Flow-based Multistage Co-allocation Service

Sudeepth Anand,[2] Srikanth Yoginath,[2] Gregor von Laszewski,[1,*]
Beulah Alunkal,[1,2] Xian-He Sun[2]
[1]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60440
[2]Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616
[*]Corresponding Author: gregor@mcs.anl.gov

to be published in:
The 2003 International Conference on Communications in Computing,
June 23-26, 2003, Las Vegas, Nevada, U.S.A.
this preprint is avalable at Argonne National Laboratory, Argonne, IL 60439, U.S.A.
Number ANL/MCS-P1046-0403

Abstract

The concept of co-allocation provides a simple mechanism to request and bind resources in a coordinated fashion in Grids across virtual organization boundaries. We have designed an advanced multistage co-allocation strategy based on resource hierarchies defined through user-specific patterns. The model manages simple flows between resources to perform managed job executions. We demonstrate the usefulness of our model on several examples and discuss advantages and disadvantages of the model.

Keywords: co-allocation, Grid, workflow, Commodity Grid Kit, CoG

1 Introduction

Grids promote a collaborative approach [**?**] to computing that enables the integration of geographically resources as part of a virtual organization [**?**]. Such resources include computers, data repositories, scientific instruments, and advanced display devices [**?**]. To use these resources at the same time as part of an adhoc environment, the user needs to request the resources under quality-of-service guaranties and bind a reservation with the requested resources in order to assemble a cooperatively managed resource pool for the application. Often resources must be reserves and bound at the same time to fulfill large numbers of reservation requests [**?**].

The collaborative assembly of such an adhoc resource pool accessed by the user is known in the Grid community as co-allocation. A typical strategy for co-allocating resources include an iterative process of reservation and binding until a quality constraint defined by the application user is fulfilled.

In order to facilitate complex co-allocations in Grids, a *co-allocation service* is needed that takes care of resource co-allocation under quality-of-service guaranties. This service must be designed under software engineering guidelines promoting usability, convenience, simplicity, platform independence, and scalability.

This paper provides an initial step addressing issues related to the development of a service-based model for co-allocation. We consider advanced allocation strategies that benefit from a hierarchical resource model that may be implicitly imposed through a virtual organization and exposed through a hierarchical resource specification. As a result, better scalability for large numbers of resources is achieved.

The paper is structured as follows. First,

we describe the present Globus Toolkit resource management architecture. Then we introduce the concept of multistage co-allocation and compare it with single-stage co-allocation. Next we describe and compare various co-allocation architectures. We illustrate how to achieve workflow-based job execution using our multistage co-allocation strategy. Finally, we discuss implementation issues.

2 Motivation

Co-allocation of compute resources was introduced by the Globus Toolkit [**?**] in order to reserve multiple supercomputers and to conduct large-scale MPI-based [**?**] calculations across domain boundaries. The toolkit provides an API called DUROC (dynamically updated request online co-allocator) with which a co-allocation service can be designed. DUROC provides neither a protocol nor a language-independent design, and thus is unsuitable for advanced Grid environments that must address language independence. It also uses a nonstandard communication library (Nexus) [**?**] that has been phased out as part of the Globus Toolkit. Hence, co-allocation has not been distributed with protocol-based toolkits such as the Java CoG Kit. Our goal was to demonstrate that one can easily provide a service-based replacement for DUROC that will allow users to use such a service any application through a protocol. We also worked to determine architectural changes that provide added functionality and scalability for large numbers of resources.

In the current DUROC design reserving a large number of independent resources leads to overhead because every machine needs to be contacted and the likelihood that all machines will be available is little. Hence the direct contact promoted by DUROC between client and resource during the resource reservation phase results in additional overhead while not utilizing the different communication speeds in Grids. Our design involves a hierarchy of smart co-allocation services based on natural virtual organization boundaries. By considering changes in inter-resource communication speeds, we reduce the communication costs dramatically. The complexity for reserving resources (under the assumption that a reservation succeeds immediately) is $O(n)$ in the case of DUROC. On the other hand, the lower bound of the reservation in our hierarchical system for large numbers of resources is $O(n \log n)$.

