Chapter 1

GRID SERVICE LEVEL AGREEMENTS

Grid Resource Management with Intermediaries

Karl Czajkowski,[1] Ian Foster,[2,3] Carl Kesselman,[1] and Steven Tuecke[2]

[1] *Information Science Institute, University of Southern California*
[2] *Mathematics and Computer Science Division, Argonne National Laboratory*
[3] *Department of Computer Science, The University of Chicago*

Abstract We present a reformulation of the well-known GRAM architecture based on the Service-Level Agreement (SLA) negotiation protocols defined within the Service Negotiation and Access Protocol (SNAP) framework. We illustrate how a range of local, distributed, and workflow scheduling mechanisms can be viewed as part of a cohesive yet *open* system, in which new scheduling strategies and management policies can evolve without disrupting the infrastructure. This architecture remains neutral to, and in fact strives to mediate, the potentially conflicting resource, community, and user policies.

1. INTRODUCTION

A common requirement in distributed computing systems such as Grids [FK99, FKT01] is to coordinate access to resources located within different administrative domains than the requesting application. The coordination of Grid resources is complicated by the frequently competing needs of the resource consumer (or *application*) and the *resource owner*. The application needs to understand and affect resource behavior and often demands assurances as to the level and type of service being provided by the resource. Conversely, the resource owner may want to maintain local control and discretion over how their resource(s) are used.

A common means of reconciling these two competing demands is to establish a procedure by which the parties can negotiate a *service-level agreement* (SLA) that expresses a resource provider *contract* with a client to provide some measurable capability or to perform a specified task. An SLA allows a client

to understand *what to expect from resources* without requiring detailed knowledge of competing workloads nor of resource owners' policies.

The Globus resource management architecture that we present here is based on the notion of negotiation of SLAs between a client and a resource provider. A resource provider can be directly associated with a resource, or alternatively may be a service that virtualizes multiple resources, i.e., a *broker* or *super-scheduler* in which case negotiation proceeds hierarchically. The SLAs are defined such that it is meaningful to construct hierarchies of negotiating entities acting as both providers and clients.

We have previously explored Grid resource management methods in our work with the Grid Resource Allocation and Management (GRAM) [CFK+98b] component of the Globus Toolkit, which supports remote access to job-submission systems; DUROC [CFK99], a co-allocation system that uses GRAM functions to compose jobs that use more than one resource; and the General-purpose Architecture for Reservation and Allocation (GARA) [FRS00, FFR+02], which extends GRAM to support immediate and advance reservation. Building on these experiences, and in particular on the decoupling of reservation and task creation introduced in GARA, we defined the Service Negotiation and Access Protocol (SNAP) framework [CFK+02], an abstract architecture that defines operations for establishing and manipulating three distinct types of SLA, as follows:

1 *Resource service level agreements* (RSLAs) that represent a commitment to provide a resource when claimed by a subsequent SLA.

2 *Task service level agreements* (TSLAs) that represent a commitment to perform an activity or task with embedded resource requirements.

3 *Binding service level agreements* (BSLAs) that represent a commitment to apply a resource to an existing task, i.e. to extend the requirements of a task after submission or during execution.

As illustrated in Figure 1.1, these three kinds of SLA support an interactive resource management model in which one can independently submit tasks to be performed, obtain promises of capability, and bind a task and a capability. These three types of agreement can be combined in different ways to represent a variety of resource management approaches including batch submission, resource brokering, adaptation, co-allocation, and co-scheduling.

Our presentation here first reviews the SNAP design and then describes a concrete realization of the framework in the GRAM-2 next-generation Globus resource management architecture. The GRAM-2 design addresses various technical issues that arise when the SNAP protocol building blocks are incor-

