A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation

Ian Foster
Mathematics and Computer Science Division, Argonne National Laboratory and
Department of Computer Science, University of Chicago
Carl Kesselman
Information Sciences Institute, University of Southern California
Craig Lee
The Aerospace Corporation
Bob Lindell
Information Sciences Institute, University of Southern California
Klara Nahrstedt
Department of Computer Science, University of Illinois, Urbana-Champaign
Alain Roy
Department of Computer Science, University of Chicago

Abstract

The realization of end-to-end quality of service (QoS) guarantees in emerging network-based applications requires mechanisms that support first *dynamic discovery* and then *advance or immediate reservation* of resources that will often be *heterogeneous in type and implementation* and *independently controlled and administered*. We propose the Globus Architecture for Reservation and Allocation (GARA) to address these four issues. GARA treats both reservations and computational elements such as processes, network flows, and memory blocks as first class entities, allowing them to be created, monitored, and managed independently and uniformly. It simplifies management of heterogeneous resource types by defining uniform mechanisms for computers, networks, disk, memory, and other resources. Layering on these standard mechanisms, GARA enables the construction of application-level co-reservation and co-allocation libraries that applications can use to dynamically assemble collections of resources, guided by both application QoS requirements and the local administration policy of individual resources. We describe a prototype GARA implementation that supports three different resource types—parallel computers, individual CPUs under control of the Dynamic Soft Real-Time scheduler, and Integrated Services networks—and provide performance results that quantify the costs of our techniques.

1 Introduction

Correct execution of emerging performance-oriented network-based applications [10] often requires an end-to-end provision of high quality of service (QoS). This end-to-end QoS can be achieved and guaranteed through proper configuration, reservation and allocation of corresponding resources. For example, interactive data analysis may require simultaneous access to a storage system holding a copy of the data, a supercomputer for analysis, network elements for data transfer, and a display device for interaction, with each resource providing a specified QoS.

Such applications require *discovery* and *selection* mechanisms for selecting from among alternative candidate resources according to QoS criteria such as reliability, availability, cost, and performance. They also require resource *allocation* mechanisms for mediating among competing requests and for preventing oversubscription. Note that the networking research literature frequently uses the terminology *immediate reservation* rather than allocation (e.g., the ReSerVation Protocol: RSVP [3]). For uniformity and brevity, we will use the term allocation to apply to all resources of interest: computers, networks, disk, and memory.

Applications may also require *advance reservation* mechanisms that provide an increased expectation that resources can be allocated when demanded, much as an airline or concert ticket provides an increased expectation of obtaining a seat. In the absence of a reservation system, we encounter either increased costs due to excess overprovisioning (as in the Public Switched Telephone Network) or degraded service for critical traffic (as in today's Internet). In some systems, such as supercomputers, overprovisioning may not be an option.

The implementation and application of these mechanisms in practical settings is made difficult by four factors: (1) deployed systems (e.g., [3, 21, 5]) lack support for advance reservations; (2) applications of interest require the discovery, reservation, and allocation of not just a single resource but potentially complex collections; (3) resources can be of widely varying types: computers, networks, disk, memory, etc.; and (4) resources are commonly located in different administrative domains and subject to different control policies and mechanisms.

The *Globus Architecture for Reservation and Allocation* (GARA) described in this paper addresses these four issues. GARA builds on techniques and concepts developed in the Globus toolkit [9] for the provision of computational QoS via the co-allocation of computers [5], generalizing and extending them to support end-to-end discovery, reservation, allocation, and management of heterogeneous ensembles of computers, networks, storage systems, and other resources under independent local control.

In particular, GARA: (1) treats both advance reservations and computational elements—such as processes, network flows, and memory blocks—as first-class entities that can be created, monitored, and managed independently; (2) supports in a uniform fashion different resource types (e.g., networks, CPUs, memory, disk) and low-level mechanisms (e.g., in the case of networks, RSVP signaling, differentiated services bandwidth brokers, and ATM virtual circuits); and (3) defines a layered architecture that allows strategies for the discovery, reservation, allocation, and management of resource collections to be encapsulated in co-reservation and co-allocation agents. Agent code can be application-specific or generic, can be linked with an application or instantiated in independent "brokers," and can incorporate centralized or distributed implementations. We believe that this flexibility is essential for advanced applications in which domain-specific knowledge is required to achieve good end-to-end QoS. For example, the above data analysis application may achieve desired QoS by passing a description of its requirements to standard co-reservation and co-allocation agents, but can then respond with domain-specific guidance if an agent signals difficulties in resource reservation or allocation.

In addition to describing GARA concepts and architecture, we also report on a prototype implementation that supports co-reservation and co-allocation of parallel computers, individual CPUs under the control of the Dynamic Soft Real-time (DSRT) scheduler [18], and networks with RSVP signaling. We present experimental results that quantify the cost of local reservation and object creation operations.

