Handbook of Research on P2P and Grid Systems for Service-Oriented Computing:

Models, Methodologies, and Applications

Nick Antonopoulos
University of Surrey, UK

George Exarchakos
University of Surrey, UK

Maozhen Li
Brunel University, UK

Antonio Liotta
University of Essex, UK

Volume I

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE
Hershey · New York

Director of Editorial Content:	Kristin Klinger
Director of Book Publications:	Julia Mosemann
Development Editor:	Christine Bufton
Publishing Assistant:	Kurt Smith
Typesetter:	Carole Coulson
Quality control:	Jamie Snavely
Cover Design:	Lisa Tosheff
Printed at:	Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com/reference

Library of Congress Cataloging-in-Publication Data

Handbook of research on P2P and grid systems for service-oriented computing : models, methodologies and applications / Nick Antonopoulos ... [et al.].
p. cm.
Includes bibliographical references and index.
Summary: "This book addresses the need for peer-to-peer computing and grid paradigms in delivering efficient service-oriented computing"--Provided by publisher.
ISBN 978-1-61520-686-5 (hardcover) -- ISBN 978-1-61520-687-2 (ebook) 1.
Peer-to-peer architecture (Computer networks)--Handbooks, manuals, etc. 2.
Computational grids (Computer systems)--Handbooks, manuals, etc. 3. Web
services--Handbooks, manuals, etc. 4. Service oriented architecture--
Handbooks, manuals, etc. I. Antonopoulos, Nick.
TK5105.525.H36 2009
004.6'52--dc22
2009046560

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

All work contributed to this book is new, previously-unpublished material. The views expressed in this book are those of the authors, but not necessarily of the publisher.

Chapter 20

Resource Co-Allocation in Grid Computing Environments

Marco A. S. Netto
The University of Melbourne, Australia

Rajkumar Buyya
The University of Melbourne, Australia

ABSTRACT

One of the promises of Grid Computing is to enable the execution of applications across multiple sites. Several multi-site applications require simultaneous access to resources hosted on autonomous domains; this problem is known as resource co-allocation. Projects working on resource co-allocation face four major problems: distributed transactions, fault tolerance, inter-site network overhead, and schedule optimization. Although resource co-allocation is fundamental for Grid Computing, no survey has covered the current projects, solutions, and open challenges on this topic. Therefore, in this chapter, the authors describe the challenges on resource co-allocation, present the projects developed over the last decade, and classify them according to their similar characteristics. In addition, the authors discuss open research issues and trends such as negotiation, advance reservations, and rescheduling of multi-site applications.

INTRODUCTION

One of the promises of Grid Computing is to enable the execution of applications across multiple sites. Some of these applications require coordinated access to resources managed by autonomous entities. This coordinated access is known as *resource co-allocation*. There are two main classes of applications that require resource co-allocation: parallel applications with inter-process communication, and workflow applications. Parallel applications with inter-process communication require all resources to be available at the same time, whereas workflows constitute the execution of tasks with precedence constraints, i.e. resources have to be available in a certain order. Although both application classes require co-allocation, in the Grid computing community, the term *co-allocation* usually refers to the *simultaneous access to resources hosted by autonomous providers* (Czajkowski, Foster, & Kesselman,

DOI: 10.4018/978-1-61520-686-5.ch020

1999). The coordinated access to resources by tasks with precedence constraints is referred as *workflow scheduling* (Yu & Buyya, 2005). In this work, we follow the Grid computing community definition.

The two main reasons for executing applications on multiple sites are: (i) the lack of special resources in a single administrative domain, such as devices for generating data, visualization tools, and supercomputers; and (ii) reduce response time of parallel applications by increasing the number of resources (Czajkowski et al., 1998). However, there are other applications that require co-allocation. Conference and multimedia users engaged in activities, such as scientific research, education, commerce, and entertainment, require co-allocation of multiparty real-time communication channels (Ferrari, Gupta, & Ventre, 1997; Xu, Nahrstedt, & Wichadakul, 2001). Data-intensive applications use co-allocation to collect data from multiple sources in parallel (Vazhkudai, 2003; Yang, Yang, Wang, Hsu, & Li, 2007). In addition, increasing the number of resources is a requirement of large-scale applications demanding considerable amounts of memory, storage, and processing power. Examples of these applications are semiconductor processing (Takemiya et al., 2006) and computational fluid dynamics (Dong, Karniadakis, & Karonis, 2005).

Various projects have developed software systems with resource co-allocation support for large-scale computing environments, such as TeraGrid, Distributed ASCI Supercomputer (DAS), and Grid'5000. TeraGrid has deployed Generic Universal Remote (GUR) (Yoshimoto, Kovatch, & Andrews, 2005) and Highly-Available Resource Co-allocator (HARC) (Maclaren, Keown, & Pickles, 2006), the DAS project has developed KOALA (Mohamed & Epema, 2005), and Grid'5000[1] has relied on the OAR(Grid) scheduler (Capit et al., 2005) to allow the execution of applications requiring co-allocation. There are also projects dedicated to the management of network links, such as G-lambda (Takefusa et al., 2006).

Although resource co-allocation is fundamental for Grid Computing, no survey has covered the four major challenges in this field: distributed transactions, fault tolerance, inter-site network overhead, and schedule optimization. Therefore, in this chapter, we describe the challenges on resource co-allocation, present some of the efforts and projects developed over the last decade, and classify them according to their similar characteristics. In addition, we discuss open research issues and trends such as negotiation, advance reservations, and rescheduling of multi-site applications.

BACKGROUND

Existing work on resource co-allocation have focused on four research problems: distributed transactions, fault tolerance, inter-site network overhead, and schedule optimization. Most of the projects we present in this chapter have considered at least two of these problems. Resource co-allocation involves the interaction of multiple entities, namely clients and resource providers. Multiple clients may ask for resources at the same time from the same providers. This situation may generate deadlocks if the resource providers use a locking procedure; or livelock if there is a timeout associated with the locks. Therefore, there has been research on protocols to handle distributed transactions in order to avoid deadlocks and livelocks, and minimize the number of messages during these transactions.

Another common problem in the resource co-allocation field is that a failure in a single resource compromises the entire execution of an application that requires multiple resources at the same time. One approach to minimize this problem is defining a fault tolerance strategy that notifies applications of a problem with a resource. A software layer could then provide the application with a new resource, or discard the failed resource if it is not essential.

One of the main problems when executing applications over different clusters is the inter-site network overhead. Several parallel applications require process communication, which may become a bottleneck due to the high latency of wide-area networks. Therefore, it is important to evaluate the benefits of multi-site execution and develop techniques for mapping application processes considering communication costs.

