example, a CPU reservation might prevent further assignments from being made or an internal scheduling priority might be adjusted to "steal" resources from a best-effort pool when necessary.

7.1 Delegation is Transitive

Transfers of rights and responsibilities are transitive in nature, in that an entity can only delegate that which is delegated to the entity. It is possible to form promise agreements about delegation out of order, but in order to exploit a delegation the dependent delegations must be valid. Such transitive delegation is limited by availability as well as trust between RM entities. A manager which over-commits resources will not be able to make good on its promises if too many clients attempt to use the resource delegations at the same time. Viewing capacity guarantees and submission as delegation simplifies the modeling of heavy-weight brokers or service providers, but it also requires a trust/policy evaluation in each delegation step. A manager may restrict its delegations to only permit certain use of the resource by a client—this client may attempt to broker the resource to other clients, but those clients will be blocked when they try to access the resource and the manager cannot validate the delegation chain.

7.2 Effects of Distribution

Collective resource scenarios are the key motivation for Grid RM. In our architecture, the local resource managers do not solve these collective problems. The user, or an agent of the user, must obtain capacity delegations from each of the relevant resource managers in a resource chain. There are a variety of brokering techniques which may help in this situation, and we believe the appropriate technique must be chosen by the user or community. The underlying Grid RM architecture must remain open enough to support multiple concurrent brokering strategies across resources that might be shared by multiple user communities.

Thus it is important to distinguish between the abstract delegation chain that always extends from end-user to resource and the particular instantiated delegations used in accounting or service guarantees. Furthermore, the decision of what delegations are hidden or publicized is a matter of information-access policy.

8 Other Related Work

Numerous researchers have investigated approaches to QoS delivery [23]and resource reservation for networks [11, 14, 40], CPUs [25], and other resources.

Proposals for advance reservations typically employ cooperating servers that coordinate advance reservations

along an end-to-end path [40, 14, 11, 24]. Techniques have been proposed for representing advance reservations, for balancing immediate and advance reservations [14], for advance reservation of predictive flows [11]. However, this work has not addressed the co-reservation of resources of different types.

The Condor high-throughput scheduler can manage network resources for its jobs. However, it does not interact with underlying network managers to provide service guarantees [2] so this solution is inadequate for decentralized environments where network admission-control cannot be simulated in this way by the job scheduler.

The concept of a bandwidth broker is due to Jacobson. The Internet 2 Qbone initiative and the related Bandwidth Broker Working Group are developing testbeds and requirements specifications and design approaches for bandwidth brokering approaches intended to scale to the Internet [37]. However, advance reservations do not form part of their design. Other groups have investigated the use of differentiated services (e.g., [42]) but not for multiple flow types. The co-reservation of multiple resource types has been investigated in the multimedia community: see, for example, [28, 30, 29]. However, these techniques are specialized to specific resource types.

The Common Open Policy Service (COPS) protocol [3] is a simple protocol for the exchange of policy information between a Policy Decision Point (PDP) and its communication peer, called Policy Enforcement Point (PEP). Communication between PEP and PDP is done by using a persistent TCP connection in the form of a stateful request/decision exchange. COPS offers a flexible and extensible mechanism for the exchange of policy information by the use of the client-type object in its messages. There are currently two classes of COPS client:

Outsourcing provides an asynchronous model for the propagation of policy decision requests. Messages are initiated by the PEP which is actively requesting decisions from its PDP.

Provisioning in COPS follows a synchronous model in which the policy propagation is initiated by the PDP.

COPS outsourcing maps well to the SNAP agreement model, but the provisioning model is not supported. SNAP resource managers run decoupled from the client—a client may affect the policy state of the manager, and the manager may notify the client of important events, but the manager never stops to request new policy from the client.

