Grid Capacity Planning with Negotiation-based Advance Reservation for Optimized QoS *

Mumtaz Siddiqui† **Alex Villazón** **Thomas Fahringer**

Institute of Computer Science, University of Innsbruck
Technikerstraße 21A/2, 6020 Innsbruck, Austria
{ mumtaz | avt | tf }@dps.uibk.ac.at

-

To Appear in the ACM/IEEE Super Computing (SC|06)
November 11-17 2006, Tampa, Florida, USA

Abstract

Advance reservation of Grid resources can play a key role in enabling Grid middleware to deliver on-demand resource provision with significantly improved Quality-of-Service (QoS). However, in the Grid, advance reservation has been largely ignored due to the dynamic Grid behavior, under-utilization concerns, multi-constrained applications, and lack of support for agreement enforcement. These issues force the Grid middleware to make resource allocations at runtime with reduced QoS. To remedy these, we introduce a new, 3-layered negotiation protocol for advance reservation of the Grid resources. We model resource allocation as an on-line strip packing problem and introduce a new mechanism that optimizes resource utilization and QoS constraints while generating the contention-free solutions. The mechanism supports open reservations to deal with the dynamic Grid and provides a practical solution for agreement enforcement. We have implemented a prototype and performed experiments to demonstrate the effectiveness of our approach.

Keywords: Grid Resource Allocation, Negotiation, Advance Reservation, Capacity Planning

1 Introduction

Grid resource management plays a fundamental role in making Grid infrastructure reliable and pervasive. It has to make the resources available on-demand while dealing with their heterogeneity, dynamic behavior, and belonging to different trust domains. As the resources are controlled and administered locally on a Grid site, automatic management of the Grid becomes non-trivial and requires a complex middleware infrastructure. The problem becomes harder and more challenging with the rapid growth of Grid resources and applications. Applications have to compete for the resources while dealing with the middleware complexities, and resources have to distribute their capacity among competing applications based on some multi-criteria QoS parameters (e.g. cost, capacity, timeframe, utilization, and fairness).

This challenge leads to the requirement of a robust middleware with a sophisticated capacity planning for optimized resource allocation. This optimization can be improved significantly with advance reservation so that a Grid Resource Management System (GRMS) can ensure that a certain resource capability will be available at some time in the future. However, advance reservation in the Grid infrastructure has been mostly ignored due to dynamic Grid behavior, concerns about under utilization of resources, applications with multiple constraints, and lack of support in the environment for agreement enforcement. These problems force a GRMS to employ runtime solutions with limited view of the overall Grid capacity availability along a time horizon. These issues significantly reduce the utility of the Grid as well as its applications compared to allocations made properly with a planning horizon.

To overcome this situation, we introduce a mechanism of Grid capacity planning for optimized QoS with negotiation-based advance reservation of Grid resources. An advance reservation, i.e. a limited and restricted delegation of a particular resource capability over a certain timeframe, is obtained for an application from the GRMS, on behalf of resource provider through a negotiation process. We propose a 3-layered cooperative negotiation protocol that is used to efficiently reach an acceptable agreement. Furthermore, in order to deal with dynamic nature of the Grid, we introduce a priority provisioning mechanism, in which the reser-

*The work described in this paper is partially supported by the Higher Education Commission (HEC) of Pakistan under the doctoral fellowship program for Austria and partially supported by European Union through the FP6-IST-004617 project ASG.

†Corresponding author!

SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 © 2006 IEEE

vation system makes a promise that a certain capability will be available in the future. The decision of the actual node allocation however is done or exposed later on, just before resource acquisition. The resources, no matter where they reside or who owns them, are automatically allocated on-demand in order to maximize the global utility. In this way, the process of resource (co)allocation is automatized and the complexity of the Grid is shielded from clients through sophisticated middleware services.

The negotiation for resource allocation is initiated by a client and continues with the generation of allocation offers for individual nodes, followed by co-allocation offers for multiple nodes. Contentions, if any, are eliminated at the third layer of negotiation protocol. At each layer, a set of possible options are proposed based on different QoS parameters. The first layer deals with allocation of a single Grid node. We model it as an on-line strip packing problem [Csirik and Woeginger 1997] and introduce a new algorithm to solve it. The second layer deals with co-allocation of multiple Grid nodes. It receives a set of allocation offers generated by the first layer for a set of available nodes and then generates a set of co-allocation offers with optimized global utility. We frame co-allocation as Constraint Satisfaction Problem (CSP) [Shang and Wah 1998] and employ a new approach to solve it.

We have implemented a prototype of the proposed system as a set of cooperative Grid services using Globus Toolkit 4 (GT4) [Globus], that is a reference implementation of WS-Resource Framework [WSRF], and introduced a practical solution for agreement enforcement using the state-of-the-art Grid technologies. We demonstrate through experiments that our system adapts to more allocation requests and ensures maximum resource utilization with better capacity planning.

The rest of this paper is as follows: Section 2 introduces the resource allocation problem along with a description of utility functions. Section 3 describes a 3-layered negotiation protocol for advance reservation. In Section 4, we discuss capacity planning strategies, followed by Section 5 with architecture and implementation details. Experiment results are depicted in Section 6 followed by the description of the related work in Section 7. Finally we conclude the paper in Section 8.

2 Grid Resource Allocation

The Grid resource allocation corresponds to on-demand provision of Grid resources to Grid applications. Generally, a Grid application consists of multiple activities (software components) which run on multiple Grid nodes in a well-defined order during a specific timeframe [Cooper et al. 2004; Yu and Buyya 2005; Fahringer et al. 2005]. The task of resource allocation involves resource selection, i.e. discovering and matching nodes to each activity, finding allocatable time intervals on each node during which an activity can execute, and finally making a combination of all available time intervals on all nodes. In this way, the Grid application can run successfully with optimized QoS parameters.

The resource selection mechanism is already available as part of our previous work called GridARM [Siddiqui et al. 2005b; Siddiqui and Fahringer 2005], that is, Askalons's GRMS [Fahringer et al. ; Fahringer et al. 2005], and it covers both physical and logical resources. New work is an extension of GridARM and consists of two main components: *allocator* that makes reservations of a single node, and *co-allocator* that makes reservations of multiple nodes for a single Grid application[1].

A co-allocator accepts requests from the clients and generates alternative co-allocation offers, which can be used by the client to execute its application. A co-allocation request may consist of a set of allocation requests for each Grid activity along with a set of constraints. The negotiation starts either on a client's request or when the state of the Grid is changed (some resources leave or join the Grid). The co-allocator negotiates with clients as a resource trader and with allocators as a cooperative negotiation mediator. Cooperative negotiation enables the system to generate offers closer to clients requirements so that negotiation interactions with client are minimized, resource utilization and other QoS constraints are optimized, and inter/intra-application contentions are eliminated. Resource contention is introduced when the same slot is simultaneously offered for multiple Grid activities. Coallocators try to agree over sharing of scarce resources without loosing their clients. This introduces the need for a cooperative negotiation mechanism, i.e. internal negotiation between components of the reservation system.

2.1 Problem Description

Grid resource allocation is a hard problem due to contention, let alone dynamic Grid behavior. Generally speaking, a resource allocation problem is the problem of assigning a limited set of Grid nodes S, each with a scarce capacity (e.g. number of processors P_s) to a set of co-allocation requests R (for an application α) by a set of clients C. Each allocation request $r \in R$ requires a certain capacity $P_r \in P_s$ for a specific time interval (duration) $T(r) = endt(r) - startt(r)$, and may have the potential for varying utility $U(r)$ depending on application constraints. Here we refer to r_i as a single allocation request for a specific Grid activity, which may have some dependencies on other requests in a co-allocation. Figure 1 depicts the format of an allocation request that consists of a set of activity and node constraints, and of a context referring to the participants. The right side of Figure 1 shows the format of a constraint that includes $name/value$ of a constraint and $flexibility \in \{0..10\}$ that represents client's willingness for negotiation.

[1]Here we use *allocator* and *co-allocator* terms which refers to node-level and Grid-level reservation managers respectively, and the term *node* refers to a Grid site

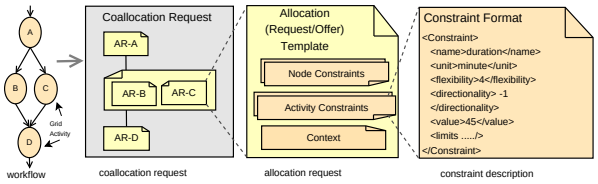

Figure 1: A workflow to co-allocation request mapping along with an allocation and a constraint format.