3 Architecture

The main components (see Figure **??**) of the co-allocation service are (a) a selection agent that selects appropriate resources under the clients service constraints (indicating which may take part in a co-allocated computation), (b) a request agent that requests resources for inclusion in the allocation once they have been selected, and (c) a barrier agent that makes the resources available at the same time. Synchronization is achieved by making each resource wait on the barrier call until the barrier service hears from all the resources. A timeout can be specified in order to not block the resources for an indefinite amount of time while preventing deadlock with other co-allocations. Upon failure, each of these agents iteratively tries to lower the resource constraints specified by the client, in order to allocate a suitable set of resources. The request and barrier agent, contact each resource individually.

To use the hierarchies implicitly or explicitly expressed as part of a Grid environment, we need to enhance the single-stage co-allocation service in two ways. In a multistage co-allocation, a co-allocation service forwards requests to other co-allocation services at successive stages. Hence, we must provide autonomous behavior within a co-allocation service that prevents excessive communication with the client. Additionally, we provide a registry within the co-allocation service to which we can register other co-allocation services to construct a hierarchy across virtual organizations (Figure **??**(b)). The co-allocation ser-

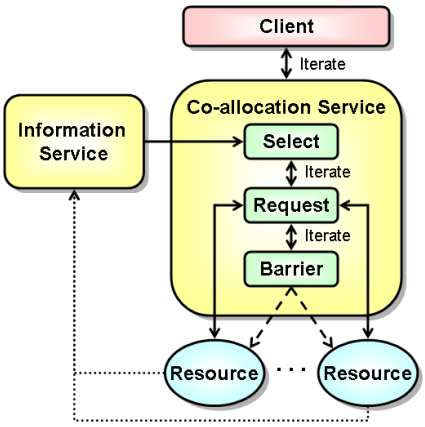

Figure 1: Single-stage co-allocation service

vice has been augmented with the authorization and authentication service that can delegate security credentials between the services to the resources so that permitted use of the resources is supported.

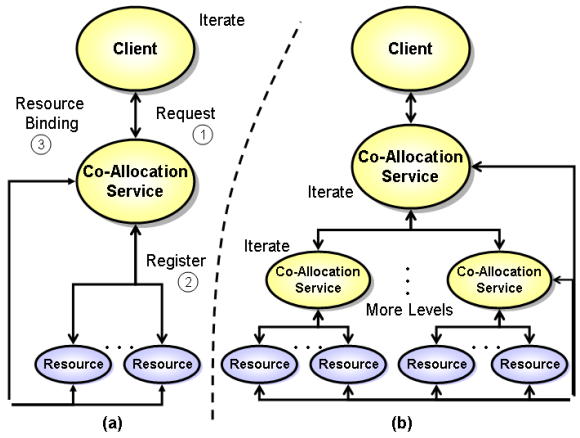

Figure 2: (a) Single-stage co-allocation, (b) Multistage co-allocation

4 Job-based Co-allocation Service

Although our architecture is applicable to any resource, we consider in the rest of the paper only compute resources to simplify our discussion. Job request are first formulated in a portable way using an XML based language as depicted in Figure **??**. Here two resources (hot and cold) are co-allocated, executing a tightly coupled climate model in parallel on the two compute resources. Nesting co-allocations in the `<coallocate>` tag enables one to specify easily a hierarchy.

```
<coallocate name="climate-model">
    <resource="hot.mcs.anl.gov" type="compute">
        <executable="climate.out"/>
        <directory="/gregor/coalloc"/>
        <arguments="europe"/>
    </resource>
    <resource="cold.iit.edu" type="compute">
        <executable="climate.out"/>
        <directory="/vonlaszewski/coalloc"/>
        <arguments="America"/>
    </resource>
</coallocate>
```

Figure 3: Pseudo multirequest specification in XML

To simplify the notation of complex flows, we use in the rest of the paper a notation that is an extended version of the one introduced by the Globus Toolkit Resource Specification Language (RSL) [**?**]. While replacing XML tags with "(" and ")" we denote single-stage co-allocation with a "+" and multistage co-allocation with a "*". Hence, a multirequest is best thought of as a conjunction of individual resource descriptions, or individual RSLs. A multirequest co-allocation specified by the client is parsed by the co-allocation service, the resources are requested, and upon successful allocation through the barrier service (that also checks authentication and authorization) the compute resources can be used in conjunction for a calculation.