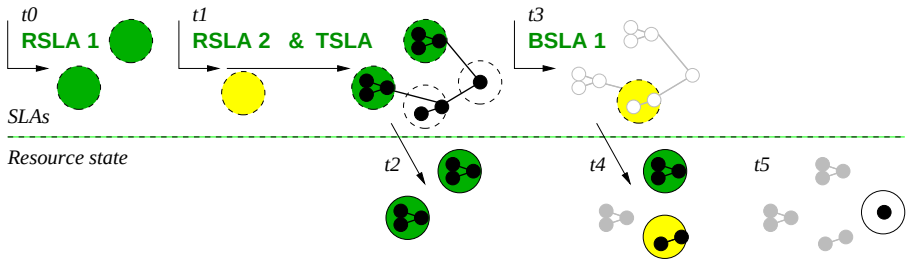


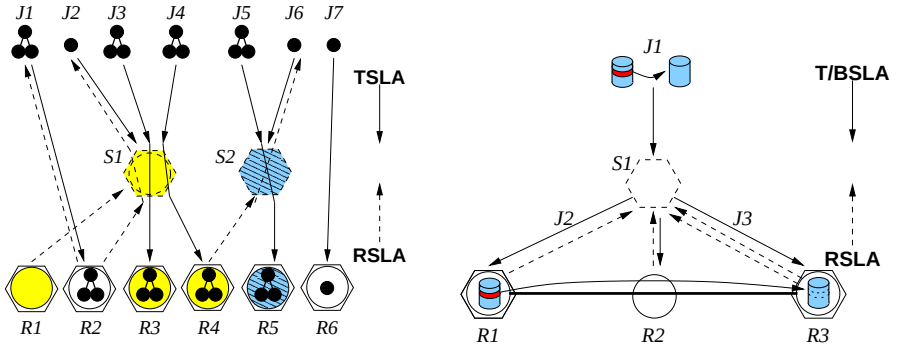
Figure 1.1. Three types of SLA—RSLA, TSLA, and BSLA—allow a client to schedule resources as time progresses from t_0 to t_5. In this case, the client acquires two resource promises (RSLAs) for future times; a complex task is submitted as the sole TSLA, utilizing RSLA 1 to get initial portions of the job provisioned; later, the client applies RSLA 2 to accelerate execution of another component of the job via BSLA 1; finally, the last piece of the job is provisioned by the manager according to the original TSLA.

porated into an operational resource management system. It provides a complete solution to the problem of hierarchical negotiation of SLAs between a resource consumer and a resource provider. (A resource provider can be directly associated with a resource, or alternatively may be a service that virtualizes multiple resources, e.g., a broker or scheduler.)

2. MOTIVATING SCENARIOS

The SNAP SLA model is designed to address a broad range of applications through the aggregation of simple SLAs. We motivate its design by examining three scenarios: a Grid in which *community schedulers* mediate access to shared resources on behalf of different client groups; a file-transfer service that uses advance reservations to perform data staging under deadline conditions; and a job-staging system that uses co-allocation to coordinate functions across multiple resource managers.

The schedulers in our scenarios are all examples of a class of resource management intermediaries that are variously referred to as brokers, agents, distributed schedulers, meta-schedulers, or super-schedulers [FRS00, MBHJ98]. What distinguishes each kind of intermediary are the policies that are supported between users, intermediaries, and underlying resource schedulers. For example, an intermediary may support a large community of users (e.g., a typical resource broker), or act on behalf of a single user (an agent). A similarly wide range of policies can exist between the intermediary and its resource(s)—some intermediaries may have exclusive access to resources while others may have no more rights than a typical user. The intermediary may not even be a distinguished entity for policy but instead may simply act via rights delegated from the client.



(a) Community scheduler scenario. Community schedulers (S1–S2) mediate access to the resources (R1–R6) by their users (J2–J6) through resale of SLAs suitable to community criteria.

(b) File transfer scenario. Transfer scheduler coordinates disk reservation before co-scheduling transfer endpoint jobs to perform transfer jobs for clients.

Figure 1.2. SLA architecture scenarios. Persistent intermediate scheduling services form SLAs with users and underlying resources to help coordinate user activity in the Grid. This supports scalable negotiation and also identifies points where mapping from one request or policy language to another is likely to occur.

2.1 Community Scheduler Scenario

A community scheduler is an entity that acts as a policy-enforcing intermediary between a community of users and a set of resources. Activities are submitted to the community scheduler rather than to end resources. The scheduler then works to schedule those activities onto community resources so as to meet community policies regarding use of the resource set: for example, to optimize response time or throughput, or to enforce allocations. We are thus faced with a two-tiered SLA negotiation process, from client to scheduler and then from scheduler to client. The strictness of the policy environment in enforcing this multi-tier SLA negotiation will affect the predictability and efficiency of the resulting schedules. Even in an open resource environment in which the community scheduler is easily bypassed by aggressive clients, lightweight clients may benefit from the sharing of resource discovery processes performed by the scheduler.

As depicted in Figure 1.2(a), a Grid environment may contain many resources (R1–R6), all presenting both an RSLA and a TSLA interface. First, the scheduler negotiates capacity guarantees (via RSLAs) with its underlying resources. With these capacity guarantees in hand, it can then negotiate RSLAs or TSLAs with its clients, fulfilling its commitments by negotiating further

SLAs with resources to map the requested user activities to the previously negotiated capacity. Depending on the community, workload, and other factors, the scheduler may variously negotiate capacity before receiving user requests (as suggested in the preceding discussion), or alternatively, may do so only *after* receiving requests. In either case, the ability to negotiate agreements with underlying resources abstracts away the impact of other community schedulers as well as any non-Grid local workloads, assuming the resource managers will enforce SLA guarantees at the individual resources.