The goal is to maximize the global utility U, choosing the right options for applications, and achieving an optimal compromise over constraints of the stakeholders. More formally, the Grid resource allocation problem comprises of

- A set of Grid applications $\alpha = \{\alpha_1, ..., \alpha_n\} \mid n \in \mathbb{N}$.
- A set of co-allocation requests $\rho = \{R_1, ..., R_n\}$ where $R_i = \{r_{i,1}, ..., r_{i,j}\}$, and r is the allocation request. Note that each co-allocation request R_i corresponds to a Grid application and consists of one or more allocation requests and may come from different clients.
- A set of Grid nodes $S = \bigcup_{i=1}^{N} s_{i,k}$ where N is number of nodes and $k \in time$ is planning horizon. Each node $s_{i,k}$ possess some capacity, for instance number of processors P_s. The total Grid capacity in terms of processors would be $P = \sum_{s=1}^{N} P_s$.
- A set of utility functions, $U = \{U_1, ..., U_n \mid U_i : 2^{|S|+k} \rightarrow \mathbb{R}\}$, each associated with an application.

The goal is to come up with a set of co-allocation offers

$$\Omega : \Omega = \{\omega_1, ..., \omega_n \mid \omega_i \in 2^{|S|+k}\}$$

such that $\sum_{i=1}^{n} U(\omega_i)$ is maximized and

$$\bigcap_{i=1}^{n} \omega_i = \varnothing.$$

We used $2^{|S|+k}$ to indicate the power-set of the available allocation options (maximum allocations possible on resource and time horizon ($|S|+k$)). Because the resource requirements may change over time, or a particular pattern of resource usage may be needed to obtain utility, allocation options are reduced on both the resource and time dimensions, hence the need for a planning horizon. Increasing the number of resources or the time horizon can have a significant effect on the overall complexity of the allocation problem, which is *NP-complete* [Shang and Wah 1998; Csirik and Woeginger 1997].

2.2 Utility Functions

The goal of resource allocation is two fold: (1) to obtain enough resource capacity requested for each single application in the requested order, and (2) to maximize resource and application utility. Depending on QoS-constraints, the utility function associated with each application and resource may change. Generally, increasing the resource capacity or making allocations with values of the QoS-parameters closer to the application requirements improves the application utility.

Application utility is a function of distance between requested allocation and real option offered. If $D(c)$ is the distance between the requested and offered value of a constraint c of a single allocation ω, then the application utility is the aggregation of utility of all allocations for the application α_i, i.e.

$$U(\alpha_i) = \sum_{\omega \in \Omega_{\alpha_i}} U(\omega) : U(\omega) = \sum_{c \in Constraints} util(D(c))$$

and $U = \sum_{\alpha_i \in \alpha} U(\alpha_i)$ is the global utility.

An allocation is generated, by using a set of objectives defined by the utility functions, to assign resource capacity to an application. Each function is expressed in terms of application's utility or in terms of resource utility which is the function of its offered QoS, for instance, the capacity, cost, and time.

Distance Formula

The application utility is derived by aggregating the differences between ideal and real values of all QoS constraints. For a specific constraint c, if c_{req} is the required (ideal) value, $c_{offered}$ is the offered (real) value, and c_{flex} is the level of flexibility for negotiation, then distance $D(c)$ for constraint c is calculated as

$$D(c) = \frac{c_{req} - c_{offered}}{c_{flex}}$$

Here $c_{flex} \in \mathbb{N} \mid 0 \leq c_{flex} \leq 10$, where 0 means no flexibility over a given constraint thus making it a *hard constraint* (i.e. has to be fulfilled in order to make an agreement), whereas 10 shows maximum flexibility for negotiation (a *soft constraint*). A negative distance may have different meaning for different constraints, for instance, in case of cost a negative distance shows that the offer is expensive, whereas in case of capacity, a negative distance shows that more capacity is offered than requested which could be acceptable. To deal with these differences, we introduce the notion of *direction of a constraint* $c_\delta \in \{-1, 0, 1\}$ that tells whether or not a negative distance matters. Depending on the direction and flexibility of a constraint, the distance $D(c)$ can be

$$D(c) = \begin{cases} \infty & (c_{flex} = 0) \wedge c_{req} - c_{offered} \neq 0 \\ 0 & c_\delta \neq 0 \wedge (c_\delta * D(c)) < 0 \\ |D(c)| & otherwise \end{cases}$$

The $D(c) = \infty$ leads to a situation where no solution is possible with available options and the capacity planner gives up. A negative distance will become 0 if $c_\delta \neq 0$, and becomes positive otherwise. For example, in the case of cost:

offer	procs	mem	cost	Distance
A	4	5	180	$\frac{4-3}{1}+\frac{5-2}{4}+\frac{180-150}{6}=6.8$
B	2	6	140	$\frac{2-3}{1}+\frac{6-2}{4}+\frac{140-150}{6}=3.9$
C	6	1.5	220	$\frac{6-3}{1}+\frac{1.5-2}{4}+\frac{1.5-2}{6}=15$

Table 1: Ranking of offers using distances.

if $cost_{req}=100$, $cost_{offered}=50$, $cost_{\delta}=-1$ then $D(cost)=\frac{100-50}{cost_{flex}}*-1=\frac{-50}{cost_{flex}}<0$, i.e. a negative distance that becomes 0 because of a cheaper offer which is acceptable.

The distance of an allocation ω is an aggregated distance of all its constraints i.e. $D(\omega)=\sum_{c\in Constraints}D(c)$, whereas $D(\Omega)=\sum_{\omega\in\Omega}D(\omega)$, i.e. distance of a co-allocation Ω. For instance, a client requests a resource with required constraints as $procs=3$, $mem=2GB$ and $cost=150$, with $flex$ 1, 4, and 6 respectively. An allocator generates offers for this request with offered constraints as shown in Table 1. According to the distance formula, the offer *B* has the least distance, therefore it is ranked as the best one.

This formula is appropriate only for numeric values, e.g. cost, number of processors etc. For lexicographical values, typically used for logical resources (e.g. operating system), we propose a different approach that uses a semantics-based hierarchical structure of the all possible values of a constraint *c*. Each level of the hierarchy is associated with a numerical value. If *i* and *j* refer to the levels of ideal value *I* and real value *J* respectively in the hierarchy and $k=root(i,j)$ is the level of their common root, then

$$D(c_{i,j})=\begin{cases}0 & i\leq j\wedge i=k\\ \frac{2^{i+j-2k}}{flex_{c_i}} & otherwise\end{cases}$$

This covers two special cases related to the semantically defined classes *I* and *J*, for ideal and real values respectively:

- $I\sqsubseteq J \mid I\equiv J$: The ideal value belongs to a sub or equivalent concept of the offered value class, with $D(c_{i,j})=0$ i.e. it is an ideal match.
- $I\sqsupseteq J$: The ideal value is a super concept of the offered value and thus it leads to a next satisfiable decision.

```
      OS           (Operating System)    Level 1
     /  \
   Unix  Windows  (Sub classes of OS)    Level 2
  /   \
Linux  Solaris     (Specialized Classes) Level 3
```

For instance, in the hierarchy of operating systems as shown above, the *Linux* to *Unix* distance

$$D(c_{Linux,Unix})=\frac{1*2^1}{flex_c}$$

and the *Linux* to *Windows* distance

$$D(c_{Linux,Windows})=\frac{1*2^3}{flex_c}$$

and for the same value of flexibility, it can be deduced that

$$D(c_{Linux,Windows})>D(c_{Linux,Unix}),$$

i.e. Linux is closer to Unix than to Windows. This makes constraints with lexicographical values, which can be described in a hierarchy of subsumption tree [Siddiqui et al. 2005a], comparable with the constraints having numerical values.

3 Negotiation Protocol

The negotiation process implies multiple interactions between clients (e.g. schedulers) and co-allocators until they reach an agreement. Resources are offered to clients, who can select the best suitable offer or can decide to re-negotiate by changing some of the constraints. The protocol introduces negotiation between a client and a co-allocator as well as between components of the reservation system. The goal is to generate co-allocation offers as optimal as possible so that interaction between the requester and provider is minimized and resource utilization is maximized. The co-allocator accepts requests in the form of WS-Agreement templates and generates a set of co-allocation offers in the WS-Agreement format. The WS-Agreement is the proposed standard for the Grid Resource Allocation Agreement Protocol [GRAAP-wg 2006].

As the problem is *NP-Complete* we use different heuristics at different levels. As such, the elegance of the protocol is actually distributed in three layers: allocation, co-allocation, and coordination. Figure 2 shows the negotiation layers along with possible negotiators.