5 Single-stage vs. Multistage Co-allocation

Multistage co-allocation has a number of advantages over single-stage co-allocation. The user can feed in a hierarchical job request to the co-allocation service. Thus, job requests can provide a meaning through a structured request by grouping related jobs based on classes formed through contextual or infrastructure

properties. Hence, we can provide better guidance for advanced allocation strategies while using the domain-specific properties of the application and the infrastructure. Furthermore, through multistage co-allocation we can provide for an optimal and efficient resource usage by employing a lazy resource acquisition pattern. This pattern defers resource acquisition to the latest possible time during system execution. Unlike single-stage co-allocation that follows eager acquisition where all the resources are to be acquired in advance, multistage co-allocation can delay resource acquisition by delegating it to later stages. Thus, it can handle resource scarcity better by not forcing all resources to be available at time of execution of the job.

Consider a scenario where a co-allocation job has a large resource requirement and there are not enough resources in the virtual organization to meet them. Single-stage co-allocation cannot proceed until all the resources become available, whereas multistage co-allocation will proceed if there exist some stages in a multistage where the resource requirements can be met. The other stages can wait until their requirements are met as could happen once another stage relinquishes its resources after completion of its requests.

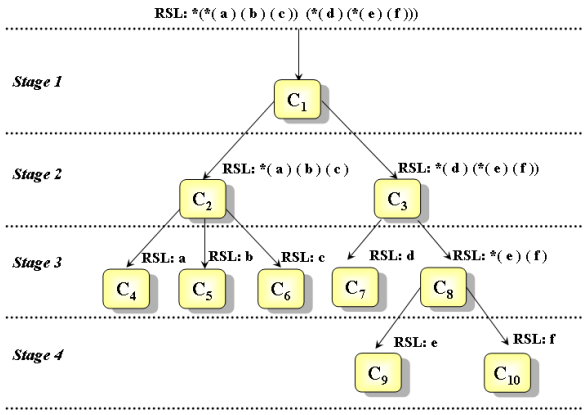

Figure 4: Multistage co-allocation

To clarify these issues, we consider a multistage request as depicted in Figure **??**: $*(*(a)(b)(c))(*(d)(*(e)(f)))$, where $a, b, c, d, e,$ and f are single-stage RSLs. Now let us consider the scenario when the topmost co-allocation service in the processes this request. It understands from the $*$ that this RSL is a multistage RSL, so it parses the RSL, separates the two RSLs $*(a)(b)(c)$ and $*(d)(*(e)(f))$, and submits each to two other co-allocation services. This hierarchical process is continued until single-stage co-allocations are reached at which point the resources are requested and acquired to conduct the calculation specified. Hence, at Stage 3, co-allocation services C_4, C_5, C_6, and C_7 execute $a, b, c,$ and d, respectively. Similarly, at Stage 4, C_9 and C_{10}, execute e and f, respectively.

As a result the user can specify a structured job request that is handled by the Grid infrastructure while using multistage co-allocation. The burden of managing such a structured job is placed on the autonomous co-allocation services. In the following sections we evaluate our multistage co-allocation service and discuss how it can easily be modified to schedule job-oriented workflows.

6 Workflow-based Co-allocation

We discuss two different concepts for achieving workflow-governed co-allocation. The first is implicit flow-based multistage co-allocation, where the workflow is implied by the multistage tree structure. The second concept is explicit flow-based multistage co-allocation, where the workflow dependencies are explicitly specified within the RSL. Our architecture provides the advantage of enhancing the core functionality of the Globus Toolkit with the concept of workflows in a straightforward fashion based on a distributed peer-to-peer model while supporting barrier synchronization within a virtual organization. Other approaches use mostly centralized workflow engines [**?**, **?**, **?**]. We note that recently we described how to formalize workflow management in a coordinated fashion based on the Grid services framework [**?**]. This work is not covered

here.