8.1 GRAM

The Globus Resource Allocation Manager (GRAM) provides job submission on distributed compute resources. It defines APIs and protocols that allow clients to securely instantiate job running agreements with remote schedulers [9]. In [10], we presented a light-weight, opportunistic broker called DUROC that enabled simultaneous co-allocation of distributed resources by layering on top of the GRAM API. This broker was used extensively to execute large-scale parallel simulations, illustrating the challenge of coordinating computers from different domains and requiring out-of-band resource provisioning agreements for the runs [5, 4]. In exploration of end-to-end resource challenges, this broker was more recently used to acquire clustered storage nodes for real-time access to large scientific datasets for exploratory visualization [7].

8.2 GARA

The General-purpose Architecture for Reservation and Allocation (GARA) provides advance reservations and end-to-end management for quality of service on different types of resources, including networks, CPUs, and disks [21, 22]. It defines APIs that allows users and applications to manipulate reservations of different resources in uniform ways. For networking resources, GARA implements a specific network resource manager which can be viewed as a bandwidth broker.

In [34], we presented a bandwidth broker architecture and protocol that addresses the problem of diverse trust relationships and usage policies that can apply in multi-domain network reservations. In this architecture, individual BBs communicate via bilaterally authenticated channels between peered domains. Our protocol provides the secure transport of requests from source domain to destination domain, with each bandwidth broker on the path being able to enforce local policies and modify the request with additional constraints. The lack of a transitive trust relation between source- and end-domain is addressed by a delegation model where each bandwidth broker on the path being able to identify all upstream partners by accessing the credentials of the full delegation chain.

9 Conclusions

We have presented a new model and protocol for managing the process of negotiating access to, and use of, resources in a distributed system. In contrast to other architectures that focus on managing particular types of resources (e.g., CPUs or networks), our Service Negotiation and Acquisition Protocol (SNAP) defines a general framework within which reservation, acquisition, task submission, and binding of tasks to resources can be expressed for any resource in a uniform fashion.

We have not yet validated the SNAP model and design in an implementation. However, we assert that these ideas have merit in and of themselves, and also note that most have already been explored in limited form within the current GRAM protocol and/or the GARA prototype system.

Acknowledgments

We are grateful to many colleagues for discussions on the topics discussed here, in particular Jeff Frey, Steve Graham, Bill Johnston, Miron Livny, Jeff Nick, and Alain Roy. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38; by the National Science Foundation; by the NASA Information Power Grid program; and by IBM.

References

[1] SOAP version 1.2 part 0: Primer. W3C Working Draft 17. www.w3.org/TR/soap12-part0/.

[2] J. Basney and M. Livny. Managing network resources in Condor. In *Proc. 9th IEEE Symp. on High Performance Distributed Computing*, 2000.

[3] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) protocol. IETF RFC 2748, January 2000.

[4] S. Brunett, K. Czajkowski, S. Fitzgerald, I. Foster, A. Johnson, C. Kesselman, J. Leigh, and S. Tuecke. Application experiences with the Globus toolkit. In *HPDC7*, pages 81–89, 1998.

[5] S. Brunett, D. Davis, T. Gottschalk, P. Messina, and C. Kesselman. Implementing distributed synthetic forces simulations in metacomputing environments. In *Proceedings of the Heterogeneous Computing Workshop*, pages 29–42. IEEE Computer Society Press, 1998.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. Technical report, W3C, 2001. http://www.w3.org/TR/wsdl/.

[7] K. Czajkowski, A. K. Demir, C. Kesselman, and M. Thiebaux. Practical resource management for grid-based visual exploration. In *Proc. 10th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 2001.

[8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proc. 10th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 2001.

[9] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[10] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational grids. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1999.

[11] M. Degermark, T. Kohler, S. Pink, and O. Schelen. Advance reservations for predictive service in the internet. *ACM/Springer Verlag Journal on Multimedia Systems*, 5(3), 1997.

[12] P. Dinda. Online prediction of the running time of tasks. In *Proc. 10th IEEE Symp. on High Performance Distributed Computing*, (to appear) 2001.

[13] D. Fallside. XML schema part 0: Primer. Technical report, W3C, 2001. http://www.w3.org/TR/xmlschema-0/.