The *allocation layer*, driven by the *allocators*, deals with reservations of individual Grid nodes. The main objective of this layer is to perform resource-level capacity planning in order to optimize the utility. At this stage, offers are available, albeit not necessarily optimal and/or conflict-free. The *co-allocation layer* moderated by the *co-allocator* takes the client's preferences into account and generates co-allocations while optimizing the global utility. Therefore, the second layer improves the quality of the generated offers in a broader sense. If there is an intra-application contention, i.e. contention between allocations generated for activities of same application, is eliminated.

It is possible that an allocation made at the second layer is optimal and contention-free for an application, but, as resources are shared, the same option can be offered to multiple applications. This introduces the inter-application contentions that demands the coordination among co-allocators so that inter-application contentions can be eliminated. Such contentions are propagated to and handled by the third layer, which can be activated by the clients and also by the system when resources join or leave the Grid. This layer also ensures that open reservations will remain internally bound to some nodes which satisfy required QoS. Open reservations are flexible allocations whose binding with nodes can be changed before application runtime. This allows a co-allocator to accommodate newly joined resources optimally and to perform some sort of *overbooking*, depending on the

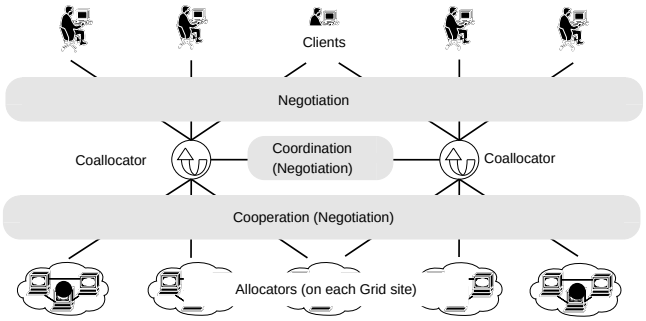

Figure 2: Cooperative Negotiation Protocol Layers.

history-based prediction for the possibility of new resources who will join in the future.

3.1 Allocation

The allocation layer deals with the capacity planning of a single node in which allocators work as capacity planners and considers only a local view of the respective resource capacity. Formally, an allocation request r_i is forwarded by the co-allocator to its underlying allocators, which then generate a set of allocation offers A_i and return the set ordered by client utility. The allocator or resource utility may depend on the providers strategy. If k is the total number of offers, then:

$$A_i = \{a_1, ..., a_k \mid U_i(a_i) > U_j(a_j), i > j\}$$

An allocation a_i is modeled as rectangle given by its width (p_i) and height (d_i) corresponding to the processing capacity and the duration. The allocation start time is $startt_i$, the requested execution time is d_i, and the deadline is $endt_i$, with $endt_i \geq startt_i + d_i$. The capacity of an entire node is also modeled as a rectangle but with fixed width P and infinite height (time horizon). The allocator tries to locate a set of available slots so that the hard constraints (with $flexibility = 0$) are met and distance of soft constraints is minimized, such that,

- $(startt_i + d_i) \leq startt_j \vee (startt_j + d_j) \leq startt_i$
- $\forall A_j \in A : A_j \neq A_i$, i.e. allocations must not overlap.
- $D(\omega) \neq \infty$ and should be minimum.

The allocation problem shares similarities with strip packing problem. In the strip packing problem we try to place a set of two dimensional boxes into a vertical strip of width W and height ∞ while keeping the total packed height of the strip minimum. Translated to our allocation problem, the width of the strip corresponds to the resource capacity, for instance, number of processors, and the vertical dimension corresponds to time horizon. If the list of rectangles is unknown in advance, the strip packing problem is called an *online strip packing*, which is *NP-hard* [Csirik and Woeginger 1997], and exactly maps to our problem.

The simplest on-line method is to check whether a newly requested allocation finds an immediate placement. If there is none, the request is rejected. This is a very simple but crude technique that needs to know only about the current view and shows a low resource utilization resulting in the wastage of the strip capacity. A sophisticated on-line method increases the acceptance ratio by planning, i.e. looking into the future according to the client's flexibility for negotiation.

On-line strip packing is addressed by different heuristics such as shelf algorithm [Csirik and Woeginger 1997]. We introduce a new algorithm called Vertical Split and Horizontal Shelf-Hanger (VSHSH), which provides a hybrid approach and fits better to our allocation problem. In contrast to classic shelf algorithm, in which the strip is horizontally split into shelves and only bottom-left justified packing is possible, the VSHSH allows top-right justified packing as well. In this way, the VSHSH keeps unused area of the strip minimum while increasing application utility. As the goal is to provide allocations as close to the requested QoS as possible, we propose top or bottom justification for an allocation depending on its distance from the requested timeframe. This approach increases the global utility (more satisfied QoS constraints) as probability of wider time-constraint distance is reduced (see Section 6 for further discussion and measurements). This is logical because

$$D_{shelf}(startt) = startt_{req} - shelf_{base}$$

whereas

$$D_{vshsh}(startt) = min(startt_{req} - shelf_{base}, shelf_{top} - startt_{req})$$

which clearly shows the higher probability of

$$D_{vshsh} < D_{shelf}.$$

Furthermore, the VSHSH also vertically splits the strip into multiple sub-strips so that $W = \sum w_i$, i.e. the width of all sub-strips is equals to the width of main strip. This splitting is used to protect an appropriate share of the resource capacity for the different communities or Virtual Organizations (VO), and may apply different allocation strategies for each strip, such as different shelf height and cost models. In this way a sub-strip becomes an independent strip. As depicted in Figure 3, a possible capacity management model for the VSHSH consists of three strips with different capacity shares, shelf heights, and cost models.

Figure 4 depicts an example strip visualizing shelf $1, 24, 45$ with allocations in shaded-solid boxes. We assume that all other shelves are fully packed. In this situation, an allocation request arrives for which the VSHSH generates three allocation offers. As the request is closer to the top of the shelf 24 therefore the VSHSH generates a top justified time-constrained offer, i.e. offer 1. The second offer (offer 2) targets the economy class of clients and is generated with least cost but higher distance of other constraints. The third offer (offer 3) is more expensive but gives earliest possible

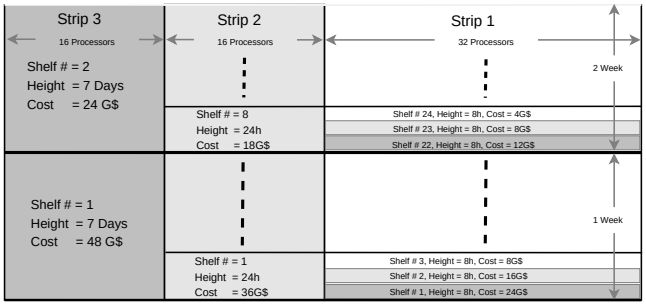

Figure 3: A possible Capacity Management model for VSHSH with three Sub-Strips having different Capacity, Shelf Height, and Cost Models.

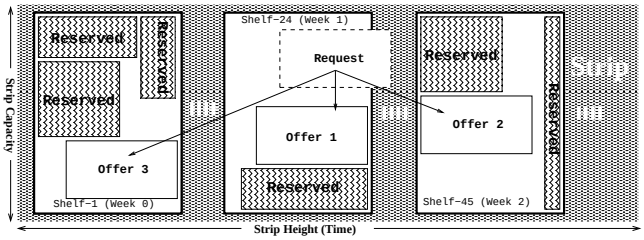

Figure 4: Possible Offers Generated by the VSHSH with a Strip having Free Space in the three Shelves (1,24 & 45), Offers are Generated According to the Client-Classes.

result. The allocator will send all three generated offers and the co-allocator will choose the best offer according to the client's overall QoS requirements.

The VSHSH also supports dynamic scaling of the underlying sub-strips with a variable width and shelf-height scenario. In a variable shelf height version of VSHSH, a new shelf is created if:

- an allocation does not fit to any existing shelf found within the client's flexible range.

  $$level(\psi_i) \leq startt_a \leq level(\psi_j) \ \vee \ \exists A_i \in A(A_i \cap a = a)$$

  In this case a new shelf with

  $$level(\psi_{new}) = max(height(\psi_{last}), startt_a)$$

  is created. Here ψ refers to a shelf.

- an existing shelf can be split into two shelves so that the new allocation can be bottom-justified in upper shelf and there is no overlap with lower shelf.

We also introduce the notion of borrowing space from the adjacent shelves. For instance, if an allocation request with longer duration fits in the multiple adjacent shelves, then it can be honored at higher price, provided that the requested QoS-constraints are fulfilled.