Community scheduler services (S1 and S2 in Figure 1.2(a)) present a TSLA interface to users. Users in this environment interact with community and resource-level schedulers as appropriate for their goals and privileges. The privileged client with job J7 in Figure 1.2(a) may not need RSLAs nor the help of a community scheduler, because the goals are expressed directly in the TSLA with resource R6. The client with job J1 acquires an RSLA from R2 in anticipation of its requirements and utilizes that reservation in a TSLA.

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 across the shared resources. Note that R2 is running job J1 while guaranteeing future availability to S1 (which is in turn guaranteeing J2 a place to run based on that reservation). Similarly, R4 is running job J4 from S1 while guaranteeing a future slot to J6 by way of S2. Scheduler S1 also maintains a speculative RSLA with R1 to more rapidly serve future high-priority job requests.

2.2 File Transfer Service Scenario

We next consider a scenario in which the user activities of interest are concerned with the transfer of a file from one storage system to another. A transfer requires multiple resources—storage space on the destination resource, plus network and endpoint I/O bandwidth during the transfer—and thus the scheduler needs to manage multiple resource types and perform co-scheduling of these resources through their respective managers.

As depicted in Figure 1.2(b), the file transfer scheduler S1 presents a TSLA interface, storage systems provide TSLA/RSLA interfaces, and a network resource manager R2 presents an RSLA/BSLA interface. A user submitting a transfer job (J1) to the scheduler negotiates a TSLA that includes a deadline. The scheduler works to meet this deadline by: (1) obtaining a storage reservation on the destination resource R3 to ensure that there will be space for the data; (2) obtaining bandwidth reservations from the network and the storage devices, giving the scheduler confidence that the transfer can be completed within the user-specified deadline; (3) submitting transfer endpoint jobs J2 and J3 to implement the transfer using the previously negotiated space and band-

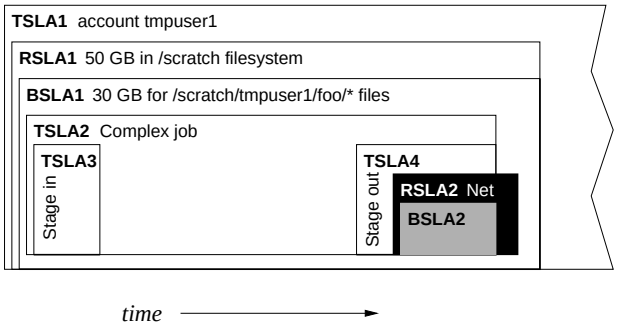


Figure 1.3. Dependent SLAs for file transfers associated with input and output of a job with a large temporary data space. BSLA2 is dependent on TSLA4 and RSLA2, and has a lifetime bound by those two. All job components depend on an outermost account sandbox assigned temporarily for the purpose of securely hosting this job.

width promises; and finally, establishing a BSLA with R2 to utilize the network reservation for the sockets created between J1 and J2.

2.3 Job Staging with Transfer Service Scenario

SLAs can be linked to address more complex resource co-allocation situations. We illustrate the use of linking by considering a job that consists of a sequence of three activities: data is transferred from a storage system to an intermediate location; some computation is performed using the data; and the result is transferred to a final destination. Such a sequence would typically be treated monolithically by the job system, but this approach is inappropriate when data transfers involve significant resource utilization that spans resource domains, as in the previous data transfer scenario, in which source and destination storage are under separate control.

As in the other examples, the computation in question is to be performed on resources allocated to a community of users. However, for security reasons, the computation is not performed using a group account, but rather, a temporary account is dynamically created for the computation. The SLA model facilitates the decomposition of staging and computation activities that is required for these functions to be integrated with dynamic account management functions.

In Figure 1.3, TSLA1 results from a negotiation with the resource to establish a temporary user account, such as might be established by a resource for a client who is authorized through a Community Authorization Service [PWF+02]. All job interactions performed by that client on the resource become linked to this long-lived TSLA, as in order for the account to be reclaimed safely, all dependent SLAs must be destroyed. The figure illustrates how the individual SLAs associated with resources and tasks can be combined

to address the end-to-end resource and task management requirements of the entire job. Of interest in this example are:

TSLA1 is the TSLA negotiated to establish the above-mentioned temporary user account.

RSLA1 promises the client 50 GB of storage in a particular file-system on the resource.

BSLA1 binds part of the promised storage space to a particular set of files within the file-system.

TSLA2 runs a complex job that will subsequently spawn subjobs for staging input and output data.

TSLA3 is the TSLA negotiated for the first file transfer task, which stages the input to the job site (without requiring any additional QoS guarantees in this case).

TSLA4 is the TSLA negotiated for the second file transfer task, to stage the large output from the job site, under a deadline, before the local file-system space is lost.