2 Related Work

The general problem of resource management in networks and wide area computing systems is receiving increased attention (for reviews, see e.g., [1, 11]). However, there has been little work on the specific problems addressed in this paper, namely advance reservation and co-allocation of heterogeneous collections of resources for end-to-end QoS. Here we review briefly some relevant work; space constraints prevent a complete survey.

Proposals for advance reservations in the Internet typically implement advance reservation capabilities via cooperating sets of servers that coordinate advance reservations along an end-to-end path [20, 8, 7, 13, 2]. Techniques have been proposed for representing advance reservations, for balancing immediate and advance reservations [8], for advance reservation of predictive flows [7], and for handling multicast [2]. However, this work has not addressed problems that arise when an application requires co-allocation of multiple resources of different types.

The Globus resource management architecture [9, 5] supports the co-allocation of heterogeneous compute resources to support end-to-end computational QoS. The architecture includes an information service, used to locate resources meeting certain criteria, such as architecture, installed software, availability, and network connectivity; local resource managers, which encapsulate the local policies and mechanisms used to initiate, monitor, and control computation on particular resources; and a "signaling protocol" used to communicate allocation requests to resource managers. Both generic and application-specific co-allocation strategies can be encapsulated in reusable libraries.

The Darwin project at CMU is building a system with many similarities to the Globus architecture [4]. A resource broker called Xena implements co-allocation strategies and a signaling protocol called Beagle is used to communicate allocation requests to local resource managers that may provide access to network, storage, and compute elements. The concept of hierarchical scheduling is introduced to allow controlled sharing of network resources managed by different providers: individual providers can specify sharing policies and the Hierarchical Fair Share Curve scheduler is used to determine an efficient schedule that meets all constraints. Like Globus, Darwin does not support advance reservations.

Also relevant to the co-allocation problem is multimedia system research concerned with identifying the appropriate mix of resources required to provide desired end-to-end QoS. Multimedia applications have motivated the development of techniques for allocating both memory and CPU for channel handlers [16] and of CPU, bandwidth, and other resources for video streams [19, 18]. However, these techniques are specific to particular mixes of resources and do not extend easily to other resource types.

3 GARA Architecture Design

The GARA design was strongly influenced by experiences with the resource management architecture [5] developed for Globus, a toolkit for the development of advanced network applications [9]. In the following, we provide a brief overview of this architecture, discuss its limitations, and then describe how GARA addresses and overcomes these limitations.

3.1 The Globus Resource Management Architecture

As discussed above, the Globus resource management architecture as described in [5] addresses the relatively narrow QoS problem of providing dedicated access to collections of computers in heterogeneous distributed systems. The architecture has been deployed on a testbed that spans hundreds of computers at dozens of sites in eight countries [9] and has proven effective in numerous large application experiments. As illustrated in Figure 1, the architecture consists of three main components: an *information service*, local *resource managers*, and various types of *co-allocation agents*, which implement strategies used to discover and allocate the resources required to meet application QoS requirements.

The information service uses Lightweight Directory Access Protocol (LDAP) concepts to define a hierarchical name space, uniform representation, and standard access methods for resources. Information service entries characterize resources in terms of their type, architecture, structure (e.g., network connectivity) and current state (e.g., current load, operating system version, installed software, availability). Automated discovery and publication mechanisms ensure that information service contents are kept up to date. Query mechanisms allow applications or agents acting on their behalf to locate resources with desired characteristics.

An application that wishes to create a computation passes a description of that computation to a co-allocation agent. This agent uses some combination of information service queries, general heuristics, and application-specific knowledge to map application QoS requirements into resource requirements, to discover resources with those requirements, and to allocate those resources. This agent also typically incorporates co-allocation strategies to provide robust startup across multiple resources in the presence of failure. For example, the Globus toolkit's Dynamically Updateable Resource Online Co-allocator (DUROC) uses upcalls to the application to signal failure of individual allocation events [6].

An agent allocates an individual resource by directing an allocation request to a local resource manager, or Globus Resource Allocation Manager (GRAM). A GRAM takes a request as input, authenticates the request using the Globus Security Infrastructure, and, if the request was successful, interfaces to local schedulers to allocate that resource and create a "job," returning a portable "job handle" as output. The requesting process can use the job handle to monitor and control the state of a computation and can request upcalls to signal such events as a job entering the run state or terminating. GRAMs have been developed for a variety of CPU schedulers [5].

This architecture addresses two of the four concerns identified in the introduction. It supports the management of collections of resources via the use of co-allocation agents and the co-allocation strategies that these agents can encapsulate. In addition, the existence of a standardized source of information about managed resources and a consistent GRAM interface for allocating and controlling resources enables applications and co-allocation agents to deal with the site-specific variations that are inevitable across collections of independently administered resources.