The VSHSH pseudo code is depicted in Algorithm 1. It generates alternative offers according to different constraints such as timeframe, cost, capacity, and client's flexibility set for these constraints. In case of client's flexibility over height (duration) and width (capacity), we propose to change the area of the rectangle according to the *isospeed scalability* $\{i.e. \psi(p,p') = \frac{Time'.P}{Time.P'}\}$ of the system for the requested application component (activity in case of workflow). For example, if requested capacity is not available, then an offer with reduced capacity but increased duration might be acceptable for the client.

Algorithm 1 The Pseudo-Code of the Allocation Offer Generation Algorithm.

negotiate4Allocation()
Input: Request $r = \{p,d,c,st,et,?...\} | p = procs, d = duration, c = cost, st = starttime, et = endtime, ? = anyTerm$ {A request r with a set of constraints}
Output: A i.e. a set of allocation offers
$h := shelfHeight$;
$sbt := stripBaseTime$;
$curr := shelf(r) = (st_r - sbt)/h$;{current shelf}
$o := \oslash$;{offer = null}
for $next := curr$ to $curr + et_{flex}$ step 1 **do**
  {generate bottom justified offer in *next* shelf, if possible.}
  $nst := sbt + next * h$;{start time of offer to be generated}
  if $(o = \{p,d,?,nst,nst+d\} \vee \exists a \in A_{next}(o = \{p,d,?,et_a,et_a+d\})) \wedge A_{next} \cap o = \oslash$ **then**
    $A := A + \{o\}$;{Add time-constrained generated offer in A}
  end if
end for
for $prev := curr$ to $curr - st_{flex}$ step -1 **do**
  {generate top justified offer in *prev* shelf, if possible.}
  $net := sbt + prev * h + h$;{endtime of offer to be generated}
  if $(o = \{p,d,?,net-d,net\} \vee \exists a \in A_{prev}(o = \{p,d,?,st_a-d,st_a\})) \wedge A_{prev} \cap o = \oslash$ **then**
    $A := A + \{o\}$;{Add time-constrained generated offer in A}
  end if
end for
if $isEconomyClient(cost_{flex})$ **then**
  Generate an offer after 2^{nd} *week* or in economy *strip*;
else
  Generate an offer in 1^{st} *week* or in expensive *strip*;
end if
return A;

The contentions among multiple allocations are propagated to the co-allocation layer instead of handling locally by the allocators with a limited view. It could be possible that higher local objective may lower the global utility. There is a tradeoff between allocator's objective and application's utility that is addressed during the contention elimination for an optimal compromise.

3.2 Co-allocation

The co-allocation or the 2^{nd} negotiation layer is responsible for the generation of a set of possible co-allocation offers, i.e. suitable combinations of allocation offers received from the underlying allocators, according to the client's global preferences or the required QoS parameters.

A co-allocator instantiates a *co-allocation manager (CM)* for each request which handles further negotiation between

the client and the underlying allocators, and then performs ongoing monitoring of the accepted co-allocation. A co-allocation request corresponds to a Grid application and a typical Grid application is a workflow that consists of multiple activities [Siddiqui et al. 2005b] which are to be executed in a well defined order. In such a case, each co-allocation request consists of multiple allocation requests, each corresponding to an activity of a workflow application. As depicted in Figure 1, the request then adds another constraint, i.e. to ensure the order of the allocations. The co-allocation layer handles such constraints as well by generating offers with allocations in requested order. Formally, a co-allocator receives a co-allocation request R, where

$$R = \{r_1, ..., r_k\} \mid k \in \mathbb{N}$$

and instantiates a CM which then sends each allocation request $r_i \in R$ to allocators of selected nodes and receives a set of allocation offers A, where

$$A = \bigcup_{i=1}^{k} (A_i) : A_i = \{a_{i,1}, ..., a_{i,m}\} \mid m \in \mathbb{N}$$

Now the CM generates co-allocation offers $\Omega = \{\Omega_1, ..., \Omega_n\}$, such that,

$$\Omega_i = \Psi(A_1, ..., A_k) = \{\omega_1, ..., \omega_k\} \mid \forall i \in k(\omega_i \in A_i)$$

Here Ψ is a set reduction operator whose domain is a set of all possible sets of allocation offers received for each request r_i from the different allocators. We frame a co-allocation as an optimization problem, that is similar to CSP, and propose a new algorithm which is a modified form of an existing min-conflict local search algorithm [Shang and Wah 1998]. In contrast to min-conflict local search algorithm, the new algorithm allows to change the resource objective depending on the number of conflicts, so that, a better compromise could be found with optimal global utility. The co-allocator logically considers each allocation offer as a resource and applies the new solution to these 'resources' in order to make a set of acceptable combinations. This is in contrast to the first layer where resources are actual physical computers. This significantly reduces the complexity of the problem, i.e. from 2^{S+k} to $2^S \equiv 2^A$. The co-allocation offer generation Pseudo code is depicted in Algorithm 2.

A co-allocation manager (CM) returns to the client a set of co-allocation offers Ω ordered by its utility. The client then filters again, by eliminating the offers which are not acceptable and sends back in a preferred order Ω_{pref} for confirmation. The confirmation process is two phase committable. In the first phase, the CM tentatively reserves each allocation offer $\omega \in \Omega_{pref}$, and in the second phase, it confirms each of the reserved allocations provided there is no contention. Otherwise, it propagates the contentions to the third layer and rolls back the tentatively reserved allocation offers.

Algorithm 2 The Pseudo code of the Co-Allocation Offer Generation Algorithm.

```
negotiate4Coallocation()
Input: R : R = {r_1,...,r_n} {Co-allocation request}
Output: Ω a set of co-allocation offers
A {a set of allocation offers to be received}
S {a set of available sites to be discovered for each r}
done := false;
for ∀r ∈ R do
   S := lookup(r);{Call Grid resource broker to select sites}
   {Negotiate with each site by calling negotiate4Allocation algorithm)}
   A_r := ⋃_{∀s∈S} negotiate4Allocation(r);
   A := A ∪ A_r;{A becomes {A_1,...,A_n}}
end for
while !done do
   if Ω^a = {ω_1,...,ω_n} and ∀_{i∈n}(ω_i ∈ A_i) and U(Ω^a) > 0 and
   ⋂_{ω_i∈Ω^a} ω_i = ∅ then
      Ω := Ω + {Ω^a};{Add a co-allocation offer in offer set Ω}
      {exclude the used allocation offers from the A}
      ∀A_i ∈ A do A_i − {A_i ∩ Ω^a}
   else
      done = true;
   end if
end while
return Ω;
```

3.3 Coordination

Contentions are unavoidable in an environment such as the Grid where resources are shared and the clients are the competitors. It is possible that while generating (co)allocation offers for multiple applications, the same slot is offered to more than one client. Such contentions reduce the QoS and lead to unacceptable solutions. The coordination layer deals with such situations and produces contention-free solutions by eliminating either conflicting offers from the solution domain or by lowering the objective level of some of the underlying allocators. The contention problem is raised by the allocator and propagated towards the co-allocator. The notified co-allocator then mediates cooperative negotiation among the contentious CMs.

First, the notified co-allocator collects all information needed to generate alternative solutions, including allocation options offered to the conflicting applications, and then enters in the solution generation process. The solution generation process starts with ordering the solution domain according to application utility. A solution for the highly constrained application is generated first, assuming that this will reduce the possibility of further contentions. In order to accommodate all requests, the mediator may lower the objective levels of the underlying allocators, depending on the number of contentions associated with each offer. In this way, the optimal solutions are generated which satisfy the required QoS constraints. The process terminates: 1) if all suitable contention-free co-allocations are generated, 2) or the objective of any of the allocators cannot be reduced further. This is slightly different from the actual constraints satisfaction problem (CSP) [Yokoo et al. 1998] in which the objective level cannot be changed. In contrast to the clas-

sic CSP, we change the objective level in order to address the same problem but to generate a next suitable solution, mostly with reduced utility which is better than having no solution at all.

Second, a mediator enters in the solution evaluation phase by sending each of the CMs a set of contention-free solutions. The CM then filters out some of the generated solutions and orders the rest of them from best to worst, based on the application's overall utility U_i. Once a mediator has the ordering from the CMs, it generates an overall solution by choosing the highest ranked alternative from each of the CMs which lead to a consistent solution. Again, most constrained applications are given higher priority while assigning a solution.

Finally, the assigned solutions are sent back to each of the CMs, which then implement the final solution by confirming the allocations and sending back the response, in the form of a ticket, to the client. No further contention arises once an allocation is confirmed unless the allocation is an open reservation.