RSLA2 and BSLA2 are used by the file transfer service to achieve the additional bandwidth required to complete the (large) transfer before the deadline.

The job for which TSLA2 is negotiated might have built-in logic to establish the staging jobs TSLA3 and TSLA4, or this logic might be incorporated within the entity that performs task TSLA2 on behalf of the client. In Figure 1.3, the nesting of SLA "boxes" is meant to illustrate how the lifetime of these management abstractions is linked in practice. Such linkage can be forced by a dependency between the subjects of the SLAs, e.g., BSLA2 is meaningless beyond the lifetime of TSLA4 and RSLA2, or alternatively can be added as a management convenience, e.g., by triggering recursive destruction of all SLAs from the root to hasten reclamation of application-grouped resources.

3. RESOURCE VIRTUALIZATION THROUGH INTERMEDIARIES

The Community Scheduler introduced above can be viewed as virtualizing a set of resources for the benefit of its user community. Resource virtualiza-

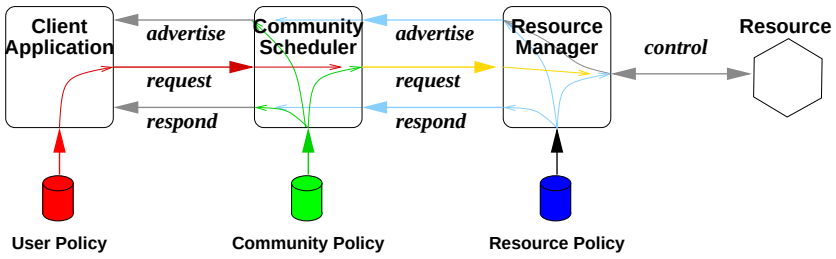


Figure 1.4. SLA negotiation with intermediaries. A negotiation pipeline between a user, community scheduler, and resource manager permits policies to be introduced at each stage which affect the outcome and are illustrated using color mixing. User policy affects what requests are initiated, community policy affects how user requests are mapped to resource-level requests, and resource policy affects how resources may be utilized. Thus policy from each source mixes into the stream of requests going to the right, and into the streams of advertisements and request-responses going to the left.

tion can serve to abstract details of the underlying resources or to map between different resource description domains. In our initial community scheduler example, the scheduler provides the same sort of resource and task description as the underlying resources, only insulating the user community from task-placement decisions. However, with the file transfer scenario, the scheduler accepts requests in a more application-level file transfer description language. In this case, the scheduler insulates the user community from the more complicated resource interactions necessary to implement a file transfer with deadline guarantees in a distributed environment.

Each such intermediary scheduler delineates a boundary between resource domains and may map from more abstract user terminology to underlying resource mechanisms, as well as bridging policy domains. Such mappings can complicate the Grid resource management problem, as important scheduling information can be lost. However, we believe that there is also the opportunity to introduce intuitively-framed policies at each such boundary.

Figure 3 illustrates how policies from different domains mix into SLA negotiation with intermediaries, each of which can potentially contribute to resource management decisions. Community policies may affect relative priority of different user tasks with a community scheduler. At the resource, owner policies may affect relative priority of different communities. In order to implement the community policy, the community scheduler must negotiate dynamic policies (SLAs) to differentiate tasks on the same physical resources.

4. UNDERSTANDING SERVICE LEVEL AGREEMENTS

The preceding sections describe the role of automated intermediaries in negotiating SLAs for complex Grid scenarios. Automated SLA negotiation requires that we be able to represent, in machine-processable terms, what is *offered* by a service and what is *desired* or requested by a client. Following discovery and negotiation, our three kinds of SLA represent what is to be *performed* by the resources.

This leads to several questions. If all three agreements capture what is to be performed, what distinguishes the three kinds of agreement? How are the SLAs represented? What does it mean for a scheduler to agree or "promise" to do something? In the remainder of this section, we answer these questions.

4.1 Different Kinds of Agreement

There are three promises we can capture in our SLAs. Resource managers can promise that a resource will be available for future use, promise that a resource will be consumed in a certain manner, and promise that a certain task will be performed. The three kinds of agreement introduced in Section 1 capture important combinations of these promises. What makes these combinations important is how they provide for the multiple negotiation phases exploited in our scenarios.

All three kinds of SLA provide a promise of future resource availability. What distinguishes RSLAs from the others is that *RSLAs only promise resource availability* without any associated plan for utilization. As illustrated in Figure 1.5, the remaining two kinds of SLA are formulated by the introduction of the other promises. Our *BSLAs add a plan for resource utilization*, without performing any new tasks, and *TSLAs capture all three promises at once*. There is not a fourth kind of SLA capturing a task promise without utilization. Such an agreement would not be meaningful, because performing a task requires resource utilization.