Scheduling multi-site applications is more complex than scheduling single-site applications due to the tasks' time dependency. In addition, as some applications have more flexibility on how to map tasks to resources, the scheduler has to analyze more mapping options. For parallel applications with process communication, the scheduler also has to take into account the network overhead. Moreover, the scheduling of a co-allocation request depends on the goals and policies of each resource provider. Figure 1 illustrates a typical scenario with a user performing a co-allocation based on advance reservations using three resource providers

When implementing and deploying a software system that supports resource co-allocation, developers initially face the first three mentioned problems. Once a system is in production, the schedule optimization becomes one of the most important issues. Most of the work has been on schedule optimization, mainly evaluated by means of simulations. In the next section, we describe in detail the solutions proposed for these four major problems in resource co-allocation. We also give an overview of each project before detailing their solutions. Some projects, especially those with middleware implementation, have faced more than one challenge. For these projects, we have included a section with a comparison of their features and limitations.

CHALLENGES IN RESOURCE CO-ALLOCATION

We have classified the existing work on resource co-allocation for Grid Computing according to the

Figure 1. Example of a user with three advance reservations within the scheduling queues of multiple resource providers

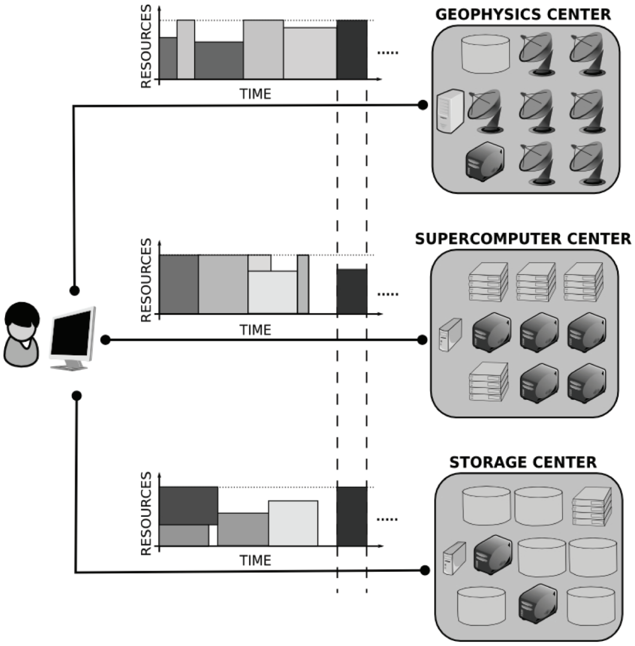

Table 1. Summary of issues and challenges and solutions for resource co-allocation

Issues/Challenges	Description	Solutions

Distributed Transactions
Fault Tolerance
Network Overhead
Schedule Optimization
	Prevention of deadlocks and livelocks; Reduction of messages during transactions.
	Hardware and software failures; Coordinated allocation.
	Evaluation of inter-site communication; Response time reductions.
	Increase system utilization; Reduce user response time.
		Two- and Three-phase Commit Protocol;Order-based Deadlock Prevention Protocol; Polling.
		Advance reservations;
		Backtracking;
		User's fault recovery strategy; Flexible resource selection.
		Topology-aware mapping; Use of network links information; Proximity of data location to resources.
		Advance reservations;
		Network-aware scheduling;
		Rescheduling and negotiation support.

Table 2. Summary of methods and goals for distributed transactions in Grid environments

Method	Goals
Two-phase Commit Protocol **Three-phase Commit Protocol** **Order-based Deadline Prevention Protocol** **Polling**	Prevent gathering partial number of resources. Prevent deadlocks and live locks; Support messages to be lost and delayed. Prevent deadlocks and livelocks. Prevent deadlocks and livelocks; Remove requirements of ordering resources; Support asymmetric communication

four major challenges. In Table 1, we have a short description and solutions for each research topic. Some of the projects have focused on more than one aspect of resource co-allocation. However, the description of such projects is in the section of the research topic with their most significant contribution.

Distributed Transactions

The research on the management of Distributed Transactions involves the development of protocols to avoid deadlocks and livelocks that may occur during the co-allocation process. In addition, the protocols aim to minimize the number of messages during these transactions. A deadlock may happen when the following two conditions are true: (i) multiple clients ask for resources at the same time from the same resource providers and (ii) these resource providers work with schedulers that lock themselves to serve requests. Similar to the two conditions of a deadlock, a livelock happens when the schedulers in the resource providers have a timeout associated with the locks. The distributed transactions research field has been quite active in database communities (Bernstein & Goodman, 1981). However, in this section we will describe projects interested in this area focusing on resource co-allocation for Grid Computing. Table 2 summarizes the methods and goals used by the researchers on this topic.

Kuo & Mckeown, 2005 presented a protocol specification, in terms of messages and finite state machines, for advance reservations and co-allocation as a requirement of the RealityGrid project. The RealityGrid users execute interactive simulations and may need to modify their simulation parameters when the simulation entered uninteresting regions of the search space. They also need to transfer data to the visualization system and need to know when their simulations start. In addition they require the co-allocation and cancellation of reservations. Their co-allocation protocol is an extension of the two-phase commit protocol with the support for cancellations that may occur at any time. Their protocol supports nested configuration, i.e. a resource can be a co-allocator for other set of resources. However, it has no support for atomic transactions. Therefore, a transaction may reach a state where a reservation executes on some resources, while other reservations are cancelled. They deal with race conditions on the request phase and propose a non-blocking protocol with a timeout mechanism.

Park, 2004 introduced a decentralized protocol for co-allocating large-scale distributed resources, which is free from deadlocks and livelocks. The protocol does not require the applications to use any information other than their own local resource allocation states. The proposed protocol ensures that every application can co-allocate the resources specified by one of its goal states through a series of requests, while preventing the application from getting involved in deadlocks or livelocks during the allocation process. The protocol is based on the Order-based Deadlock Prevention Protocol ODP2, but with parallel requests in order to increase its efficiency. The protocol uses the IP address as the unique local identifier to order the resources. Another approach to avoid deadlock and livelock is the exponential back-off mechanism, which does not require the ordering of resources. Jardine, Snell, & Clement, 2001 investigated such a mechanism for co-allocating resources.

Czajkowski, Foster, Kesselman, Sander, & Tuecke, 2002 proposed the Service Negotiation and Acquisition Protocol (SNAP), which aims at managing access to and use of distributed computing resources in a coordinated fashion by means of Service Level Agreements (SLAs). SNAP coordinates the resource management through three types of SLAs, which separate task requirements, resource capabilities, and biding of tasks to resources. From the moment users identify target resources to the moment when they submit tasks, other users may access the chosen resources. This happens because information obtained from the providers may be out-of-date during the selection and actual submission of tasks. In order to solve this problem, Haji, Gourlay, Djemame, & Dew, 2005 developed a Three-Phase commit protocol for SNAP-based brokers. The key feature of their protocol is the use of *probes*, which are signals sent from the providers to the candidates interested in the same resources to be aware of resource status' changes.