[14] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. *ACM/Springer Verlag Journal on Multimedia Systems*, 5(3), 1997.

[15] I. Foster and C. Kesselman. The Globus project: A status report. In *Proceedings of the Heterogeneous Computing Workshop*, pages 4–18. IEEE Computer Society Press, 1998.

[16] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, 1999.

[17] 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 *Proceedings of the International Workshop on Quality of Service*, 1999.

[18] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. Technical report, Globus Project, 2002. www.globus.org/research/papers/ogsa.pdf.

[19] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *ACM Conference on Computers and Security*, pages 83–91. ACM Press, 1998.

[20] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Intl. Journal of High Performance Computing Applications*, 15(3):200–222, 2001. http://www.globus.org/research/-papers/anatomy.pdf.

[21] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Res ource Reservation and Application Adaptation. In *International Workshop on Quality of Service*, 2000.

[22] I. Foster, A. Roy, V. Sander, and L. Winkler. End-to-End Quality of Service for High-End Applications. Technical report, Argonne National Laboratory, Argonne, 1999. http://www.mcs.anl.gov/qos/qos_papers.htm.

[23] R. Guérin and H. Schulzrinne. Network quality of service. In *[16]*, pages 479–503.

[24] A. Hafid, G. Bochmann, and R. Dssouli. A quality of service negotiation approach with future reservations (nafur): A detailed study. *Computer Networks and ISDN Systems*, 30(8), 1998.

[25] H. hua Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 296–301. IEEE Computer Society Press, June 1999. Florence, Italy.

[26] C. Lai, G. Medvinsky, and B. C. Neuman. Endorsements, licensing, and insurance for distributed system services. In *Proc. 2nd ACM Conference on Computer and Communication Security*, 1994.

[27] M. Livny. Matchmaking: Distributed resource management for high throughput computing. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, 1998.

[28] A. Mehra, A. Indiresan, and K. Shin. Structuring communication software for quality-of-service guarantees. In *Proc. of 17th Real-Time Systems Symposium*, December 1996.

[29] K. Nahrstedt, H. Chu, and S. Narayan. QoS-aware resource management for distributed multimedia applications. *Journal on High-Speed Networking, IOS Press*, December 1998.

[30] K. Nahrstedt and J. M. Smith. Design, implementation and experiences of the OMEGA end-point architecture. *IEEE JSAC, Special Issue on Distributed Multimedia Systems and Technology*, 14(7):1263–1279, September 1996.

[31] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *The IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, June 2002.

[32] R. Raman, M. Livny, and M. Solomon. Resource management through multilateral matchmaking. In *Proc. 9th IEEE Symp. on High Performance Distributed Computing*, 2000.

[33] L. Rodrigues, K. Guo, P. Verissimo, and K. Birman. A dynamic light-weight group service. *Journal on Parallel and Distributed Computing*, (60):1449–1479, 2000.

[34] V. Sander, W. A. Adamson, I. Foster, and A. Roy. End-to-End Provision of Policy Information for Net work QoS. In *Proc. 10th IEEE Symp. on High Performance Distributed Computing*, 2001.

[35] W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[36] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, pages 268–278, 1998.

[37] B. Teitelbaum, S. Hares, L. Dunn, V. Narayan, R. Neilson, and F. Reichmeyer. Internet2 QBone - Building a testbed for differentiated services. *IEEE Network*, 13(5), 1999.

[38] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, and C. Kesselman. Grid services specification. Technical report, Globus Project, 2002. www.globus.org/research/-papers/gsspec.pdf.

[39] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. Aaa authorization application examples. Internet RFC 2905, August 2000.

[40] L. Wolf and R. Steinmetz. Concepts for reservation in advance. *Kluwer Journal on Multimedia Tools and Applications*, 4(3), May 1997.

[41] R. Wolski. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, 1997.

[42] I. Yeom and A. L. N. Reddy. Modeling tcp behavior in a differentiated-services network. Technical report, TAMU ECE, 1999.