In the simplest case, these SLAs are negotiated in sequential order, as specified above. More generally, there is a partial order based on references made in the resulting SLAs. For example, an RSLA may be referenced in a TSLA, or it may only be referenced in a subsequent BSLA that augments the task. In the latter case, the TSLA and RSLA can be negotiated in any order. General many-to-many references are meaningful, though the range of possible scenarios may be limited by policy in a given negotiation.

RSLAs correspond to (immediate or advance) reservations, i.e. they represent a promise that can be employed in future SLA negotiation. No RSLA has any effect unless it is claimed through a follow-up TSLA or BSLA negotiation. This split-phase negotiation is useful when attempting to synchronize

interactions with multiple resource providers, because it allows a client to obtain commitments for future availability of capability before details of the use of that capability have been decided [DKPS97, FGV97, HvBD98, FRS00].

By binding a resource reservation to a task, a BSLA allows for control of the allocation of resources to tasks (*provisioning*), independent of task creation. Negotiation of a BSLA does not initiate any task: the task must be created separately (either before or after the BSLA, depending on the task naming mechanisms employed in BSLA). This decoupled provisioning can be used to provision a task created outside the SLA negotiation framework, e.g., a network socket or local UNIX process created through interactive mechanisms. BSLA negotiation can also be used to separate the provisioning decisions from basic task management with TSLAs; if the task has variable requirements, these can be expressed by shorter duration BSLAs associated with a long-lived TSLA that only represents the baseline requirements of the task. A BSLA can add to, but not diminish, the resource allocation requirements expressed during task creation.

Finally, the negotiation of an TSLA represents a commitment by a resource manager (task scheduler) to perform the described task with the described levels of resource provisioning. In all cases, there is a need for complex SLA meta-data to denote whether or how implied requirements are addressed.

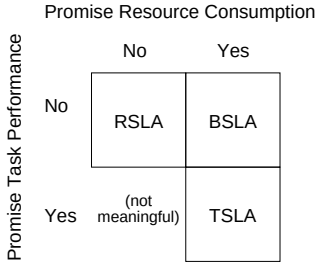


Figure 1.5. Three kinds of SLA. All three SLAs promise resource availability, but TSLAs and BSLAs add additional promises of task completion and/or resource utilization.

4.2 Representing SLAs

To build a system using these three kinds of SLA, we need to represent them in some machine-processable form. For integration with XML-based systems such as Web Services [CCMW01] or Grid Services [TCF+03], we would want XML schema definitions for the SLA content. However, these definitions are cumbersome to present and should be developed in a community standards body.

We can describe the content of the SLAs, both to help envision the scope of the terms and for input into standards processes. Our descriptions make use of the following elemental descriptions:

- *SLA references*, which allow the newly negotiated SLA to be associated with pre-existing SLAs;
- *Resource descriptions*, which are the main subject of RSLA negotiations and may also appear within a TSLA or BSLA (potentially accompanied by RSLA references);
- *Resource metadata*, which qualify the capability with time of availability, reliability, etc.;
- *Task descriptions*, which are the main subject of TSLA negotiations and may also appear within a BSLA;
- *SLA metadata*, which qualify the SLA content with degree of commitment (see Section 4.2), revocation policies, SLA lifetime, etc.

The content of an RSLA includes resource descriptions and metadata, as well as SLA metadata. The description captures resource capabilities such as storage space, nodes in a multicomputer, or processing throughput during a certain interval of time. The SLA metadata might capture the level of commitment *promised* by the resource manager to the client.

The content of a BSLA includes TSLA references or task descriptions, resource descriptions, optional RSLA references, and SLA metadata. The TSLA references or task descriptions identify tasks which will consume resources. The resource descriptions describe what resources will be provided to the tasks and the RSLA references identify existing resource promises to be exploited. Example tasks might be job processes, network flows, or filesystem accesses.

The content of a TSLA includes a task description, resource descriptions, optional RSLA references, and SLA metadata. The task description describes what task will be completed. The resource description describes requirements of the task and the RSLA references identify existing resource promises to be exploited.

The ability to negotiate SLAs can be beneficial regardless of how much credence one is able to put in the agreements. At one extreme, an SLA may represent simply a guess as to what may be possible; at another, it may be accompanied by a strict contractual agreement with defined financial penalties for noncompliance. Even if this degree of commitment is formalized in SLA meta-data, it is always possible for a manager to be error-prone or untrustworthy. Licensing mechanisms might be used to allow users to judge which managers are to be trusted [LMN94].