6.1 Implicit Flow-based Co-allocation

By combining the features of multistage co-allocation and the barrier service, we can achieve a workflow-based co-allocation that enables the user to dictate the job execution flow. This co-allocation is achieved by progressing only when all registered members to a barrier service report back. If the co-allocation service gets a multirequest RSL, meaning that no successive lower stage of co-allocation is required, it immediately calls the barrier before processing the request. On the other hand, if the co-allocation service gets a multistage request, implying that a successive lower stage of co-allocation is required, it delays calling the barrier until it hears from the successive stage. All these factors contribute to stalling the execution of jobs at a co-allocation service until all the jobs at successive lower stages in the subtree have finished execution. Thus, if there is a job a at a lower stage in the subtree than another job b, then job b waits until job a finishes. In essence we achieve a workflow of $a \rightarrow b$.

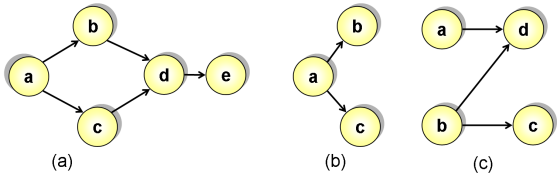

Figure 5: Workflows

To see how we formulate such workflows, consider Figure **??**(a). Here, the workflow specifies that jobs b and c are dependent on job a; job d is dependent on jobs b and c; and job e is dependent on job d. In order to achieve the workflow specified by the user, the following multistage RSL is created: $*(*(*(*(a))(b)(c))(d))(e)$, where a, b, c, d, and e are multirequest RSLs. This multistage RSL is submitted to a co-allocation service, and a multistage execution tree structure is generated. Figure **??** shows the corresponding multistage execution structure. To understand the process, we consider "Stage 4" of the multistage co-allocation tree. This stage has three co-allocators C_6, C_7, and C_8. Co-allocators C_7 and C_8 are leaf nodes, with no lower stages below them. Hence they wait on the barrier before running the jobs b and c, respectively. C_6, not being a leaf node, does not execute the barrier until it submits job a to co-allocator C_9 and subsequently gets the result back from C_9. After getting back the result, C_6 signals C_7 and C_8 out of the barrier. Co-allocators C_7 and C_8 then continue on to execute b and c, respectively. Thus the flow shown in Figure **??** (b) has been achieved. Similar operations take place at every stage of the multistage co-allocation tree, with the final result of achieving the specified workflow shown in Figure **??** (a). This architecture can achieve simple workflows that map from a direct acyclic Graph into a tree.

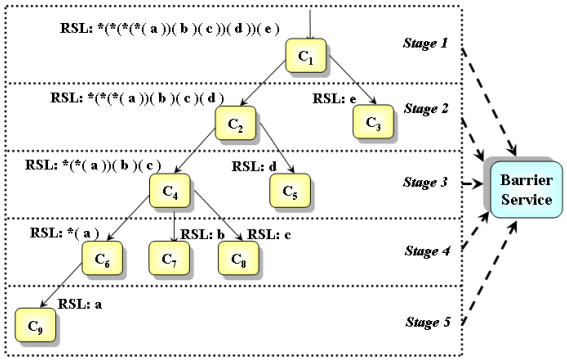

Figure 6: Multistage Tree Structure

A disadvantage of the implicit flow-based multistage allocation services (IFMAS) is that each job executing at any stage in the multistage co-allocation tree waits for the completion of all the jobs in stages lower down the subtree. There might be jobs at a stage that do not depend on some other jobs lower in the subtree, that distinction cannot be made in this architecture.

To clarify this point, we consider the workflow shown in Figure **??**(c). In this example,

job d depends on both jobs a and b, whereas job c depends only on job b. Although the flow can be achieved by using the IMAS architecture, it will lead to a nonoptimal schedule as Figure **??**(b) shows. It is inefficient because job c, which depends only on job b, is made to wait until both jobs a and b complete. Hence, this architecture will not be suitable for many workflows.