The two issues that this architecture does not address are advance reservations and (apart from some preliminary investigations of network scheduling [17]) heterogeneous resource types. The absence of advance reservations means that we cannot ensure that a resource can provide a requested QoS when required, which drastically restricts our ability to perform co-allocation, as specialized resources such as supercomputers and high-bandwidth virtual channels are typically in high demand. The lack of support for network, disk, and other resource types makes it impossible to provide end-to-end QoS guarantees when (as is normally the case) an application involves more than just computation.

3.2 The Globus Architecture for Reservation and Allocation

GARA extends the Globus resource management architecture in two major ways: it introduces the generic *resource object*, which encompasses network flows, memory blocks, disk blocks, and other entities as well as processes; and introduces the *reservation* as a first class entity in the resource management architecture. Various other architectural changes follow from these new concepts.

We discuss resource objects first. In GARA, we reformulate computation-specific allocation functions in terms of general resource *objects*, hence allowing different application components to be manipulated in common ways. A generic "Create Object" operation is used to create a process, flow, disk object, memory object, etc., according to the supplied arguments. Each "Create Object" call returns an object handle that can subsequently be used to monitor and control the object (e.g., to delete it); a system or application process can also request upcalls on specific events, such as reservation applied to object, object termination, or QoS contract violations. Upcalls allow the construction of adaptive systems.

We next consider reservations. GARA splits the task of creating a resource object into two phases: reservation and allocation. In the reservation phase, a reservation is created, which provides some confidence that a subsequent allocation request will succeed; however, no object is created at this time. Instead, a reservation handle is returned, that can be used to monitor and control the status of the reservation and that can also be passed to a subsequent "Create Object" call in order to associate that object with the reservation.

The introduction of a distinct reservation phase has two important ramifications. First, by splitting reservation from allocation, it enables us to perform advance reservation of resources, which can be critical to application success if a required resource is in high demand. Second, if reservation is cheaper than allocation (as is often the case for large parallel computers, for example), we can implement lighter-weight resource reservation strategies than if objects must

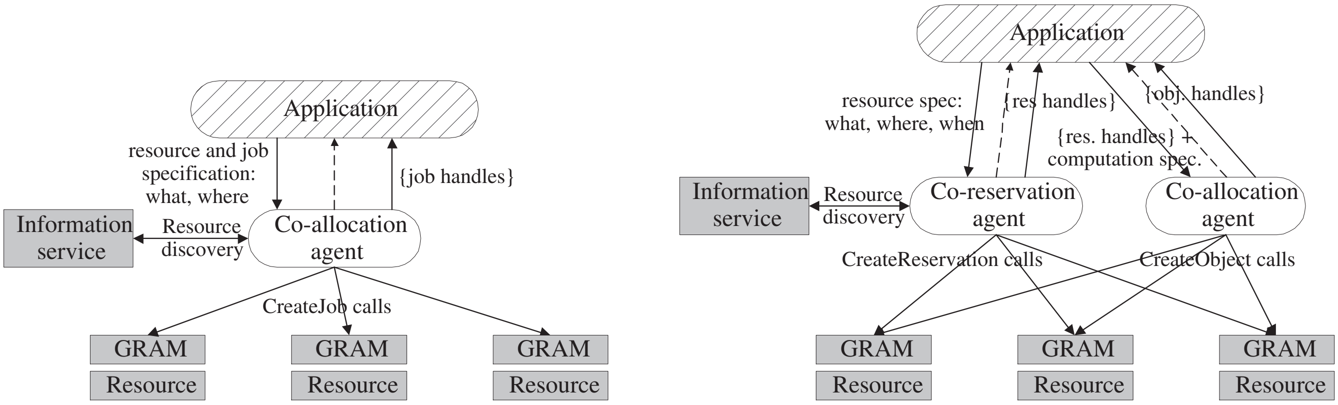

Figure 1: The Globus and GARA resource management architectures, on the left and right, respectively. Notice the distributed information service and local resource managers (GRAMs), and the co-reservation and co-allocation agents that applications use to create distributed computations. The dashed line represents upcalls, which may be used to invoke application-specific routines, for example on failure.

be created in order to guarantee access to a resource.

A reservation is created by a generic "Create Reservation" operation, which interacts with local resource management elements to ensure that the requested quantity and quality of the resource will be available at the requested start time and will remain available for the desired duration. If the resource cannot make this assurance, the "Create Reservation" operation fails. All "Create Object" operations require a reservation in order to proceed. This reservation is normally created via a preceding "Create Reservation" call, but for some resources a default "best effort" reservation can be specified. Note that the GARA concept of reservation encompasses both *immediate* reservations, which are assumed to be followed immediately by an allocation, and *advance* reservations, which are created in the present to reserve resources for use in the future.