5. SLA CONSTRAINT-SATISFACTION MODEL

The SNAP model is concerned primarily with the protocols used to negotiate SLAs. However, it is also useful to provide some informal discussion

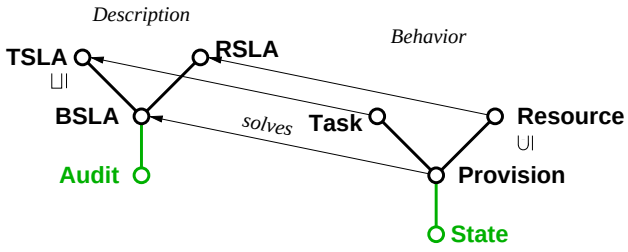


Figure 1.6. Constraint domain. Lower items in the figure conservatively approximate higher items. The solution spaces on the right are ordered as subsets, e.g., Provisioning⊆Reserves because provisioning constrains a resource promise to a particular task. Solution ordering maps to the model relation for constraints, e.g., BSLA⊑RSLA on the left.

of the semantics associated with SLA negotiation. Given a particular task description and resource description language, the purpose of a resource provider (whether resource owner or scheduler) is to attempt to *satisfy* SLAs specified in requests. An SLA is satisfied if the resource provider can produce a non-empty solution set of possible resource and task schedules that delivers the capabilities and performs the directives encoded in the SLA content. A self-contradictory or unsatisfiable SLA has an empty solution set. We have previously explained [CFK+02] how satisfaction of SLA terms by resource behavior is related to the notion of problem solving by refinement of plans. As application goals are translated through a sequence of intermediaries, the terms are refined until a concrete resource schedule is reached and application goals satisfied.

A TSLA says that a manager will run a job according to its self-expressed performance goals and provisioning requirements. A RSLA says that a manager will provide a resource capability when asked by the client. A corresponding BSLA encompasses both of these agreements and says the manager will apply a resource to help satisfy requirements while performing a job.

5.1 Descriptive and Behavioral Concept Domains

Figure 1.6 illustrates this ordering of SNAP concepts in terms of refinement and satisfaction. Descriptive or behavioral elements at the bottom of the figure are more concrete realizations of the concepts connected above them in the figure. This domain diagram partially orders abstract concepts by vertical position, with more abstract concepts on top. Concepts are only ordered if there is a connecting path of lines representing a transitive ordering relationship. The thick lines between the right-hand *behavioral* concepts represent the subset relationship, e.g., actual resource states are a subset of possible states fitting a provisioning plan (State⊆Provision) and a concrete provisioning solutions are a subset of task solutions.

The left-hand SLA terms in Figure 1.6 are ordered by the *satisfaction* relationship that we introduced previously [CFK+02], in which, for example, the more specific BSLA terms can be thought to satisfy the more general TSLA or RSLA terms while also introducing new details absent from the more general SLAs. Note the interesting parallel between the two halves of the figure, shown by thin arrows that link behavioral concepts to descriptive terms. The behavioral concept *solves* a constraint specified with the descriptive terms. For a particular BSLA b and TSLA t, there are corresponding provisioning and task solution sets $S_{\mathbf{B}}(b)$ and $S_{\mathbf{T}}(t)$, respectively, such that $b \sqsubseteq t$ if and only if $S_{\mathbf{B}}(b) \subseteq S_{\mathbf{T}}(t)$.

6. APPROACHES TO IMPLEMENTATION

The SNAP architectural model serves to frame our understanding of existing resource management mechanisms and how they fit together in a Grid environment. Adoption of the abstract model does not require much change to existing systems, though it does help identify limitations and policy-biased features of existing systems. Just as GRAM [CFK+98b] *adapts* localized schedulers to a Grid-wide protocol, we believe that the SLA-based SNAP architecture can be deployed as GRAM-2 with adapters to local schedulers. However, the best implementation approach would be for vendors to natively support new interoperable negotiation protocols in the local schedulers.

6.1 Toward Standards for Interoperability

Within the Global Grid Forum, the GRAAP working group is chartered to standardize negotiation protocols in the form of Open Grid Service Architecture *portTypes* (interfaces). Further work needs to be chartered to produce SLA content standards.

GRAM-2 implementation is underway using similar but proprietary portTypes in anticipation of this work; we will migrate to use standard interfaces when they are available. Multiple commercial scheduler vendors are interested in implementing GRAAP interfaces—and interim Globus GRAM-2 interfaces while GRAAP standardization progresses.

Multiple products, including Globus Toolkit 3 (GT3), will provide OGSA infrastructure implementation within which a scheduler implementation can be hosted.

6.2 SLA Implementation Through Policy Mapping

A TSLA transfers specific task-completion responsibilities from the user to a manager. The scheduler then 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. A RSLA similarly

delegates specific resource capacity from a manager to a user. The manager might implement this delegation via hidden operational policy statements that enforce the conditions necessary to deliver on the guarantee. For example, a CPU reservation might prevent further reservations from being made, or an internal scheduling priority might be adjusted to steal resources from a best-effort pool when necessary.