Takefusa, Nakada, Kudoh, Tanaka, & Sekiguchi, 2007 developed a resource co-allocation framework, called GridARS (Grid Advance Reservation-based System), based on advance reservation, which utilizes WSRF/GSI (Web Services Resource Framework/Grid Security Infrastructure) and a Two-Phase Commit (2PC) Protocol. The motivation of their work is the human interaction still required for co-allocating resources across different administrative domains with different Grid technologies. Their 2PC protocol uses a polling approach from the client to the server. The authors argue that although there is a communication overhead between the client and server due to the polling, this non-blocking approach allows asymmetric communication, and hence, the client does not need a global address. Moreover, it eliminates firewall problems, avoids hang-ups because of server or client side troubles, and enables the recovery of each process from the failure. They evaluated their framework on top of Globus by co-allocating computing and network resources from 7 sites in Japan and 3 sites in US. For the resources in US, they used a wrapper on top of the Highly-Available Robust Co-allocator (Maclaren et al., 2006).

Maclaren et al., 2006 discussed the problem of resource co-allocation, in particular focusing on fault tolerance, and presented a co-allocation system called HARC (Highly-Available Robust Co-allocator). Their system uses advance reservation to co-allocate resources and relies on a Three-Phase Commit Protocol based on Paxos consensus algorithm (Gray & Lamport, 2006). In this algorithm, the coordinator responsible for receiving confirmation answers from resource providers is replaced with a set of replicated processes called Acceptors. A leader process coordinates the acceptor processes to agree on a value or condition. Any acceptor can act as

Table 3. Summary of methods used for fault tolerance in resource co-allocation

Method	Goals
Advance Reservations	Ensure all resources are available at required time.
Backtracking	Replace failed/unavailable resources.
User's fault recovery strategy	Users specify their own recovery strategy.
Flexible resource selection	Ignore optional resources; Specify alternative resources.

the leader and replace the leader if it fails. This algorithm allows messages to be lost, delayed or even duplicated. Therefore, the Paxos Commit protocol is a valuable algorithm when considering the fault tolerance for distributed transactions in order to co-allocate resources in Grids.

Azougagh, Yu, Kim, & Maeng, 2005 introduced the Availability Check Technique (ACT) to reduce the conflicts during the process of resource co-allocation. The conflicts are generated when multiple jobs are trying to allocate two or more resources in a crossing way simultaneously, resulting in deadlocks, starvations, and livelocks. They described an analogy of this co-allocation problem with the dinning philosophers' problem. In their solution, jobs wait for updates from resource providers until they fulfill their requirements. They evaluated ACT on top of All-or-Nothing protocol, in which all the allocated resources are released if one of them cannot be allocated, and Order-based Deadlock Prevention protocol (ODP2), in which there is an assumption of a global linear order of the resources. Once the resources are allocated, the job starts running. Therefore, their work does not rely on advance reservations and consequently does not provide guarantees for the start time of user applications.

Fault Tolerance

Hardware and software failures are common in Computational Grids due to their complexity in terms of resource autonomy, heterogeneity, and scalability. Improper configuration, network error, and authorization difficulties are examples of problems that affect the execution of an application. For a multi-site application, failures are even more frequent since a failure in a single resource may compromise the entire execution. In this section, we describe some of the projects working on fault tolerance for multi-site applications. Table 3 summarizes the main methods used for fault tolerance in resource co-allocation.

Czajkowski et al., 1998 introduced a Resource Specification Language (RSL) to allow users to submit co-allocation requests to a broker. This broker contains an entity called resource co-allocator responsible for producing and submitting multiple sub-requests to each determined resource manager. In order to co-allocate resources, their system relied on the current availability of the resources and queue-time estimations of the resource providers. Using this approach, an RSL request could not provide guarantees that the resources would be available at the same time. Czajkowski et al., 1998 concluded that such an approach was not scalable since many failures were common, e.g. improper configuration, network error, and authorization difficulties.

Czajkowski, Foster, & Kesselman, 1999 proposed a layered architecture to address failures for co-allocation requests. During the allocation phase, users should include a barrier function in their application for starting purposes. They introduced two methods for co-allocation: Atomic Transaction and Interactive Transaction. In the atomic transaction, all the required resources were specified at the request time. The request succeeds if all resources are allocated. Otherwise, the request fails and none of the resources is acquired. The user could modify the co-allocation content until the request initializes. The authors argued that for large-scale applications this approach was not appropriate since a resource failure usually cannot be detected until the application starts. Therefore, they proposed the interactive transaction method, in which the content of a co-allocation request could be modified via add, delete, and substitute operations. In order to simplify the reconfiguration of a request, resources could be classified in three categories: required (failure or timeout of this kind of resource causes the entire computation to be terminated---similar to atomic operation); interactive (failure or timeout of a resource results in a call-back to the application, which can delete or substitute to another resource---i.e. the resource is not essential or it is easy to find replacements); optional (failure or timeout is ignored). The authors performed experiments using both methods and concluded that interactive transaction is more suitable for large-scale applications in Computational Grids.

Foster et al., 1999 proposed and described the prototype of the Globus Architecture for Reservation and Allocation (GARA). This prototype aimed to provide a platform with support for Quality of Service (QoS) guarantees. Based on the support of advance reservations, the authors argued that the number of candidate resources could be larger. That is because the users could consider more plans, i.e. they did not simply rely on the current resource availability. However, having this possibility of creating more plans requires more efficient scheduling heuristics. Regarding fault tolerance, GARA had the concept of backtracking, in which when there was a resource failure, it was possible to try other resources until the request succeeded or failed.

Sinaga, Mohamed, & Epema, 2004 designed and implemented an extension for the DUROC system to enhance two functionalities: resource-brokering and fault tolerance. In DUROC, users had to specify where their job components had to be executed. The authors modified the system to allow the scheduler to select the target sites of the job components. Moreover, in terms of fault tolerance, once a job could not get the resources, DUROC considered it as failed. The authors extended the scheduler such that it could keep trying to schedule jobs until they could get all the required resources, or until the number of tries achieved a certain threshold.

Roblitz & Reinefeld, 2005 presented a framework to manage reservations for applications running concurrently on multiple sites and applications with components that may be linked by temporal or spatial relationships, such as job flows. They defined and described co-reservations along with their life cycle, and presented an architecture for processing co-reservation requests with support for fault tolerance. When handling confirmed co-reservations, part of the requested resources may not be available, therefore alternative resources should substitute them. If it is not possible, a best-effort option could be followed or the request should be canceled. Users define such a behavior through a fault recovery strategy in the request specification. The authors also discussed the concept of virtual resources to provide the user with a consistent view on multiple reservations. Therefore, it is possible to have modifications of the reservations in a transparent way for users.

One of the main problems when executing message passing parallel applications over different clusters is the network overhead. Due to the inter-process communication, the wide-area networks may degrade the performance of these parallel applications, thus generating a considerable delay. Therefore, it is important to evaluate the benefits of multi-site executions and investigate techniques for mapping application processes considering communication costs. Several researchers have investigated the benefits of multi-site executions using different methods and testbeds. Network overhead has also been investigated for co-allocation data and processors. Table 4 summarizes the main methods for evaluating network overhead for multi-site applications.

Table 4. Summary of methods used for evaluating network overhead for multi-site applications

Method	Goals
Application specific	Evaluate specific application properties.
Data-intensive applications	Consider transfer of large amounts of data.
Simulation-based evaluation	Evaluate wide range of parameters and scenarios.
Real-testbed-based evaluation	Evaluate network in real conditions.
Topology-aware mapping	Consider network heterogeneity to map tasks.