Reservation and object creation operations are implemented by a renamed GRAM: the Globus Reservation and Allocation Manager. As illustrated in Figure 1, GARA introduces a new entity called a *co-reservation agent.* A co-reservation agent, like our earlier co-allocation agent, is responsible for discovering a collection of resources that can satisfy application end-to-end QoS requirements (a *resource set*); however, rather than allocating those resources, it simply reserves them. Hence, a call to a co-reservation agent specifies QoS requirements and returns a set of reservation handles that can then be passed to an co-allocation agent. In GARA, the co-allocation agent remains but now has the simpler task of allocating a resource set, given a reservation handle generated by a co-reservation agent. (In practice it may need to either incorporate some aspects of co-reservation agent functionality, or interact with an external co-reservation agent, in order to recover from allocation failures that may occur.)

In summary, GARA supports advance reservation directly. The introduction of generalized resource objects along with the standardized interface provided by GRAM addresses issues of heterogeneity in the resource set. Co-reservation and co-allocation agents layered on top of GRAM and the standardized information services enable the dynamic construction of collections of independently administered resources that satisfy application QoS requirements.

4 Co-Reservation/Allocation Agents

Co-reservation (and, to a lesser extent, co-allocation) agents play a critical role in GARA. They provide a bridge between the application and the available resources, constructing sets of resources that both match application QoS requirements and conform to the local practices and policies of resource providers.

Because GARA does not constrain agent design other than to define the GRAM and information service functions used to construct implementations, a wide range of agent architectures are possible. An agent can take the form of a library, linked with an application, that makes reservation decisions on behalf of that single application [1]. Alternatively, an agent may be a global system-oriented "broker" that provides reservation services to numerous users and applications. Functionality may be centralized or may be distributed over multiple agent instances or throughout a hierarchy of different agents. (The latter organization is useful, for example, if certain subsets of required resources are under the control of "local" reservation systems, such as bandwidth brokers.) Finally, an agent can act autonomously, responding to a reservation request with either success or failure, or may proceed interactively, allowing a user or application to guide the construction of a resource set.

To clarify the role of co-reservation agents, we return to the data analysis example of the introduction. Let us assume that while visualization must occur at a specific location (the user's computer), we can choose from among several cached replicas of the data and from among several alternative analysis supercomputers (Figure 2). Reservation and object creation operations on these data stores, supercomputers, and network elements can be achieved by calls to appropriate GRAMs, although each system may of course implement these functions with quite different mechanisms.

We imagine a co-reservation agent that takes as input a specification of the dataset that is to be analyzed and an indication of desired QoS, expressed in terms of how precise the result should be, how soon results are required, and how much the user is prepared to pay. The agent uses the information service to locate cached data replicas—if the data is not cached, the agent may return control to the user to migrate data from a tape archive—and then, having determined relevant data properties (e.g., its size and location), invokes a secondary agent to determine computational and network requirement for analysis and data transfer.

The agent must now discover computational and bandwidth resources that can collectively provide desired end-to-end QoS. Applications developed in the context of the current Globus system achieve this goal—for co-allocation—by using exhaustive search [5]. In an advance reservation environment, we can consider a range of future times, and so the number of candidate resources can be larger; hence, efficient search heuristics will typically be required. For example, we can consider each potential data cache in turn, consulting for each the information service to locate a supercomputer that can deliver the required computational power. At this point the agent may need to consider issues such as acceptable use and security policies, perhaps because data is proprietary. Then, the agent attempts to reserve both supercomputer nodes and network bandwidth between the supercomputer and the visualization engine. If both reservations succeed, the agent can proceed to discover and reserve a network link between the supercomputer and the data cache.

This example illustrates an important aspect of end-to-end reservation, namely the importance that application-level criteria (e.g., end-to-end security requirements) can have for resource selection. The example also illustrates other issues that must be considered when developing agent strategies. For example, search procedures such as that just outlined can produce several alternative resource sets. If it does, we can choose between alternatives based on selection criteria such as first found and "best" found. If no suitable resources exist, the agent may either fail or attempt to renegotiate with the user, who might for example decide to proceed with fewer analysis nodes than was originally desired, or with best effort rather than reserved bandwidth.

In this example, reservation failure is handled by backtracking: we proceed to try alternative resources until either the request is successful or fails. In other situations, we may prefer to wait until all required resources are available. In this case, we need to be concerned about the possibility of deadlock, as other agents may attempt to acquire some of the same resources simultaneously. Deadlock can be avoided by using variants of well-known deadlock prevention and avoidance schemes [14, 12], such as enforcing orderings on how resources are acquired (e.g., processors sorted by IP address, followed by disks, followed by networks, etc) or timeout mechanisms.

The search and deadlock strategies just described are necessary because in this application there are dependencies among required resources. In other situations, simpler techniques can be used. For example, consider an application that simply requires N computational resources with a specified minimum network connectivity. If some number $M < N$ of an attempted N reservations succeed, then we can either return to the information service to locate additional candidates or generate a callback to the application to determine whether it is possible to proceed with just M resources, a strategy used by the DUROC co-allocator when allocations fail [6].