Tasks may make resource requests dynamically during their execution: for example, I/O requests or low-level memory allocation requests. Thus, we can configure the task-resource binding expressed by a BSLA so that task resource requests are interpreted as claims against the RSLA promise. In the general case, a BSLA binding may include capability descriptions to restrict what claims can be made, as well as more fine-grained resource-to-task mapping information. When such restriction and mapping information is expressible, it is possible to create complex many-to-many binding relationships between RSLAs and TSLAs without ambiguity or over-subscription to capabilities.

In the face of ambiguous or over-committed binding, fall-back policies must still resolve conflicts at runtime. Traditional first-come, first-serve or fair-share job schedulers can be seen as implementing such fall-back policies in an environment where every TSLA is bound to the same machine-wide RSLA. The addition of fine-grained binding information simply partitions these conflicts into smaller logical resource domains, thus allowing the scheduler to be guided with application and administrator goals.

6.3 Security Considerations

Whether SLA guarantees are enforced via such policy-mapping or not, the negotiation of SLAs is easily seen as a form of distributed policy management. As such, work needs to be started to get SLA proponents involved with existing security standards activities.

With the Global Grid Forum, the OGSA working group is chartered to guide narrowly-targeted groups providing OGSA-relevant standards. The OGSA-SEC working group is specifically chartered to develop approaches complementary of basic Web Service standards coming from other communities such as W3C and OASIS.

7. RELATED WORK

Resource management in networks and wide area computing systems is receiving increased attention (for overviews, see [Ber99, GS99]). However, there has been little work on the specific problems addressed here, namely general-purpose architectural constructs for reservation and co-allocation of heterogeneous collections of resources. Here we review briefly some relevant work; space constraints prevent a complete survey.

Various proposals have been made for advance reservations in the Internet [WS97, FGV97, DKPS97, HvBD98, BL98]. These capabilities are typically encapsulated into the function of the network, in the form of cooperating sets of servers that coordinate advance reservations along an end-to-end path. Such efforts have proposed solutions for various network reservation challenges, but do not address problems that arise when an application requires co-allocation of multiple resource types.

The theory of co-allocation is well understood, and sophisticated techniques exist for determining resource requirements (e.g., identifying the CPU and network resources required to meet application QoS requirement [MIS96, NS96, NCN98]) and for scheduling scarce resources in the face of competing demands. However, the mechanics of co-allocation in a distributed computing environment have received less attention. RSVP [BZB+97] and Beagle [CFK+98a] can be used to signal resource QoS requirements through a network, but they focus on networks and do not address directly how to discover and select (perhaps under application control) appropriate resources from potentially large populations.

Multimedia applications have motivated techniques for allocating both memory and CPU for channel handlers [MIS96] and CPU, bandwidth, and other resources for video streams [NS96, NCN98]. However, these specialized approaches do not extend easily to other resource types. Many approaches deal with collection of network resources only [FGV97, DKPS97, WG98]. Other related work presents generic methods which could be used for heterogeneous resource types [HvBD98, SP98].

The policy issue is investigated within admission control [WG98] and resource sharing [SP98]. For example, in [WG98] effective admission control policy is proposed for booking ahead network services. Admission control is based on a novel application of effective bandwidth theory to the time domain.

In decentralized, wide area systems, a lack of exclusive control of resources, reduced resource reliability, and a larger resource base from which to select candidate resources introduces the problem of co-allocating multiple resources while individual allocation requests may fail. We believe that effective strategies for discovering alternative resources subject to policy variation and reservation failure will be highly application specific, and any solution that embeds this strategy into the basic infrastructure will fail to meet QoS requirements.

The Darwin project at CMU has built a system with some similarities to SNAP [CFK+98a]. However, Darwin deals with network resources only. It assumes that the network is controlled via Darwin-specific protocols (i.e., Beagle) and hence does not accommodate independently administered resources.

In summary, previous work has not focused on the integration of heterogeneous collections of locally administered resources. Furthermore, much of the

effort has been to support network-centric applications, such as multimedia, rather than the more general applications that we seek to support here.

8. 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.

Our SLA-based model, with hierarchies of intermediaries, emphasizes multi-phase negotiation across the policy domain boundaries that structure the Grid. By identifying three important types of SLA needed for multi-phase negotiation, we show how generic brokering patterns can be deployed and extended with details of new resource types. The use of generic negotiation patterns and the allowance for resource virtualization—by which intermediaries translate from one set of negotiable terms to another—allows for evolution in a resource management architecture. Evolution is an important step toward realizing a permanent global Grid, in which new resource capabilities and application modes must be incorporated into a running distributed system.