Inter-Site Network Overhead

The Message Passing Interface (MPI) has been broadly used for developing parallel applications in single site environments. However, executing these applications on multi-site environments imposes different challenges due to network heterogeneity. Intra-site communication has much lower latency than inter-site communication. There are several MPI implementations, such as MPICH-VMI (Pant & Jafri, 2004), MPICH Madeleine (Aumage & Mercier, 2003), and MPICH-G2 (Karonis, Toonen, & Foster, 2003), that take into account this network heterogeneity and simplified the application development process.

Ernemann, Hamscher, Schwiegelshohn, Yahyapour, & Streit, 2002 studied the benefits of sharing jobs among independent sites and executing parallel jobs in multiple sites. When co-allocating resources, the scheduler looks for a site that has enough resources to start the job. If it is not possible, the scheduler sorts the sites in a descending order of free resources and allocates those resources in this order to minimize the number of combined sites. If it is not possible to fit the job, the scheduler queues the job using Easy Backfilling (Mu'alem & Feitelson, 2001). Through discrete event driven simulations, the authors varied the network overhead from 0 to 40% and concluded that multi-site applications reduce average weighed response time when the communication overhead is limited to about 25%.

Bucur & Epema, 2003a investigated the feasibility of executing parallel applications across wide-area systems. Their evaluation, which is based on simulations, has as input parameters the structure and size of jobs, scheduling policy, and communication speed ratio between intra- and inter-clusters. They used mean job response time as the main metric as a function of system utilization. The simulation setup is based on the Distributed ASCI Supercomputer (DAS) system composed of five clusters. Either the user or the scheduler can decide the job components to be submitted to each cluster. For the latter case, the scheduler maps the job components by their decreasing order of size. Users can also specify only the total number of processors required by their applications. The scheduler has three placement policies: Cluster-Filling, Load-Balancing on Smallest number of clusters, and Load-Balancing on All clusters. In the first policy, the scheduler submits the job components to the clusters that have the largest number of idle processors. The other two policies balance the number of processors for each cluster. Based on their study, they concluded that: (i) the user response time increases when users specify the size of each job component and the target clusters; (ii) even when the ratio between inter- and intra-cluster is 50, it is worth co-allocating resources instead of waiting for all resources to be available in a single cluster; and (iii) when the scheduler can split jobs and choose their target clusters, it should balance the load to accommodate jobs with less splitting flexibility.

Dong et al., 2005 showed a performance evaluation of two parallel applications in biological and physical sciences on the TeraGrid environment: the simulation of blood flow in the entire human arterial tree, and the direct numerical simulation of bluff-body turbulent wake flows. They used 256 processors of two sites located in US. They investigated the impact of the network communication on the application's speedups according to the number of processors in a single and two sites. They concluded that multi-site execution is a viable alternative to reduce the response time of large-scale scientific experiments.

Jones, III, & Shrivastava, 2006 proposed scheduling strategies that use available information of the network link utilization and job communication topology to define job partition sizes and job placement. Their motivation for using co-allocation is to reduce the user response time by merging fragments from the scheduling queues of multiple resource providers. Rather than assuming a fixed amount of time for all inter-cluster communication or assigning execution time penalties for the network overhead, the authors considered that inter-cluster bandwidth changes over time due to the number and duration of multi-site executions in the environment. Therefore, they explored the scheduling of multiple co-allocation jobs sharing the same computing infrastructure. Their scheduling policy for job selection is Fit-Processors-First-Served, which is similar to Easy Backfilling (Mu'alem & Feitelson, 2001) but without the restriction of not delaying the job in the head of the queue. As for the co-allocation strategies, the authors investigated:

- **First-Fit:** Which performs resource co-allocation by assigning tasks starting with the cluster having the largest number of free nodes and does not use any information of neither the job communication characterization nor network link saturation
- **Link saturation level threshold only:** Which is similar to First-Fit but discards clusters with saturated links;
- **Link saturation level threshold with constraint satisfaction:** Which tries to put jobs into a large portion of a single cluster (e.g. 85% of resources); and Integer Constraint Satisfaction, which uses jobs' communication characterization and current link utilization to prevent link saturations.

Through simulations, Jones et al., 2006 concluded that it is possible to reduce multi-site applications' response time by using information of network usage and jobs' network requirements. In addition, they concluded that this performance gain depends heavily on the characteristics of the arriving workload stream.

Mohamed & Epema, 2005 addressed the problem of co-allocating processors and data. They presented two features of their co-allocating scheduler, namely different priority levels of jobs and incrementally claiming processors. The scheduler may not be able to find enough resources when jobs are claiming for resources. In this case, if a job j claiming for resources has high priority, the scheduler verifies whether the number of processors used by low priority jobs is enough to serve the job j. If it is enough, the scheduler preempts the low priority jobs in a descending order until enough resources are released. The scheduler moves the preempted jobs into the low priority placement queue. The scheduler uses the Close-to-Files (CF) job-placement algorithm to select target sites for job components (Mohamed & Epema, 2004). The CF algorithm attempts to place the jobs in the sites where the estimated delay of transferring the input file to the execution sites is minimal.

Schedule Optimization

Most of the work on resource co-allocation for Grid Computing focuses on how to optimize the schedule of multi-site applications. Scheduling co-allocation requests is more complex than scheduling single site requests due to the tasks' time dependency. Moreover, some parallel applications have the flexibility on how they can be decomposed to run in multiple sites. In case of

Table 5. Summary of methods and scenarios for schedule optimization of co-allocation requests

Methods/Scenarios	Goals
Advance reservation	Ensure all resources are available at the same time.
Non-advance-reservation	Support for middleware without advance reservations; Reduce resource fragmentation in scheduling queues.
Global queue	Simplify evaluation. Focus on small scale environments
Autonomous queues	Consider local load and scheduling policies for multiple resource providers.
Network-aware scheduling	Consider inter-site network overhead
On-line scheduling	Make scheduling decisions based only already accepted requests
Batch-mode scheduling	Make decisions knowing all requests a priori
Negotiation support	Achieve common goals of users and resource providers
Rescheduling support	Reduce resource fragmentation and user response time; Increase system utilization

parallel applications with process communication, the scheduler has to take into account the network overhead. Table 5 summarizes the main methods and environments for optimizing the schedule of co-allocation requests.

Snell et al., 2000 investigated the importance of using advance reservations for scheduling Grid jobs, rather than periodically blocking resources dedicated to Grid usage. They defined three scheduling strategies for co-allocation requests: (i) Specified co-allocation, where users specify the resources and their location; (ii) General co-allocation, in which users do not specify the resource location; and (iii) Optimal scheduling, in which the scheduler tries to determine the best location for every required resource in order to optimize cost, performance, response time or any other metric specified by the user. They evaluated the impact of using advance reservations for meta jobs against reserving periods for external usage. They concluded that the former approach is a viable solution for co-allocating resources for Grid jobs.