5 GARA Application Programming Interface

The GARA client-side application programming interface (API) includes calls to create and cancel reservations and objects, and to query and to request notification of changes in the status of reservations and objects. As outlined in Figure 3, arguments include resource manager contact addresses; portable reservation and object handles; and specifications of required resources and object characteristics, expressed in a declarative resource specification language (RSL) [5].

Briefly, a `CreateReservation` call takes as input a representation of the required resources and returns a reservation handle, while a `CreateObject` call takes as input a specification of the required object and a reservation handle representing the resources with which the object is to be associated, and returns a portable object handle. `CancelReservation` and `CancelObject` allow cancellation of previously created reservations or objects, using the appropriate handles. `RegisterCallback` allow an application to request upcalls on selected events, such as the failure of a reservation. `ModifyReservation` allows an application to adapt if existing reservations can't be honored or an application's requirements change. Besides these API function calls, a user can use reservation and object specifications to control various aspects of object behavior, for example what should happen to an object when its reservation expires.

The use of these calls is illustrated in Figure 4, which shows in pseudo-code logic that can be used to discover a computer ("`b`") that can be linked via a network ("`net`") to an originating computer ("`a`") to provide desired QoS.

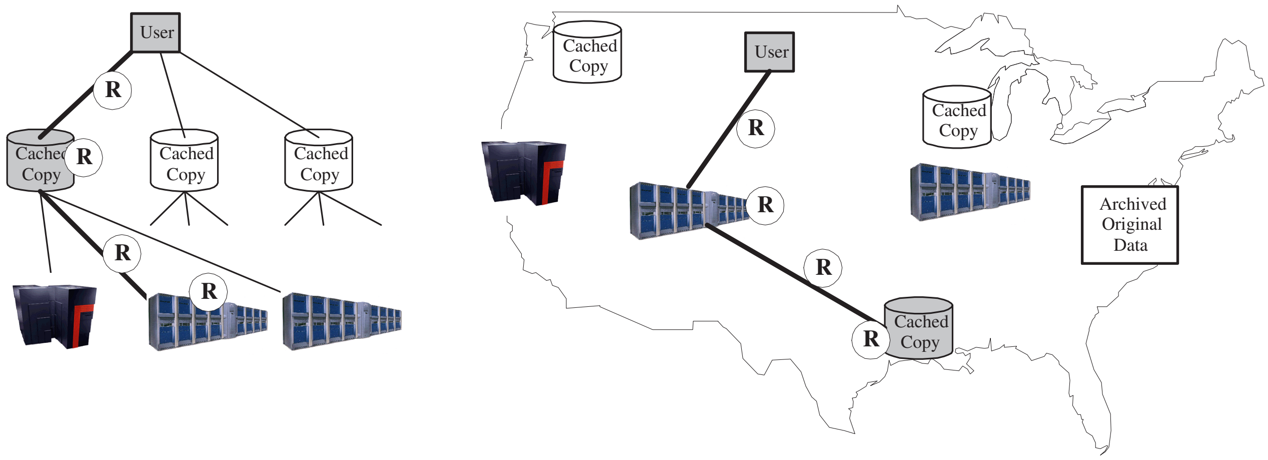

Figure 2: Candidate resources for a data analysis application include multiple cached copies of the Terabyte dataset; multiple supercomputers used for analysis; and network links (not shown) connecting these components. On the right, we show the physical location of the resources and on the left the search tree constructed by the co-reservation agent; the "R"s represent reservations. This configuration is typical of problems encountered in high-performance data-intensive computing.

reservation-handle	=	`CreateReservation`(*resource-manager-contact*, *reservation-specifation*)
object-handle	=	`CreateObject`(*resource-manager-contact*, *reservation-handle*, *object-specification*)
result	=	`CancelReservation`(*reservation-handle*)
result	=	`CancelObject`(*object-handle*)
new-resv-handle	=	`ModifyReservation`(*reservation-handle*, *new-reservation-specification*)
result	=	`RegisterCallback`(*handle*, *callback-function*, *callback-argument*)

Figure 3: Pseudo-code prototypes for selected GARA client-side API functions

(This problem is a simplified version of the data analysis system described in Section 5.) The co-reservation agent `ReserveResources` uses an exhaustive search procedure to locate and reserve resources for the required time, constructing a set of reservation handles (`rh-a`, `rh-b`, and `rh-net`) representing these reservations. The routine `FindNextCandidate` (not shown) queries the information service to locate a candidate compute and network resources, returning GRAM contacts and a host name for these resources (`contact-b`, `id-b`, `contact-net`). While this routine embeds specific resource requirements (the italicized RSL arguments to the `CreateReservation` calls), it can easily be parameterized to provide a completely generic procedure.

6 GARA Implementation

We limit discussion of GARA implementation to the local resource manager component as this is where issues of heterogeneity are addressed. We explain the basic structure of our implementation and describe the three resource managers that we have constructed to date. The discussion illustrates how advance reservations can be implemented via a *slot manager* and how heterogeneous resources can be encapsulated under common interfaces.