4 Capacity Planning

Capacity management and planning has been addressed in several other fields, e.g. airline yield management, where perishable resources are advertised and sold in a way to maximize overall profit [Netessine and Shumsky(2002)]. As computing power of the Grid can be considered perishable, the capacity management can be performed in a similar way. Grid capacity planning is a forward looking activity of monitoring, understanding, and reacting to the clients' behavior in order to maximize the global utility. The advance reservation of Grid resources cannot be fully exploited without anticipating adequate needs of its clients so that a proper share of the resource capacity can be protected for the clients who can be more profitable in the future.

The VSHSH allocation mechanism for advance reservation, as introduced earlier, splits resource capacity into multiple strips and assigns each strip a different capacity, shelf height, and cost models. We associate each strip to a group of clients, e.g. a Virtual Organization (VO). Determining and associating proper capacity share for each strip is an ongoing process based on monitoring the clients' demands. Associating a static share with each strip is impractical, especially due to dynamic Grid behavior. We extend VSHSH with *dynamically scalable parameters*, such as strip capacity, shelf-height, and cost, depending on the client's requirements by using the history of allocation requests/demands.

In case of a strip with static capacity share, different allocation strategies can be configured for different timeframes. This is useful for planning a specific time interval or events. For instance, it is more likely that during off time or vacations, more nodes join the Grid and less clients come online. In order to avoid wastage of capacity and to attract more clients, we propose overbooking and offering low cost during off-times. The overbooking can be possible with the help of priority provision as described in the next section.

Furthermore, some special events can be considered during which a certain user will be more interested in making a reservation even at a higher price. For instance, a researcher may pay more for the Grid resources just before the submission deadline of a high-profile conference. We can also fairly assign the equal capacity shares to each strip, for example, in the case of processors, $P = \sum_{i \in strips} p_i$. In this way, each user-class gets a fair-share. But this strategy is useful only if the total number of active users in each class is the same, otherwise it reduces the resource and application utility by under utilizing one strip and reducing required QoS of the other strip clients. In such a case, VSHSH with dynamically scalable strips is more practical.

4.1 Priority Provision

Priority provision is an open reservation with soft allocation of resources. In open reservations, actual node binding either can be changed over time or deferred until application runtime. The resources are allocated to an application, but real binding is shielded from the client. In this way, a promise is made that a certain capability will be available at sometime in the future without assigning a specific node to the client. That means that a next available node that fulfills the required QoS constraints is allocated on-demand.

The priority provision is a way of keeping the promise of advance reservation by dynamically associating physical resources with allocations in an underlying unpredictable and dynamic Grid environment. Furthermore, it allows reservation of logical resources as well. This is an important feature, which according to our understanding, has not been considered for advance reservation so far in any other Grid computing infrastructure.

4.2 Cost Handling

In order to deal with cost-centric optimization, we introduce the concept of fictitious money (Grid$) and provide a cost model for different strips or different shelves of a single strip. The clients' account for fictitious money can be maintained by the co-allocator, and the underlying allocators charge the cost of their resources to the client's account. This is similar to the open resource allocation model applied to mobile code [Tschudin 1997], which in contrast maintains a fictitious money based on the lottery scheduling [Waldspurger and Weihl 1994].

Currently, clients are categorized using the value of negotiation $flexibility$ set for the cost constraint. We support three categories of clients: economy, moderate, and wealthy; these are derived from the $cost_{flex}$. Recharging the fictitious money account can be associated with the lifetime of the Grid-user's proxy used to access the resources.

We use the idea of fictitious money for our experiments in order to demonstrate multiple QoS-constraint-based op-

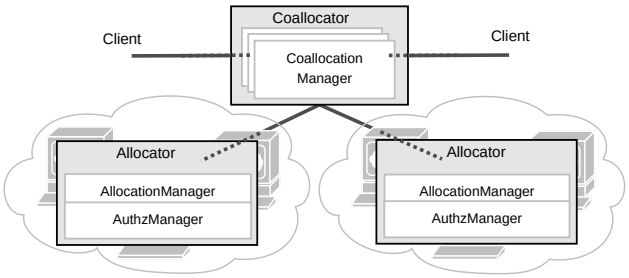

Figure 5: Advance Reservation System Architecture.

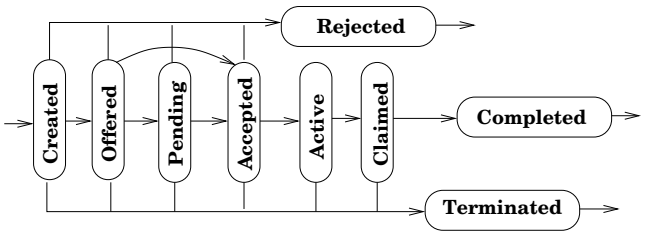

Figure 6: Possible States of a Reservation Instance.

timization of resource allocation, but it can be extended to real money as well. We are exploring mechanisms such as those proposed for GridBank [Barmouta and Buyya 2003] and community authorization service (CAS) [Globus], so that the clients can be charged properly. They can maintain their accounts indirectly through VO administrators with support of CAS, where account balance can be set depending on the user's privileges, VO policy, or actual money. Each time a client confirms a reservation, the cost of the allocated resources can be charged. Similarly, a possibility for the reimbursement, in case of cancellation can also be considered.

5 Implementation

The proposed reservation system consists of resource and Grid level components called *allocator* and *co-allocator*, respectively (see Figure 5). The allocator is responsible for the provision of advance reservation of a single Grid node, whereas co-allocator handles reservations of multiple nodes. Both allocator and co-allocator are implemented as WSRF Grid services, based on the Globus Toolkit 4 (GT4), and have been integrated in the GridARM, a GRMS.

An allocator further consists of two sub components: an *allocation manager (AM)*, that generates offers while optimizing QoS constraints, and an *AuthzManager* that authorizes whether or not a client should be permitted to acquire the reserved resource (e.g. to submit a job). An AM provides a mechanism in which different algorithms can be plugged-in and configured according to the resource usage constraints and provider's strategy.

In order to ensure sophisticated integration of the local resource management with reservation, we have chosen not to change the low level mechanism of the Job Submission service (as proposed in [Elmroth and Tordsson 2005]), but rather exploit the customizability of the GT4 Job submission service called WS-GRAM [Czajkowski et al. 1998; Globus], which allows addition of customized resource authorization policies. The AuthzManager acts as a special Policy Decision Point (PDP) for WS-GRAM and ensures client's authorization, i.e. whether the resource was actually reserved by the client through AM.

The co-allocator covers a set of allocators (each associated with a node) and runs on a superpeer node. A superpeer node is one that works as a frontend or root node for a group of nodes and can reference one or more other superpeer nodes. In this way, if a GridARM service such as a co-allocator cannot find an answer within its own group, then it can refer to the peer services running on the remote superpeer nodes for the answer.

A co-allocator works as a factory service of co-allocation managers (CM) and instantiates a separate CM for each request. The CM then handles further negotiation between a client and underlying allocators, and performs ongoing monitoring of the accepted co-allocation. Furthermore, a co-allocator interacts with a GridARM resource broker for candidate selection and may filter out nodes for which hard constraints (i.e. requests with $flexibility = 0$) cannot be fulfilled. It also mediates the contention elimination process.

A reservation can pass through different states during its lifecycle from negotiation to job execution and can be monitored and managed by using standard WSRF constructs (as each reservation instance is available as WS-Resource in the form of WS-Agreement). Figure 6 shows the state sequence of a reservation instance. A client can register for the event notification. Each time a reservation changes from one state to another, a registered client receives a state change notification. For example, a client can start re-negotiation with a TERMINATED state notification or submit a job to the resource with an ACTIVE state notification.

After making a successful reservation, the clients can submit their jobs for execution within the reservation timeframe. By default, only a client with valid credentials and having a valid reservation can submit a job. Moreover, a CM also generates a *ticket* which can be delegated to other clients who can acquire associated reservations by presenting the ticket. A client will not be authorized if it fails to provide a valid *ticket* or fails to prove itself as an owner of the reservation.

5.1 Agreement Enforcement

A special PDP (Policy Decision Point) for the authorization of reserved resources is an essential component of our reservation system. It exploits the customizability of the WS-GRAM in which multiple authorization points (PDPs) can be configured. We introduce an additional *ReservationPDP* for the authorization of clients, it works as part of the chained authorization points invoked by WS-GRAM. As depicted in Figure 7, this additional PDP interacts with the AuthzMan-

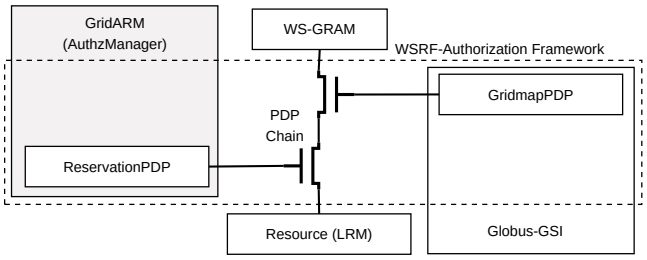

Figure 7: A Policy Decision Point (PDP)-chain with a Special ReservationPDP for WS-GRAM.

ager and gets verification whether or not the client has advance reservation at that particular time. The ReservationPDP is used for the enforcement of an agreement.