6.2 Explicit Flow-based Co-allocation

To enable a larger range of workflows, we developed an architecture that enhances IFMAS with some minor changes. First, we change the barrier service model to a wait-and-signal model. We introduce a unique job-ID for each job. According to this new model a co-allocation service executing a wait call on a job-ID can be released from its waiting state only by a co-allocation service executing a signal call on the same job-ID. Figure **??** (b) depicts which job-IDs propagate down the multistage co-allocation tree. We also modified our multistage RSL to include references to job-IDs. Because we explicitly define the workflows, we call this architecture explicit flow-based multistage co-allocation service (EFMAS).

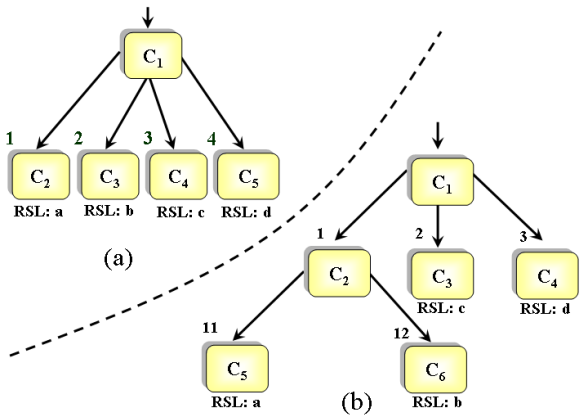

Figure 7: (a) Multistage tree structure, (b) propagation of job-IDs

Using this new architecture, we can achieve the workflow shown in Figure **??**(b) by submitting the multistage RSL: $*(a)(b)(c(wait = 2))(d(wait = 1, 2))$. The resulting multistage tree structure¡ is shown in Figure **??**(c). The co-allocation service C_1 parses it and then submits jobs a, b, c, and d to co-allocation services C_2, C_3, C_4, and C_5, respectively. The multi-request RSL submitted to C_4 is $c(wait = 2)$, which implies that the co-allocator should execute a wait call on job-ID of 2 before running job c. Once C_3 gets the RSL b from C_1, it runs job b, since no wait is specified in its RSL. On completion of job b, C_3 signals on job-ID 2. This signal prompts C_4 out of its waiting state. Similarly C_5 waits on C_2 and C_3 and will execute job d only after it gets signals from C_2 and C_3. As we can see, with little modification this new architecture provides more flexibility and efficiency to the user.

7 Implementation

We developed a prototype implementation based on our architectural design to perform job-based co-allocation through reuse of elementary Grid services such as job execution, file transfer, and security authentication. We access such services through the Java CoG Kit [**?**], which allows the application programmer or middleware developer to readily use a subset of Globus Toolkit Grid services from a higher-level framework. It thus allows for easier and more rapid application development. We augmented in our implementation the workflow co-allocation service with appropriate error-handling routines that react upon failures to provide a more fail-safe environment. The user can decide how to react to failures: either wait for an unbounded time for the necessary resources to become available or retract the job execution after a configurable number of retrials. The user specifies these details within the multistage RSL. For example, `(on_error=2)` specifies a maximum of two retrials for necessary resources, and `(on_error=n)` implies that the current job is a high-priority one and thus the co-allocator is required to wait until the necessary resources becomes available. For

propagating the failure messages to all dependent jobs, we have enhanced our barrier service to give either a positive or a negative reply to all clients depending on the outcome.

8 Conclusions

In this paper we have introduced a new multistage co-allocation strategy based on resource hierarchies. This strategy handles simple workflow-related resource scheduling. We have implemented and successfully tested our strategy, focusing on resources that allow job submissions on the Grid. The implementations achieves better scalability than traditional co-allocation methods for large number of resources. It also provides more flexibility and efficiency. Convenient workflows not based on this research can already be used as part of the Java CoG Kit [**?**] through known under the names GridAnt [**?**], and GSFL [**?**].

9 Acknowledgment

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38. Globus Toolkit research and development have been supported by DARPA, DOE, and NSF. The Globus Toolkit and Globus Project are trademarks held by the University of Chicago.