Alhusaini, Raghavendra, & Prasanna, 2001; Alhusaini, Prasanna, & Raghavendra, 2000 proposed a two-phase approach for scheduling tasks requiring resource co-allocation. The first phase is an off-line planning where the scheduler assigns tasks to resources assuming that all the applications hold all the required resources for their entire execution. The second phase is the run-time adaptation where the scheduler maps tasks according to the actual computation and communication costs, which may differ from the estimated costs used in the first phase. In addition, applications may release a portion of the resources before they finish. The authors considered the scheduling of a set of applications rather than a single one (batch mode). Their optimization criterion was to minimize the completion time of the last application, i.e. the makespan. They modeled the applications as Directed Acyclic Graphs (DAGs) and used graph theory to optimize the mapping of tasks.

Ernemann, Hamscher, Streit, & Yahyapour, 2002 studied the effects of applying constraints for job decomposition when scheduling multi-site jobs. These constraints limit the number of processes for each site (lower bound) and number of sites per job. When selecting the number of processors used in each site, they sort the sites list by the decreasing number of free nodes in order to minimize the number of fragments for the jobs. The decision of using multi- or single-site to execute the application is automatic and depends on the load of the clusters. In their study, a lower bound of half of the total number of available resources appeared to be beneficial in most cases. Their evaluation considers the network overhead for multi-site jobs. They summarized the overheads caused by communication and data migration as an increase of the job's run time.

Azzedin, Maheswaran, & Arnason, 2004 proposed a co-allocation mechanism that requires no advance reservations. Their main argument for this approach is the strict timing constraints on the client side due to the advance reservations, i.e. once a user requests an allocation, the initial and final times are fixed. Consequently, advance reservations generate fragments that the schedulers cannot utilize. Furthermore, the authors argued that a resource provider can reject a co-allocation request at any time in favor of internal requests, and hence the co-allocation would fail. Their schema, called synchronous queuing (SQ), synchronizes the subtasks at the scheduling cycles, or more often, by speeding them up or slowing them down.

Li & Yahyapour, 2006 introduced a negotiation model that supports co-allocation. They extended a bilateral model, which consists of a negotiation protocol, utility functions or preference relationships for the negotiating parties, and a negotiation strategy. For the negotiation protocol, the authors adopted and modified the Rubinstein's sequential alternating offer protocol. In this protocol, players bargain at certain times. For each period, one of the players proposes an agreement and the other player either accepts or rejects. If the second player rejects, it presents an agreement, and the first player agrees or rejects. This negotiation continues until an agreement between the parties is established or the negotiation times out. As it may take several rounds to find a common time slot between resource providers, the authors introduced the non-binding state, in which neither negotiation parties need to commit to an agreement unless all the parties agree to commit. They evaluated the model through simulations with different input parameters for prices, negotiation behaviors, and optimization weights.

Sonmez, Mohamed, & Epema, 2006 presented two job placement policies that take into account the wide-area communication overhead when co-allocating applications across multiple clusters. The first policy is the Cluster Minimization in which users specify how to decompose the jobs and the scheduler maps the maximum job components in each cluster according to their processor availability (more processors available first). The second policy is Flexible Cluster Minimization in which users specify only the number of required processors and the scheduler fills the maximum number of processors in each cluster. The main goal of these two policies is to minimize the number of clusters involved in a co-allocation request in order to reduce the wide-area communication overhead. The authors implemented these policies in their system called KOALA and evaluated several metrics, including average response time, wait time and execution time of user applications. Their work does not use advance reservations, so at time intervals (4 seconds in their experiments), the scheduler looks for idle nodes in the waiting queues of co-allocation requests. If the placement of a job fails, KOALA places the job at the tail of the waiting queue. For each job in the queue, the system records its number of placement tries, and when this number achieves a certain threshold, the job is considered as rejected.

Bucur & Epema, 2007, 2003b investigated scheduling policies on various queuing structures for resource co-allocation in multi-cluster systems. They evaluated the differences of having single global schedulers, only local schedulers and both schedulers together, as well as different priorities for local and meta jobs. They used First Come First Serve in the scheduling queues. They have concluded that multi-site applications should not spend more than 25% of their time with wide-area communication and that there should be restrictions on how to decompose the multi-site jobs in order to produce better schedules.

Elmroth & Tordsson, 2007 modeled the co-allocation problem as a bipartite graph-matching problem. Tasks can be executed on specific resources and have different requirements. Their model relies on advance reservations with flexible time intervals. They explored a relaxed notion of simultaneous start time, where jobs can start with a short period of difference. When a resource provider cannot grant an advance reservation, it suggests a new feasible reservation, identical to the rejected one, but with a later start time. They presented an algorithm to schedule all the jobs within the start window interval, which tries to minimize the jobs' start time.

Decker & Schneider, 2007 investigated resource co-allocation as part of workflow tasks that must be executed at the same time. They extended the HEFT (Heterogeneous Earliest-Finish-Time) algorithm to find a mapping of tasks to resources in order to minimize the schedule length (makespan), to support advance reservations and co-allocation, and to consider data channel requirements between two activities. They observed that most of the workflows were rejected because no co-allocation could be found that covered all activities of a synchronous dependency or because there was not enough bandwidth available for the data channels. Therefore, they incorporated a backtracking method, which uses not only the earliest feasible allocation slot for each activity that is part of a co-allocation requirement, but all possible allocation ranges as well.

Netto & Buyya, 2008 proposed a resource co-allocation model that supports rescheduling. The model allows the schedulers to change the start time of the job components and remap the number of processors used in each site. The authors evaluated the impact of rescheduling co-allocation requests due to the inaccurate runtime estimations provided by users. Their results show that local jobs may not fill all the fragments in the scheduling queues and hence rescheduling co-allocation requests reduces the response time of both local and multi-site jobs. Moreover, they observed that, in some scenarios, the processor remapping operation increases the chances of placing the tasks of multi-site jobs into a single cluster, thus eliminating the inter-cluster network overhead.

SYSTEMS WITH RESOURCE CO-ALLOCATION SUPPORT

In previous section, we described the existing solutions for each challenge on resource co-allocation. Although several projects have focused on one aspect of resource co-allocation, some research groups have faced more than one challenge, in

Table 6. Summary of the main features and limitations of six middleware systems that support resource co-allocation for each challenge

System	Distributed Transactions	Fault Tolerance	Network Overhead	Schedule Optimization
GARA **KOALA** **HARC** **OAR** **GridARS** **JSS**	Two-phase commit protocol None (processors claimed incrementally) Three-Phase Commit Protocol based on Paxos consensus algorithm. One-phase: All-or-none approach. Two-Phase Commit Protocol with polling. Negotiation with multiple interactions without blocking resource.	Use of alternative/optional resources until application receives resources. Backtracking. Flexible resource selection until application receives resources. Only on the allocation transaction phase. Unavailable for co-allocation requests. On the allocation transaction phase with a rollback mechanism. Only on the moment to find a feasible schedule.	User pre-selects sets of resources Close-to-Files policy User pre-selects sets of resources User pre-selects sets of resources User pre-selects sets of resources Network-aware scheduling based on ranking.	Advance reservation based scheduling. Scheduling based on incremental processor claiming. No use of advance reservations. Advance reservation based scheduling. Metascheduler makes decisions based on timetables offered by providers. Reservations can be modified to support rescheduling. Advance reservation based scheduling using first fit. Advance reservation based scheduling Advance reservation based scheduling with multiple interactions between metascheduler and provider to find a common job start time.

particular groups that developed middleware systems with resource co-allocation support. In this section, we present a brief description of the existing systems that support resource co-allocation and compare them according to their features and limitations based on the challenges we have presented (Table 6).