AuthzManager may be configured with a customized policy, for instance, executions can be performed only when the resource is reserved by the user, or if the resource is not reserved at all during the requested timeframe. If an application does not finish within the reserved timeframe, the tasks should be either terminated or suspended. Termination of an application which is about to complete could be counter productive, especially when application execution time is longer. On the other hand, suspension of an application requires low-level system interaction, which is an open research topic.

A reservation made by an allocator is independent from the underlying Local Resource Manager (LRM), e.g. PBS [Altair], LSF [Platform], SGE [Sun Microsystems]. This means that a job can be submitted through WS-GRAM to any of the LRM and is honored only if proper reservation is made by the client. Each time WS-GRAM is invoked, it interacts with AuthzManager through the special Reservation PDP for verification of the caller.

In some cases (depending on the Grid node policy and configuration), it may be possible that a client bypasses WS-GRAM and submits jobs directly to any of the deployed LRMs. In such a case the reservation will not be verified. This is, in fact, a well-known open issue. One possible enhancement is to provide low-level reservation with the help of LRMs (e.g. based on Maui [Jackson et al. 2001]). However, this will break the generality of the solution. As WS-GRAM aims to provide a higher-level job submission functionality abstracting from various LRMs, integrating advance reservation mechanisms directly to interact with WS-GRAM (i.e. making a LRM-independent reservation service) is a more portable and practical approach for better capacity planning of the resources and the agreement enforcement in the environment.

5.2 Standards Adaptation

We adapt WS-Agreement, a proposed Grid Resource Allocation Agreement Protocol [GRAAP-wg 2006], for contract specification. Allocations or co-allocations are composed as WS-Agreements and maintained in the form of WS-Resources [WSRF].

JSDL is another proposed standard for job submission description [JSDL-wg] that consists of a set of constructs which we use to specify constraints as part of the WS-Agreement specification. Furthermore, constraints like *maximum number of allowed processes* and *job termination time* can be specified as part of job description in JSDL, and we plan to further strengthen the process of agreement enforcement by exploiting these parameters.

6 Experiments

We have implemented a prototype of the proposed system as a set of cooperative middleware services, in which reservation instances are maintained as WS-Resources [WSRF] in the form of WS-Agreements. The reservations then can be manipulated by using WSRF features, such as lifecycle management. We tested the system in the Askalon's Grid environment [Fahringer et al.], which is deployed on the Austrian Grid [Consortium]. We performed our experiments on AMD Opteron 64*bit* nodes located on a lightly loaded network with a maximum latency between two nodes of less than 2ms.

We compared our proposed VSHSH algorithm against existing First Fit (FF), Best Fit (BF), and Next Fit (NF) on-line strip packing algorithms [Bays 1977]. The experiments were performed in order to compare the allocation (at node-level) as well as the co-allocation (at Grid-level) algorithms. The start time of each allocation request varied from 1-min to 14-days. A set of requests was randomly generated, and the same set was applied to all the algorithms and tests. The duration of each allocation request was also randomly selected so that 80% of the requests were smaller than 4 hours, and 20% were between 4-36 hours. These values were chosen based on our experience running real Grid workflow applications as described in [Fahringer et al. 2005]. For the co-allocations, we have generated requests following the structure of the workflow application composed of 4 sequential regions with 2 parallel regions. The values corresponding to the parallel region were randomly selected as a multiple of 6 minutes depending on the problem size.

For the measurements, we used the strip splitting and the cost model described in Section 3.1, where the cost varies between strips as well as between shelves, i.e. three strips with 50%, 25% and 25% of the total capacity respectively. Application utility, as described in the Section 2.2, is based on the client request, and is a function of the distance between the expected allocation and the actual/real option offered by the reservation service. The higher the distance, the lower the utility. On the other hand, resource utility is calculated based on the resource utilization and the generated revenue in the form of fictitious money (see Section 4.2).

Figure 8 and 9 compares the VSHSH with the legacy algorithms. For both resource and application utilties we can

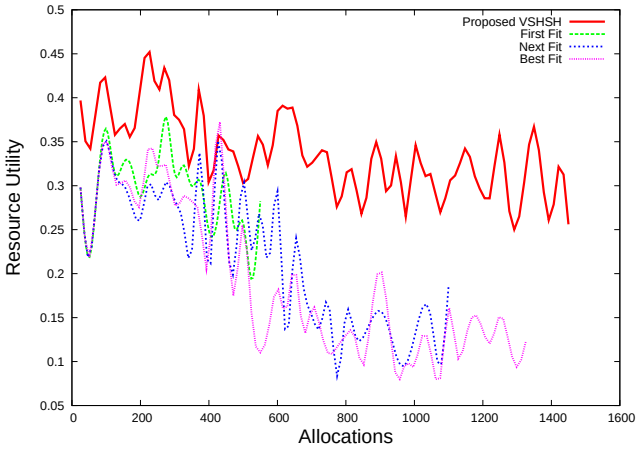

Figure 8: Average Resource Utility with Allocations.

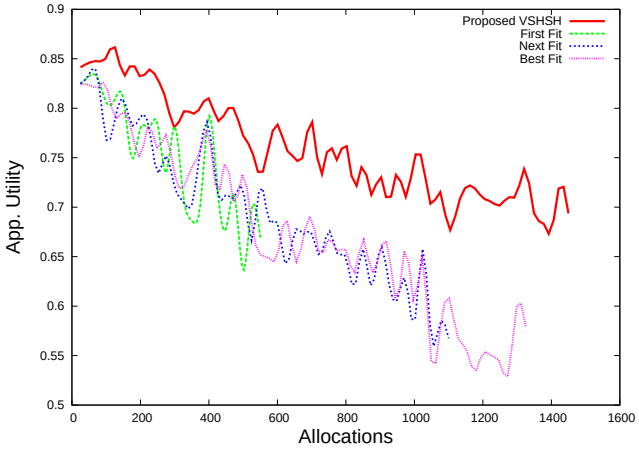

Figure 9: Average Application Utility with Allocations.

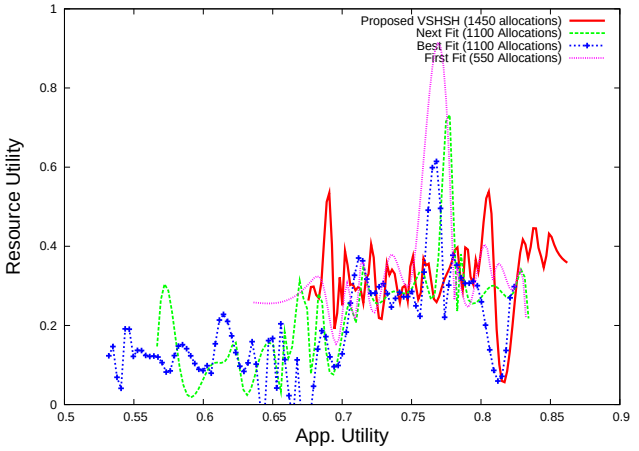

Figure 10: Resource Utility Compared with Application Utility.

Concurrent Users	2	4	6	8
Response Time (ms)	96	124	148	168

Table 2: Average response time per transaction.

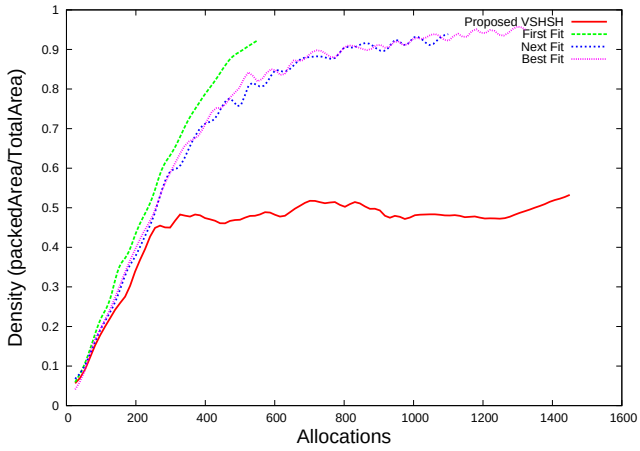

Figure 11: Resource utilization (density).