Acknowledgments

We are grateful to many colleagues for discussions on the topics discussed here, in particular Larry Flon, Jeff Frey, Steve Graham, Bill Johnston, Miron Livny, Jeff Nick, Alain Roy and Volker Sander. 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

[Ber99] F. Berman. High performance schedulers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, chapter 12, pages 279–309. Morgan Kaufmann, 1999.

[BL98] S. Berson and R. Lindell. An architecture for advance reservations in the Internet. Technical report, Information Sciences Institute, University of Southern California, 1998. Available from `www.isi.edu/~berson/advance.ps`.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 functional specification. Technical Report RFC 2205, Internet Engineering Task Force (IETF), 1997.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C, 2001. Available from `http://www.w3.org/-TR/-wsdl/`.

[CFK+98a] Prashant Chandra, Allan Fisher, Corey Kosak, T. S. Eugene Ng, Peter Steenkiste, Eduardo Takahashi, and Hui Zhang. Darwin: Resource management for value-added customizable network service. In *Proceedings of the Sixth IEEE International Conference on Network Protocols (ICNP'98)*, 1998.

[CFK+98b] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Fourth International JSSPP Workshop; LNCS #1459)*, pages 62–82. Springer-Verlag, 1998.

[CFK99] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational Grids. In *Proceedings of the Eighth IEEE*

International Symposium on High Performance Distributed Computing (HPDC-8), August 1999.

[CFK+02] K. Czajkowski, I. 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. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (Proceedings of the Eighth International JSSPP Workshop; LNCS #2537)*, pages 153–183. Springer-Verlag, 2002.

[DKPS97] M. Degermark, T. Kohler, S. Pink, and O. Schelen. Advance reservations for predictive service in the Internet. *Multimedia Systems*, 5(3):177–186, 1997.

[FFR+02] I. Foster, M. Fidler, A. Roy, V. Sander, and L. Winkler. End-to-end quality of service for high-end applications. *Computer Communications, Special Issue on Network Support for Grid Computing*, 2002.

[FGV97] D. Ferrari, A. Gupta, and G. Ventre. Distributed advance reservation of real-time connections. *Multimedia Systems*, 5(3), 1997.

[FK99] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffmann, 1999.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. Also available from `http://www.globus.org/research/papers/anatomy.pdf`.

[FRS00] I. Foster, A. Roy, and V. Sander. A quality of service architecture that combines resource reservation and application adaptation. In *Proceedings of the International Workshop on Quality of Service*, 2000.

[GS99] Roch Guérin and Henning Schulzrinne. Network quality of service. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 479–503. Morgan Kaufmann, 1999.

[HvBD98] A. Hafid, G. von Bochmann, and R. Dssouli. A quality of service negotiation approach with future reservations (NAFUR): a detailed study. *Computer Networks and ISDN Systems*, 30(8):777–794, 1998.

[LMN94] C. Lai, G. Medvinsky, and B. C. Neuman. Endorsements, licensing, and insurance for distributed system services. In *Proceedings of the Second ACM Conference on Computer and Communication Security*, 1994.

[MBHJ98] D. Marinescu, L. Boloni, R. Hao, and K. Jun. An alternative model for scheduling on a computational Grid. In *Proceedings of the Thirteenth International Symposium on Computer and Information Sciences (ISCIS'98)*, pages 473–480, 1998.

[MIS96] A. Mehra, A. Indiresan, and K. Shin. Structuring communication software for quality-of-service guarantees. In *Proceedings of Seventeenth Real-Time Systems Symposium*, December 1996.

[NCN98] K. Nahrstedt, H. Chu, and S. Narayan. QoS-aware resource management for distributed multimedia applications. *Journal on High-Speed Networking, Special Issue on Multimedia Networking*, 8(3-4):227–255, December 1998.

[NS96] K. Nahrstedt and J. M. Smith. Design, implementation and experiences of the OMEGA end-point architecture. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Distributed Multimedia Systems and Technology*, 14(7):1263–1279, September 1996.

[PWF+02] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Proceedings of the IEEE Third International Workshop on Policies for Distributed Systems and Networks*, 2002.

[SP98] O. Schelen and S. Pink. Resource sharing in advance reservation agents. *Special issue on Multimedia Networking*, 7(3-4), 1998. Also available from `http://www.cdt.luth.se/~olov/publications/JHSN-98.pdf`.

[TCF+03] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructure (OGSI) 1.0 draft. Technical report, Global Grid Forum (GGF), March 2003. Available from `http://www.gridforum.org/ogsi-wg/`.

[WG98] D. Wischik and A. Greenberg. Admission control for booking ahead shared resources. In *Proceedings of IEEE INFOCOM'98*, 1998.

[WS97] L. C. Wolf and R. Steinmetz. Concepts for reservation in advance. *Kluwer Journal on Multimedia Tools and Applications*, 4(3), May 1997.