- **GARA:** The Globus Architecture for Reservation and Allocation (GARA) enables applications to co-allocate resources, which include networks, computers, and storage (Foster et al., 1999). GARA uses advance reservations to support co-allocation with Quality-of-Service (QoS). GARA was one of the first projects to consider QoS for co-allocation requests.
- **OAR:** OAR is the batch scheduler that has been used in Grid'5000 (Capit et al., 2005; Cappello et al., 2005). OAR uses a simple policy based on all-or-none approach to co-allocate resources using advance reservations. One of the main design goals of OAR is the use of high level tools to keep low software complexity.
- **KOALA:** Is a grid scheduler that has been deployed on the DAS-2 and the DAS-3 multi-cluster systems in the Netherlands (Mohamed & Epema, 2005, 2008). KOALA's users can co-allocate both processors and files located in autonomous clusters. KOALA supports malleable jobs, which can receive messages to expand and reduce the number of processors at application runtime, and has fault tolerance mechanisms, which are used to deal with reliability of the grid resources.
- **HARC:** Highly-Available Resource Co-allocator (HARC) is a system for reserving multiple resources, which can be processors and lightpaths, in a coordinated fashion (Maclaren et al., 2006). HARC has been deployed in several computing infrastructures, such as TeraGrid, LONI (Louisiana Optical Network Initiative), and UK National Grid Service. One of the main features of HARC is its Three-Phase Commit Protocol based on Paxos consensus algorithm (Gray & Lamport, 2006), which increases fault tolerance during the allocation process.
- **GridARS:** Grid Advance Reservation-based System (GridARS) is a co-allocation framework based advance reservation, which utilizesWSRF/GSI (Web Services Resource Framework/Grid Security Infrastructure). GridARS can co-allocate both computing and network resources. One of the main aspects of GridARS is its Two-Phase Commit (2PC) Protocol based on polling (Takefusa et al., 2007). The framework was evaluated on top of Globus by co-allocating computing and network resources from 7 sites in Japan and 3 sites in US. For the resources in US, they used a wrapper on top of the Highly-Available Robust Co-allocator (Maclaren et al., 2006).
- **JSS:** Job Submission Service (JSS) is a tool for resource brokering designed with the focus on software component interoperability (Elmroth & Tordsson, 2009). This tool has been used in NorduGrid and Swegrid. JSS relies on advance reservations for resource co-allocation. These advance reservations are flexible, i.e. users can provide a starting time interval for the allocation. JSS also considers time prediction for file staging when ranking resources to schedule user applications. JSS does not access the resource providers' scheduling queues to decide where to place the advance reservations. Thus the co-allocation is based on a set of interactions between the metascheduler and resource providers until the co-allocation can be accomplished.

TRENDS

Resource co-allocation is one of the main requirements in a Grid environment to enable cross-site executions. Due to the demand for Quality-of-service, several researchers have been relying on advance reservations for resource co-allocation. Therefore, we believe that most of the future work on resource co-allocation will continue to follow this approach as well. In addition, we have seen more researchers working on *negotiation mechanisms* for co-allocation requests in order to better satisfy users' demand and resource provider requirements (Sim, 2007; Elmroth & Tordsson, 2009; Czajkowski et al., 2002; Li & Yahyapour, 2006). Negotiation is an important mechanism to avoid providers disclosing private information, such as load and resource capabilities, to the metascheduler. The use of resource offers, rather than the resource providers' scheduling queues, will also become more common, especially due to utility computing paradigm (Netto & Buyya, 2009).

Another important trend is the development of *rescheduling policies* for co-allocation requests. As users cannot predict the runtime of their applications, the scheduler has to reschedule them frequently. Other reasons for rescheduling applications are resource failures, dynamic resource demand, and optimization of metrics such as system utilization, power consumption, and users' response time. Rescheduling has also impact on management of contracts, also called Service Level Agreements, in utility computing environments. In these environments, users pay to access resources or services, and providers have to guarantee the delivery of these services with a pre-established Quality-of-Service level. For co-allocation users, several entities may participate in these contracts and hence managing issues such as violation becomes a complex task. Thus, for the next years, especially due to the increasing number of utility computing centers around the world, researchers will be facing the challenge of developing and improving existing policies for managing contracts involving multiple entities

Virtualization is another concept that will be highly explored to provide transparency to users when co-allocating multiple resources that are hosted in either a single or multiple administrative domains. Virtual clusters can be dynamically formed to deploy applications with various application requirements (Chase, Irwin, Grit, Moore, & Sprenkle, 2003). Moreover, with the consolidation of Cloud Computing, resource/service provisioning centers can avoid contract violations and increase system utilization by co-allocation resources from multiple parties on demand.

CONCLUSION

Co-allocation is the process of allocating resources from multiple administrative domains in a coordinated manner. We have shown in this chapter the main research efforts in the area of resource co-allocation. These efforts involve four research directions: distributed transactions, fault tolerance, evaluation of inter-site network overhead, and schedule optimization. We have presented existing work for each of these research directions. We have also described and compared six systems that support resource co-allocation.

From our survey on resource co-allocation, we observed that most of the work do not take into account the distributed transactions because they perform experiments by means of simulation. However, when implementing real systems to deploy in production environments, the support for managing distributed transactions properly becomes an important issue. In terms of fault tolerance, co-allocation systems have been supporting the notion of optional and alternative resources that allows the scheduler to remap the application to other resources in case of failures. As for wide-area network overhead, we noticed that most of the works that consider it have used 25% of the execution time as the threshold to perform experiments. In addition, researchers have been considering location of data and computing resources to schedule multi-site applications. This is particularly necessary when scheduling data-intensive applications.

Most of the work on resource co-allocation focus on scheduling strategies for multi-site jobs. Scheduling strategies have been developed mainly with the use of advance reservations. When scheduling jobs with co-allocation requirements, there are several factors to take into account. Apart from the network overhead and fault tolerance aspects, the scheduling relies on the amount of information available for finding a placement for jobs. The use of a global queue or autonomous queues has a considerable influence on the scheduling strategies. Fortunately, several researchers have been considering the use of autonomous queues to perform their experiments, which is a fundamental characteristic of a Grid Computing environment.

Although there are several researchers working on scheduling policies for co-allocation requests, we have observed that most groups that developed middleware systems with co-allocation support use simple scheduling techniques. That is because there are still several technical details before more advanced scheduling policies can be deployed. Some of these technical problems are interoperability between metaschedulers and resource providers' middleware, inter-site network overhead, and the autonomous policies of each resource provider.