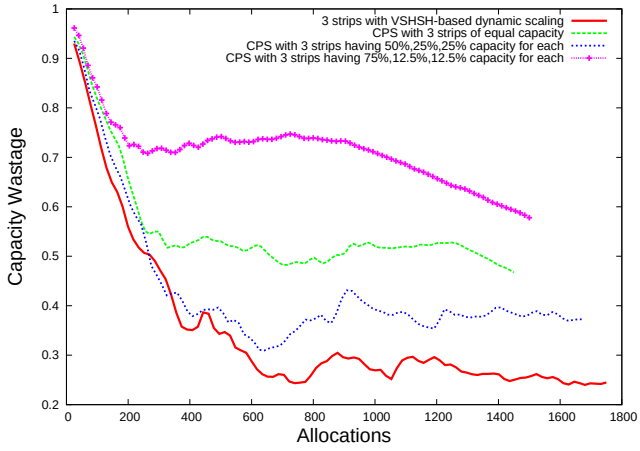

Figure 12: The Resource Wastage with Different Capacity Planning Strategies (CPS).

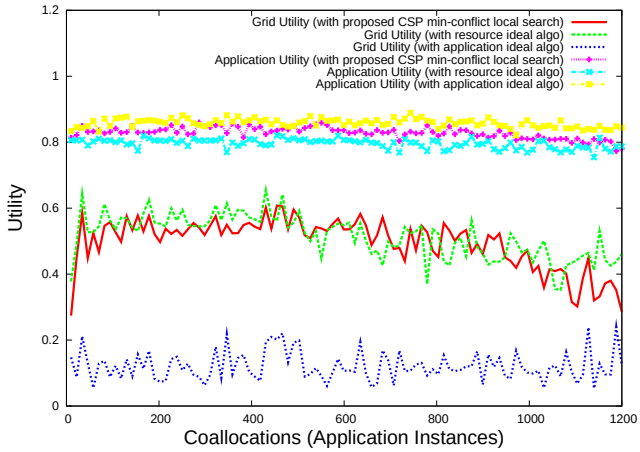

Figure 13: Comparison of Co-Allocation Algorithm with Resource-Ideal and Application-Ideal Allocation Algorithms Showing Expected and Worse Performance.

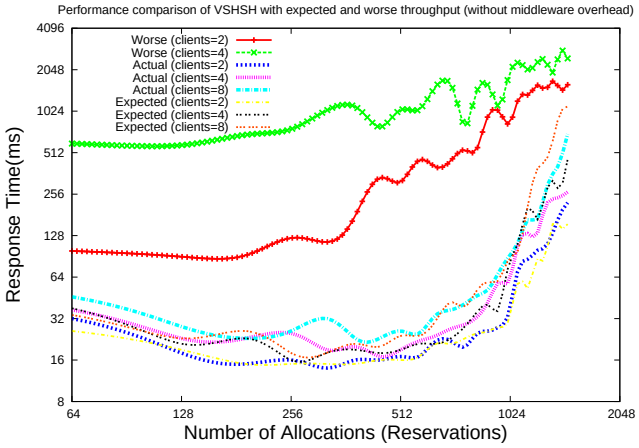

Figure 14: Performance (Response Time per Transaction) of the VSHSH Compared with Expected and Worse Possible with Different Concurrent Clients but Without Grid Middleware Overhead.

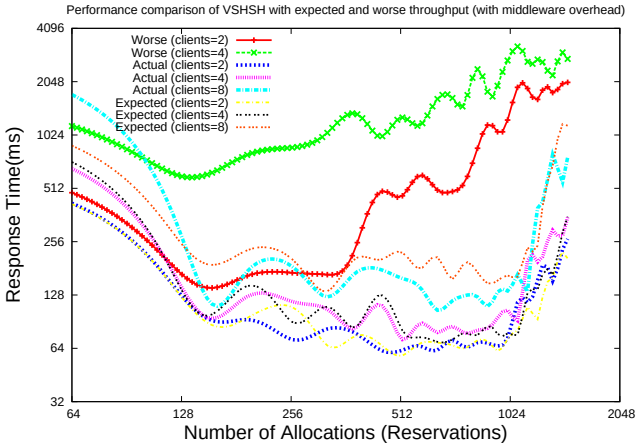

Figure 15: Performance of the VSHSH along with middleware overhead compared with expected and worst possible with different concurrent clients along with Grid middleware overhead.

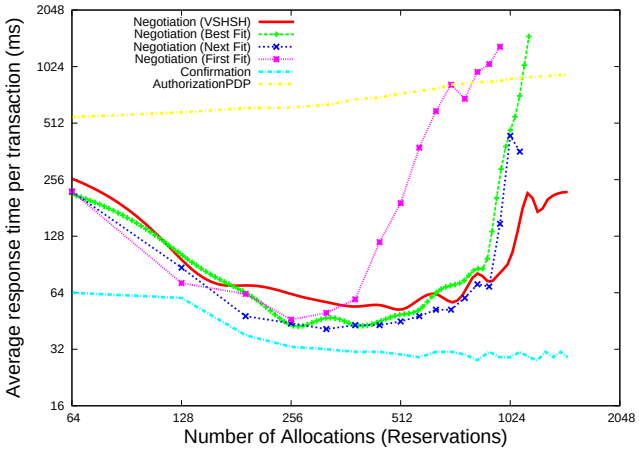

Figure 16: Response time with different allocations.

observe that VSHSH maximizes the utility function. Interestingly, the behavior of VSHSH is similar to the Best Fit algorithm, but with higher utility. In addition, we can observe that resource utility scales well, independently of the number of allocations.

Figure 10 depicts comparison of resource utility against application utility. All existing algorithms maintain the ratio up to some extent, but after handling a certain number of allocations (as reported in the Figure), they stop fulfilling further requests. In particular, First Fit has a higher resource utility (around 0.9), but only handled a maximum of 550 allocations. The VSHSH remains consistent with growing number of allocations and continues accepting further requests.

Figure 11 depicts average density of allocated area, packed with varying number of allocations. The VSHSH shows lower density which is due to the capacity planning strategy in which we get better application and resource utility. We can observe that VSHSH maintains a density of 50% between 200 and 1300 allocations. In addition, VSHSH continues accepting more requests but the density grows linearly after 1300 requests.

Figure 12 compares average capacity wastage, i.e. the unused total area using different capacity planning strategies. We have set different capacity values to the strips in order to see how wastage can be reduced. We can observe that setting equal capacity to each strip is worse than our base settings (50%,25%,25%) used for the previous measurements, and that increasing the first strip capacity (75%,12.5%,12.5%) does not reduces the wastage. This is due to the multiple QoS constraints (cost and capacity) which effects the trade-off between resource utility and application utility. The strategy with dynamically scalable strips shows minimum resource wastage but requires keeping the historic information of the requests.

Figure 13 depicts comparison of the proposed co-allocation offer generation algorithm as described in Section 3.2 that optimizes overall Grid and application utilities. The measurements were performed using 4 allocator instances. We compare the proposed algorithm with two other algorithms: (a) a Grid-ideal that maximizes Grid utility (i.e. the aggregated resource utilities), and (b) an application-ideal that maximizes application utility. We can observe that the application ideal algorithm reduces the resource utility significantly, whereas the resource ideal introduces small reduction to the application utility. In the case of our approach, the results are quite encouraging: the Grid utility is closer to the Grid-ideal, and application utility is closer to the application-ideal algorithm. We performed this experiment by using allocation durations according to the average execution time of each serial and parallel activity of our real-world workflow application. Dependencies of activities are also kept while generating coallocation offers.

Table 3 shows average overhead of different operations of the system for an allocation request. It shows that the ne-

Function	Time (MS)
Negotiation (with FF,NF)	190
Negotiation (with VSHSH)	198
Confirmation	30.5
WS-GRAM (Default)	600
WS-GRAM (with ReservationPDP)	636

Table 3: Average overhead of different operations.

gotiation step, which also involves a compute intensive offer generation, is a bit time-expensive. Confirmation and authorization do not add any significant overhead, as both deal with existing reservations available as WS-Resources. Table 3 (lower row) compares overhead of job submission with WS-GRAM configured with and without reservation-PDP. We performed this test by submitting a small job to WS-GRAM in *"quiet"* and *"batch"* mode, which means that the client did not wait for the completion of the job and returned immediately after successful submission. The average overhead of the special PDP is 36*ms* per job, which is just 6.0% of the total submission overhead of WS-GRAM.

Table 2 shows an average response time for a single negotiation session with varying number of concurrent clients. Response time increases linearly with the number of users due to the compute-intensive algorithm. As most of the Grid applications run for a longer duration, the overhead of the reservation system is quite negligible, even for time-critical scientific applications.