GARA has a layered structure: a GARA External Interface (GEI) component addresses issues such as authentication and dispatch of incoming requests, the registration and propagation of upcalls to remote processes, and publication of resource information; a lower-level Local Resource Allocation Manager (LRAM) provides basic object and reservation services, interacting only with system-specific resource management components and services. We focus here on the LRAM because of its dependence on the underlying resource.

The structure of a particular LRAM implementation depends upon the nature of the local resource management services (e.g., the "scheduler") associated with the resource in question. Three major cases can be distinguished:

1. If the scheduler provides appropriate advance reservation support, then an LRAM for that resource can pass advance reservation requests directly to the scheduler.

2. Otherwise, we have to deal with two cases:

    (a) If an LRAM associated with a resource has total control over the resource (i.e., all object creation calls must pass via the LRAM), then the LRAM

```
subroutine ReserveResources
  rh-a = CreateReservation(contact-a,
    "&(reservation_type=compute)
      (start_time="10:30 pm")
      (duration="1 hour") (nodes=32)")
  if rh-a is null then exit
  repeat until rh-b and rh-net defined:
    (contact-b, id-b, contact-net) =
      FindNextCandidate()
    rh-b = CreateReservation(contact-b,
      "&(reservation_type=compute)
        (start_time="10:30 pm")
        (duration="1 hour")
        (percent_cpu=75)")
    if rh-b is null then continue
    rh-net = CreateReservation(contact-net,
      "&(reservation_type=network)
        (start_time="10:30 pm")
        (duration="1 hour") (bandwidth=200)
        (endpoint-a=id-a) (endpoint-b=id-b)")
    if rh-net is null then
      CancelReservation(rh-b)
  end repeat loop
  if rh-b is null then
    signal that search failed
end subroutine
```

Figure 4: Pseudo-code for a GARA-based co-reservation agent `ReserveResources` that reserves a set of three resources. The variables with "`rh`" prefixes are reservation handles and the italicized text is RSL specifications.

can use a slot manager (see below) to implement advance reservations for that resource.

(b) Otherwise, only probabilistic advance reservations can be supported, for example by using a slot manager and defining "available" resources at any one time as the amount of resources that we predict to be available at that time.

We introduce briefly the concepts of a *timeslot table* (slot table for short) and a *timeslot manager* [8, 15]. If the available capacity of a resource is a known, enumerable quantity—e.g., peak network bandwidth or number of processors—then a slot table data structure can be used to keep track of current allocations and future reservations (Figure 5). We have constructed a resource-neutral slot manager library that uses a slot table to ensure that committed resources never exceed a specified limit (e.g., 70% of peak network bandwidth). This library provides functions for slot table creation, for requesting and canceling a reservation, for querying the state of the current reservation table, for examining properties of a specific reservation, and for requesting upcalls at the time of reservation activation or termination. Note that our slot manager represents just one, relatively simple implementation of advance reservation functionality; more sophisticated slot managers might for example support pre-emption, or invoke a policy agent [15] to determine whether requests should be allowed.

To our knowledge, no widely deployed resource management system for networks, computers, or other resources supports advance reservations. Hence, the three LRAMs developed to date are all of type 2(a) or 2(b). As illustrated in Figure 6, each uses our slot manager to manage allocations and reservations and differs from the others only in the nature of the resource being managed and the low-level mechanisms used to create and destroy objects.

The *SMP* LRAM supports parallel execution on a shared memory multiprocessor for which the number of processes created must not exceed a specified threshold N. We use the slot manager to manage resource reservation and creation. For example, a `CreateReservation` call creates a slot manager entry, a `CreateObject` call registers a "create process" (Unix fork) function to be called when a reservation becomes active, and reservation termination is handled by registering a "destroy process" (Unix kill) function.

If all process creation requests on an SMP must occur via our LRAM then this LRAM is of type 2(a) and reservations can be guaranteed, modulo system failures. If users can also create processes by other means, then we have an LRAM of type 2(b) and reservations can only be probabilistic.

The *DSRT* LRAM manages fractions of a single CPU to provide "soft" realtime response, with DSRT [18] used to manage task scheduling. Again, the slot manager is used to manage the allocation of resources, with in this case the quantity available for allocation being a CPU fraction (typically 70%) rather than a number of CPUs. Object creation involves the use of normal Unix system functions to create a process; reservation activation involves a call to a DSRT functions to adjust process scheduling. The *DSRT* LRAM, like *SMP*, may be of type 2(a) or 2(b), depending on whether or not all DSRT operations must occur via LRAM.

Finally, the *IntServ* LRAM is concerned with the management of network flows and with the assignment of bandwidth reservations to those flows. In this case the quantity managed by the slot manager is bandwidth, object creation occurs via standard Unix socket calls, and reservations are associated with flows via calls to an RSVP signaling API (SCRAPI). In our current prototype, RSVP calls are serviced on a first-come first-served basis and so our LRAM is of type 2(b). A type 2(a) LRAM can be obtained if the RSVP implementation is constrained so that allocations proceed only if approved by our slot manager. One approach to implementing this approval process would be to use a COPS policy agent to control allocations, with our slot manager being charged with updating policies.