ACKNOWLEDGMENT

We would like to thank Chee Shin Yeo and the anonymous reviewers for their comments on the chapter. This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research (DIISR).

REFERENCES

Alhusaini, A. H., Prasanna, V. K., & Raghavendra, C. S. (2000). A framework for mapping with resource co-allocation in heterogeneous computing systems. In C. Raghavendra (Ed.), *Heterogeneous Computing Workshop* (pp. 273-286). Los Alamitos, CA: IEEE Computer Society.

Alhusaini, A. H., Raghavendra, C. S., & Prasanna, V. K. (2001). Run-time adaptation for grid environments. In B. Werner (Ed.), *International Parallel and Distributed Processing Symposium* (pp. 864-874). Los Alamitos, CA: IEEE Computer Society.

Aumage, O., & Mercier, G. (2003). MPICH/MADIII: A cluster of clusters enabled MPI implementation. In F. Titsworth & D. Azada (Eds.), *International Symposium on Cluster Computing and the Grid* (pp. 26). Los Alamitos, California: IEEE Computer Society.

Azougagh, D., Yu, J.-L., Kim, J.-S., & Maeng, S. R. (2005). Resource co-allocation: A complementary technique that enhances performance in grid computing environment. In L. Barolli (Ed.), *International Conference on Parallel and Distributed Systems* (Vol. 1, pp. 36-42). Los Alamitos, CA: IEEE Computer Society.

Azzedin, F., Maheswaran, M., & Arnason, N. (2004). A synchronous co-allocation mechanism for grid computing systems. *Cluster Computing*, *7*(1), 39–49. doi:10.1023/B:CLUS.0000003942.73875.29

Bernstein, P. A., & Goodman, N. (1981). Concurrency control in distributed database systems. *ACM Computing Surveys*, *13*(2), 185–221. doi:10.1145/356842.356846

Bucur, A. I. D., & Epema, D. H. J. (2003a). The performance of processor co-allocation in multicluster systems. In F. Titsworth & D. Azada (Eds.), *International Symposium on Cluster Computing and the Grid* (pp. 302-309). Los Alamitos, CA: IEEE Computer Society.

Bucur, A. I. D., & Epema, D. H. J. (2003b). Priorities among multiple queues for processor co-allocation in multicluster system. In R. Bilof (Ed.), *Annual Simulation Symposium* (pp. 15-27). Los Alamitos, CA IEEE Computer Society.

Bucur, A. I. D., & Epema, D. H. J. (2007). Scheduling policies for processor coallocation in multicluster systems. *IEEE Transactions on Parallel and Distributed Systems*, *18*(7), 958–972. doi:10.1109/TPDS.2007.1036

Capit, N., Costa, G. D., Georgiou, Y., Huard, G., Martin, C., Mounie, G., et al. (2005). A batch scheduler with high level components. In *International Symposium on Cluster Computing and the Grid* (pp. 776-783). Los Alamitos, CA: IEEE Computer Society.

Cappello, F., Caron, E., Dayd'e, M. J., & Desprez, F. J'egou, Y., Primet, P. V.-B., et al. (2005). Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In *International conference on grid computing* (p. 99-106). Los Alamitos, CA: IEEE.

Chase, J. S., Irwin, D. E., Grit, L. E., Moore, J. D., & Sprenkle, S. (2003). Dynamic virtual clusters in a grid site manager. In *International Symposium on High-Performance Distributed Computing* (p. 90-103). Los Alamitos, CA: IEEE Computer Society.

Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., et al. (1998). A resource management architecture for metacomputing systems. In D. G. Feitelson & L. Rudolph (Eds.), *International Workshop on Job Scheduling Strategies for Parallel Processing* (Vol. 1459, pp. 62-82). Berlin: Springer.

Czajkowski, K., Foster, I., & Kesselman, C. (1999). Resource co-allocation in computational grids. In *International Symposium on High Performance Distributed Computing* (pp. 219-228). Los Alamitos, CA: IEEE Computer Society.

Czajkowski, K., Foster, I. T., Kesselman, C., Sander, V., & Tuecke, S. (2002). SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In D. G. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *8th international workshop job scheduling strategies for parallel processing* (Vol. 2537, p. 153-183). Berlin: Springer.

Decker, J., & Schneider, J. (2007). Heuristic scheduling of grid workflows supporting co-allocation and advance reservation. In B. Schulze, R. Buyya, P. Navaux, W. Cirne, & V. Rebello (Eds.), *International Symposium on Cluster Computing and the Grid* (pp. 335-342). Los Alamitos, CA: IEEE Computer Society.

Dong, S., Karniadakis, G. E., & Karonis, N. T. (2005). Cross-site computations on the TeraGrid. *Computing in Science & Engineering*, *7*(5), 14–23. doi:10.1109/MCSE.2005.92

Elmroth, E., & Tordsson, J. (2007). *A standards-based grid resource brokering service supporting advance reservations, coallocation and cross-grid interoperability*. Manuscript submitted for publication.

Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., & Streit, A. (2002). On advantages of grid computing for parallel job scheduling. In *International Symposium on Cluster Computing and the Grid* (pp. 39). Los Alamitos, CA: IEEE Computer Society.

Ernemann, C., Hamscher, V., Streit, A., & Yahyapour, R. (2002). Enhanced algorithms for multisite scheduling. In M. Parashar (Ed.), *International Workshop on Grid Computing* (Vol. 2536, pp. 219-231). Berlin: Springer.

Ferrari, D., Gupta, A., & Ventre, G. (1997). Distributed advance reservation of real-time connections. *Multimedia Systems*, *5*(3), 187–198. doi:10.1007/s005300050055

Foster, I., Kesselman, C., Lee, C., Lindell, B., Nahrstedt, K., & Roy, A. (1999). A distributed resource management architecture that supports advance reservations and co-allocation. In *International Workshop on Quality of Service* (pp. 27-36). Piscataway, NJ: IEEE Computer Society.

Gray, J., & Lamport, L. (2006). Consensus on transaction commit. *ACM Transactions on Database Systems*, *31*(1), 133–160. doi:10.1145/1132863.1132867

Haji, M. H., Gourlay, I., Djemame, K., & Dew, P. M. (2005). A SNAP-based community resource broker using a three-phase commit protocol: A performance study. *The Computer Journal*, *48*(3), 333–346. doi:10.1093/comjnl/bxh088

Jardine, J., Snell, Q., & Clement, M. J. (2001). Livelock avoidance for meta-schedulers. In A. D. Williams (Ed.), *International Symposium on High Performance Distributed Computing* (pp. 141-146). Los Alamitos, CA: IEEE Computer Society.

Jones, W. M., III. W. B. L., & Shrivastava, N. (2006). The impact of information availability and workload characteristics on the performance of job co-allocation in multi-clusters. In *International Conference on Parallel and Distributed Systems* (pp. 123-134). Los Alamitos, CA: IEEE Computer Society.