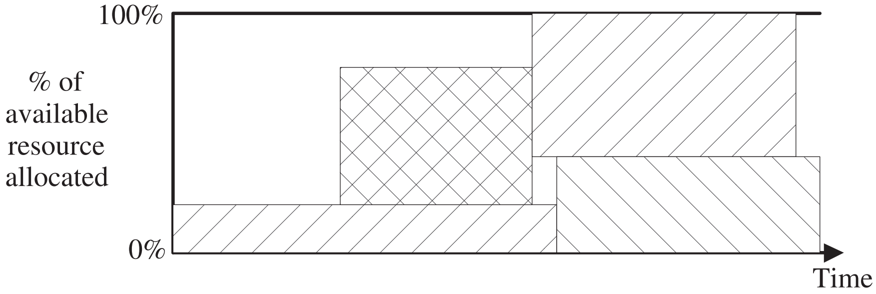

Figure 5: A slot table keeps track of current allocations and future reservations for a resource

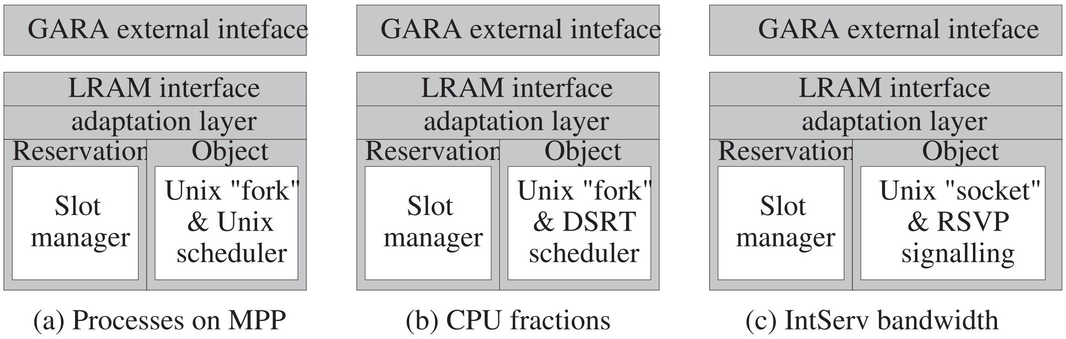

Figure 6: The three GARA resource managers constructed to date, showing the resource-independent components (shaded) and resource-specific components (unshaded)

7 Results

We have constructed implementations of the SMP, DSRT, and RSVP resource managers described above, and demonstrated our ability to perform co-reservations and co-allocations across multiple resources of different types. Here, we present experimental results that provide insights into the performance of our techniques and implementations.

We are interested in answering three questions: (1) What are the costs of our reservation and object creation mechanisms? (2) How do these costs compare with the "native" costs that would be incurred if resources were allocated without GARA mechanisms? (3) What does a comparison of reservation and creation costs tell us about the practical utility of search-based reservation strategies?

We performed experiments on a pair of Sun Ultra-2 workstations running Solaris 2.6 and on the same 100 Mb/s Ethernet segment. We timed certain major GARA and LRAM operations, measuring however only the cost of individual operations, not the time required to make end-to-end reservations. (The latter experiments are clearly important, but require that GARA resource managers be deployed on a larger number of systems.) For each of our three LRAM implementations, we measured "native object creation" costs (fork for SMP, fork plus set priority for DSRT, create flow plus bind to reservation for RSVP) and then the costs of `CreateReservation`, `CreateObject`, and `CancelObject` calls when performed via a local LRAM call and via a GARA call from a remote workstation, with and without authentication. LRAM times, presented in Table 1, show that reservation is much cheaper than object creation within a single computer. (The DSRT object creation and cancelation times are higher than those for SMP because in the DSRT case we must communicate with a custom CPU scheduler; RSVP object creation costs are higher still because of the need to communicate with an RSVP agent to instantiate the reservation.)

Table 1: The cost of selected native object creation and LRAM operations. All times are the average of 10 runs and are in milliseconds.

	DSRT	SMP	RSVP
Native object creation	6.80	6.30	25.20
LRAM reservation	0.81	0.82	0.82
LRAM object creation	17.63	11.87	31.20
LRAM object cancellation	7.29	0.45	28.30

For operations performed remotely using the GARA interface, we incur additional costs of around 11 msec for network communication and around 100 msec for authentication. The communication cost includes the time to open a socket, which could be avoided on subsequent requests when multiple requests are made to the same LRAM computer from a single host. The authentication cost includes three round trip communications performed by the SSL handshake and 1024-bit public key operations for credential verification. Clearly, remote access time is dominated by authentication rather than communication. These results suggest that the GRAM constructed to date can sustain around 8-10 reservation operations per second; a single user process can sustain significantly higher reservation rates, if multiple calls are issued simultaneously.

We find these results encouraging in terms of what they reveal of GARA performance. In a wide area environment, in which the heavyweight authentication used here is necessary, the cost of creating and then cancelling an object is slightly higher than that of creating and then cancelling a reservation: for example, 283 vs. 225 msec in the case of RSVP. (In other systems, such as large parallel computers, object creation costs can be significantly higher.) Hence, search schemes that create and then cancel reservations will be at least as efficient as schemes that create and destroy objects, and in many cases will be significantly more efficient. (Soft state pre-reservations that time out can be used to avoid the need to cancel unwanted reservations [15], but will make successful reservations more expensive.) "Immediate" object creation requires a reservation call followed by an object creation call; however, the additional cost inherent in the two calls can be avoided by defining a "create reservation and object" operation. Finally, we see that if a computation requires multiple resources within the same administrative domain, it can be desirable to avoid repeated remote operations so as to avoid multiple authentication operations, for example by downloading an agent able to negotiate on a user's behalf.

8 Conclusions

A significant impediment to the production use of advanced networked applications has been the difficulty of achieving end-to-end QoS guarantees across heterogeneous collections of shared resources. We have presented a resource management architecture, GARA, that addresses this problem. GARA exposes both reservations and objects as first-class, abstract objects; defines uniform representations and operations for diverse resource types; and uses an information service to reveal site-specific policies. These constructs enable the construction of reusable co-reservation and co-allocation agents that can combine domain- and resource-specific knowledge to discover, reserve, and allocate resources that meet application QoS requirements.

We have constructed a prototype GARA implementation and conducted initial performance experiments for simple end-to-end scenarios. Results demonstrate that the cost of GARA mechanisms are not large when compared to underlying resource management operations. However, while previous experience with the Globus resource management architecture has shown the utility of the basic approach, large-scale experimentation, particularly with advance reservations, remains for the future. One area of uncertainty relates to whether it is really feasible or useful to treat all computational objects in a uniform fashion, as proposed here. While reservations can clearly be represented and manipulated in a uniform fashion, it may well be that processes and flows are sufficiently different in their characteristics that distinct representations and operations are warranted.

We plan future work in four major areas. First, we will implement additional resource managers: for example, to mediate access to parallel computer schedulers providing advance reservations; to premium traffic on ingress and egress routers in Differentiated Services networks [15]; and to other resource types, such as disk I/O, disk blocks, and memory blocks. We are also interested in understanding how our techniques can be applied to multicast. Second, we will deploy GARA functionality more widely in our testbed environment and develop and evaluate high-performance distributed applications that exploit GARA capabilities. Third, we will investigate techniques for co-reservation and co-allocation. Finally, we plan GARA extensions designed to support notification of QoS violations and application-level adaptation.

Acknowledgments

We gratefully acknowledge helpful discussions with Fred Baker, Gary Hoo, Bill Johnston, Brian Toonen, and Steven Tuecke, and the assistance of Soonwook Hwang with the implementation of the RSVP LRAM. This work was supported in part by the Defense Advanced Research Projects Agency under contract N66001-96-C-8523, by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by the National Science Foundation.

References

[1] F. Berman. High-performance schedulers. In *[10]*, pages 279–309.

[2] S. Berson and R. Lindell. An architecture for advance reservations in the internet. Technical report. Work in Progress.

[3] R. Braden, L. Zhang, D. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 functional specification. Internet Draft, Internet Engineering Task Force, 1996.

[4] 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 *Sixth IEEE International Conference on Network Protocols (ICNP'98)*, 1998.

[5] 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. Springer-Verlag LNCS 1459, 1998.

[6] Karl Czajkowski, Ian Foster, and Carl Kesselman. Co-allocation services for computational grids. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1999.

[7] 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.

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

[9] I. Foster and C. Kesselman. Globus: A toolkit-based grid architecture. In *[10]*, pages 259–278.

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

[11] Roch Guérin and Henning Schulzrinne. Network quality of service. In *[10]*, pages 479–503.

[12] A.N. Habermann. Prevention of system deadlocks. *Communications of the ACM*, 12(7):373–377, July 1969.

[13] 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.

[14] J.W. Havender. Avoiding deadlock in multitasking systems. *IBM Systems Journal*, 7(2):74–84, 1968.

[15] G. Hoo, W. Johnston, I. Foster, and A. Roy. QoS as middleware: Bandwidth broker system design. Technical report, 1998.

[16] 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.

[17] K. Nahrstedt. Globus and quality of service. In *Proc. 1998 Globus Retreat*. Argonne National Laboratory, 1998.

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

[19] 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.

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

[21] L. Zhang, V. Jacobson, and K. Nichols. A two-bit differentiated services architecture for the internet. Internet Draft, Internet Engineering Task Force, 1997.