Karonis, N. T., Toonen, B. R., & Foster, I. T. (2003). MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, *63*(5), 551–563. doi:10.1016/S0743-7315(03)00002-9

Kuo, D., & Mckeown, M. (2005). Advance reservation and co-allocation protocol for grid computing. In H. Stockinger, R. Buyya, & R. Perrott (Eds.), *International Conference on e-Science and Grid Technologies* (pp. 164-171). Los Alamitos, CA: IEEE Computer Society.

Li, J., & Yahyapour, R. (2006). Negotiation model supporting co-allocation for grid scheduling. In D. Gannon, R. M. Badia, & R. Buyya (Eds.), *International Conference on Grid Computing* (pp. 254-261). Los Alamitos, CA: IEEE Computer Society.

Maclaren, J., Keown, M. M., & Pickles, S. (2006). Co-allocation, fault tolerance and grid computing. In S. J. Cox (Ed.), *UK e-Science All Hands Meeting* (pp. 155-162). New York: NeSC Press.

Mohamed, H. H., & Epema, D. H. J. (2004). An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *International Conference on Cluster Computing* (pp. 287-298). Los Alamitos, CA: IEEE Computer Society.

Mohamed, H. H., & Epema, D. H. J. (2005). Experiences with the koala co-allocating scheduler in multiclusters. In *International Symposium on Cluster Computing and the Grid* (pp. 784-791). Los Alamitos, CA: IEEE Computer Society.

Mohamed, H. H., & Epema, D. H. J. (2008). KOALA: a co-allocating grid scheduler. *Concurrency and Computation*, *20*(16), 1851–1876. doi:10.1002/cpe.1268

Mu'alem, A. W., & Feitelson, D. G. (2001). Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, *12*(6), 529–543. doi:10.1109/71.932708

Netto, M. A. S., & Buyya, R. (2008). Rescheduling co-allocation requests based on flexible advance reservations and processor remapping. In *International Conference on Grid Computing* (p. 144-151). Los Alamitos, CA: IEEE Computer Society.

Netto, M. A. S., & Buyya, R. (2009). Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems. In *International Heterogeneity in Computing Workshop, in conjunction with the 23rd IEEE International Parallel and Distributed Processing Symposium.* Los Alamitos, CA: IEEE Computer Society.

Pant, A., & Jafri, H. (2004). Communicating efficiently on cluster based grids with MPICH-VMI. In *International Conference on Cluster Computing* (pp. 23-33). Los Alamitos, CA: IEEE Computer Society.

Park, J. (2004). A deadlock and livelock free protocol for decentralized internet resource coallocation. *IEEE Transactions on Systems, Man, and Cybernetics . Part A*, *34*(1), 123–131.

Roblitz, T., & Reinefeld, A. (2005). Co-reservation with the concept of virtual resources. In *International Symposium on Cluster Computing and the Grid* (pp. 398-406). Los Alamitos, CA: IEEE Computer Society.

Sim, K. M. (2007). Relaxed-criteria G-negotiation for grid resource co-allocation. *ACM SIGecom Exchanges*, *6*(2), 37–46. doi:10.1145/1228621.1228625

Sinaga, J. M. P., Mohamed, H. H., & Epema, D. H. J. (2004). A dynamic co-allocation service in multicluster systems. In D. G. Feitelson, L. Rudolph, & U. Schwiegelshohn (Eds.), *International Workshop Job Scheduling Strategies for Parallel Processing* (Vol. 3277, p. 194-209). New York: Springer.

Snell, Q., Clement, M. J., Jackson, D. B., & Gregory, C. (2000). The performance impact of advance reservation meta-scheduling. In D. G. Feitelson & L. Rudolph (Eds.), *International Workshop on Job Scheduling Strategies for Parallel Processing* (Vol. 1911, pp. 137-153). Berlin: Springer.

Sonmez, O., Mohamed, H., & Epema, D. (2006). Communication-aware job placement policies for the koala grid scheduler. In *International Conference on e-Science and Grid Computing* (p. 79). Los Alamitos, CA: IEEE Computer Science.

Takefusa, A., Hayashi, M., Nagatsu, N., Nakada, H., Kudoh, T., & Miyamoto, T. (2006). G-lambda: Coordination of a grid scheduler and lambda path service over GMPLS. *Future Generation Computer Systems*, *22*(8), 868–875. doi:10.1016/j.future.2006.03.005

Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y., & Sekiguchi, S. (2007). GridARS: an advance reservation-based grid co-allocation framework for distributed computing and network resources. In E. Frachtenberg & U. Schwiegelshohn (Eds.), *International Workshop on Job Scheduling Strategies for Parallel Processing*. Berlin: Springer.

Takemiya, H., Tanaka, Y., Sekiguchi, S., Ogata, S., Kalia, R. K., Nakano, A., et al. (2006). Sustainable adaptive grid supercomputing: multiscale simulation of semiconductor processing across the pacific. In *Conference on High Performance Networking and Computing* (p. 106). New York: ACM Press.

Vazhkudai, S. (2003). Enabling the co-allocation of grid data transfers. In B. Werner (Ed.), *International Workshop on Grid Computing* (pp. 44-51). Los Alamitos, CA: IEEE Computer Society.

Xu, D., Nahrstedt, K., & Wichadakul, D. (2001). QoS and contention-aware multi-resource reservation. *Cluster Computing*, *4*(2), 95–107. doi:10.1023/A:1011408729750

Yang, C.-T., Yang, I.-H., Wang, S.-Y., Hsu, C.-H., & Li, K.-C. (2007). A recursively-adjusting co-allocation scheme with cyber-transformer in data grids. *Future Generation Computer Systems*.[INSERT FIGURE 001]Retrieved January 21, 2007, from http://dx.doi.org/doi:10.1016/j.future.2006.11.005.

Yoshimoto, K., Kovatch, P. A., & Andrews, P. (2005). Co-scheduling with user-settable reservations. In D. G. Feitelson, E. Frachtenberg, L. Rudolph, & U. Schwiegelshohn (Eds.), *International Workshop on Job Scheduling Strategies for Parallel Processing* (Vol. 3834, pp. 146-156). Berlin: Springer.

Yu, J., & Buyya, R. (2005). A taxonomy of workflow management systems for Grid computing. *Journal of Grid Computing*, *3*(3-4), 171–200. doi:10.1007/s10723-005-9010-8

KEY TERMS AND DEFINITIONS

Advance Reservation: The process of booking resources in advance for future utilization.

Inter-Process Communication: Message exchange among processes of an application.

Livelock: Situation where multiple processes keep trying to access multiple resources but none of them is able to.

Multi-Site Applications: Applications that are executed on multiple sites, which can be from the same or different administrative domains.

Parallel Application: An application comprising multiple processes that can be executed in multiple processing units.

Resource Co-Allocation: The process of allocating resources from multiple providers. It is usually referred to simultaneous access of multiple resources.

Resource Fragmentation: An idle portion of resources in a scheduling queue that cannot be used depending on the scheduler's policies.

Scheduling: The process of defining where and when a task is executed.

ENDNOTE

[1] Grid'5000 had approximately 6000 co-allocation requests in 2.5 years, i.e. an average of 200 requests per month. Based on data collected from the Grid Workloads Archive: http://gwa.ewi.tudelft.nl/pmwiki