Figures 14 and 15 show the response time of the VSHSH compared with expected and worst possible performance with different number of concurrent clients, i.e. 2, 4, and 8, with and without Grid middleware overhead, respectively. The measurements were performed using the same set of requests as described above. The measurements with the middleware overhead were made calculating the response time for the client, whereas the measurements without the middleware overhead were calculated only at the server side. The relative values of worse performance are taken with the first two weeks are fully packed (reserved), and approximately 23000+ allocations were already made. The values for the expected performance correspond to the maximum utility, i.e. minimum response time required to perform the allocation, and to send the result to the client. The actual value corresponds to the response time of our VSHSH algorithm.

We can observe that the performance of VSHSH is close to the expected performance, and that response-time grows linearly after approximately 1024 allocations. This is because of the increase in number of data structures needed to store the reservation instances. Furthermore, the Grid middleware overhead as shown in Figure 15 is much higher due to communication, WSRF, and container overheads. In addition, there is an observable startup overhead (until 128 reservations), because of the initialization of the middleware and the data objects. The possible worse response time can be reduced by increasing the capacity of the underlying Grid resources including processors and memory.

Finally, Figure 16 shows an average response time for a single *negotiate-reserve* session with varying number of allocations. Initially all algorithms give the same performance with similar behavior, but afterward in contrast to existing algorithms, the VSHSH response time increases linearly instead of exponentially with allocations. Confirmation response time is almost consistent as it deals with already generated reservation. These depictions confirms that the main overhead is of Grid middleware (including communication and WSRF part etc.) and not in the proposed algorithm and the reservation system.

7 Related work

Advance reservation and negotiation-based optimized QoS delivery has been a subject of numerous studies [Hafid et al. 1998; Wolf and Steinmetz 1997; Wang et al. 2003; Netessine and Shumsky(2002)]. A few researchers are also investigating the same mechanisms for the Grid. In this section, we identify distinguishing features of our approach compared with existing work.

In the Grid environment, GARA [Foster et al. 1999] and DUROC [DUROC-team] are the initial works on advance reservation that define a basic architecture and simple API for the manipulation of advance reservation of different resources. These two concentrate mainly on the applicability of resource reservation for job management. Our work focuses on Grid resource provision with advance reservation by optimizing multiple and flexible QoS constraints.

SNAP [Czajkowski et al. 2002] proposes a negotiation protocol for a distributed Grid resource management model in which resource interactions are mapped onto a set of Service Level Agreements (SLAs). This work has been replaced by the GGF proposed standard [GRAAP-wg 2006] on which we based our 3-layered negotiation protocol that generates optimal allocation offers in order to efficiently reach an agreement.

The usefulness of advance reservation is presented at a theoretical level in [McGough et al. 2005]. Modelling and solving resource allocation problems with soft constraint techniques is discussed in [Zhang 2002]. Performance impact of advance reservations on workflows is depicted in [Singh et al. 2005], and effects of different models to support advance reservations are presented in [Smith et al. 2000].

An algorithm described by [Roeblitz et al. 2006] supports fuzziness in a limited set of parameters and applies speedup models to propose flexible allocations. Co-reservations are proposed as virtual resources. However, dynamic Grid behavior, reservation of logical resources, and optimization from the perspective of the Grid had not been considered.

ICENI [McGough et al. 2005] uses a two tier system for reserving resources on the Grid. It exposes reservation capability of underlying resource managers such as [Sun Mi-

crosystems] without considering dynamic behavior of the Grid. No flexible negotiation mechanism that considers multiple QoS parameters other than the timeframe is available.

Gridbus [Buyya and Venugopal 2004] provides a broker service for the Grid resources that takes into account the fact that deadline and budget are specified, and then optimizes the usage of resources only by considering the current state of the resources but without any planning horizon.

Different reservation-aware schedulers such as [Altair ; Platform ; Sun Microsystems] do not consider flexibility of different constraints, thus resulting in lack of negotiability. Maui/Silver [Cluster Resources] is a job scheduler for cluster/Grid systems in which advance reservation scheme makes it possible to allocate local resources in the future. On a Grid node, it can be used in combination with the other LRMs but it requires an extention in the Grid job submission service in order to make it reservation-aware. Also, it does not support any negotiation mechanism.

The work described in [Elmroth and Tordsson 2005] introduces a resource broker, developed as part of the NorduGrid project [NorduGrid-team], that supports advance reservations. However, in order to make it work, the proposed reservation mechanism requires modifications in the basic Grid middleware services like GRAM [Czajkowski et al. 1998] and GridFTP [Allcock et al. 2002]-client/server.

GridSim [Sulistio and Buyya 2004] provides a Grid-simulated infrastructure that also supports simulation of advance reservation, but it does not consider negotiation.

In contrast to these existing works, our work focus on a flexible specification of reservation requests, and a 3-layered negotiation protocol for the provision of optimal allocation offers. We consider dynamic Grid behavior by introducing open reservations that are dynamically mapped to the physical resources before or during application runtime. Multiple QoS parameters are optimized. For maximum utility, an optimal Grid capacity is protected for different classes of the clients. Finally, a practical solution for the enforcement of reservation agreements is introduced, one which can work with different local resource managers without requiring any modification in their functionality.

8 Conclusion

Grid resource management systems have been mostly used for resource brokerage, i.e. the discovery and selection of resources for the Grid applications. In our work, we focused on enhancing a GRMS so that it can work as resource provisioner and make resources available on-demand. Advance reservation of Grid resources can play a key role in enabling a GRMS to deliver on-demand resource provisioning with significantly improved QoS. For this purpose, we introduced a new mechanism for advance reservation of Grid resources. This mechanism provides a 3-layered negotiation protocol for flexible resource allocation. The offered allocations are then used to efficiently reach an agreement with minimal client interactions.

We propose smart, offer-generation algorithms by framing (co)allocation as on-line strip packing and constraint optimization problems, and by providing solutions that optimize the required QoS constraints and resource utilization. Our allocation algorithm, called Vertical Split and Horizontal Shelf-Hanger (WSHSH), introduces a novel approach to optimize application utility with improved capacity planning. Contention elimination and open reservations are supported to deal with dynamic Grid behaviour, whereas, a practical solution for agreement enforcement is provided using state-of-the-art Grid technologies.

A prototype of the proposed system has been implemented to examine the effectiveness of our approach. We have demonstrated that with proper capacity planning using negotiation-based advance reservation, the utilization can be maximized independently of the number of requested allocations. In addition, the proposed approach better deals with the dynamic nature of the Grid and generates more optimal allocations compared to existing heuristics used for NP-hard resource allocation problems. Furthermore, the proposed approach does not add any significant overhead to the existing Grid middleware services.

We plan to extend the allocation strategies and capacity planning with in creased horizon by considering more QoS parameters, for instance, memory, network, and storage etc. An enhancement, such as semantics description of allocations would improve the negotiation process in order to reach a more flexible but optimal resource allocation agreements.

Acknowledgments

We thank our colleagues especially Otheus for their constructive discussions and proof readings, and the anonymous reviewers for their valuable feedback and pointing us to the additional related work.

References

ALLCOCK, B., BESTER, J., BRESNAHAN, J., CHERVENAK, A. L., FOSTER, I., KESSELMAN, C., MEDER, S., NEFEDOVA, V., QUESNEL, D., AND TUECKE, S. 2002. Data management and transfer in high-performance computational grid environments. *Parallel Computing 28*, 5 (May), 749–771.

ALTAIR. Portable batch system (pbs) professional 7.1. http://www.altair.com/software/pbspro.htm.

BARMOUTA, A., AND BUYYA, R. 2003. Gridbank: A grid accounting services architecture (gasa) for distributed systems sharing and integration. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IEEE Computer Society, Washington, DC, USA, 245.1.

BAYS, C. 1977. A comparison of next-fit, first-fit, and best-fit. *Commun. ACM 20*, 3, 191–192.

BUYYA, R., AND VENUGOPAL, S. 2004. The gridbus toolkit for service oriented grid and utility computing: An overview and status report. In *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models*, IEEE Press, New Jersey, USA, 19–36.

CLUSTER RESOURCES, I. Super clusters: Center for hpc cluster resource management and scheduling. http://www.supercluster.org/projects/.

CONSORTIUM, T. A. G. http://www.austriangrid.at.

COOPER, K., DASGUPTA, A., KENNEDY, K., KOELBEL, C., MANDAL, A., MARIN, G., MAZINA, M., MELLOR-CRUMMEY, J., BERMAN, F., CASANOVA, H., CHIEN, A., DAIL, H., LIU, X., OLUGBILE, A., SIEVERT, O., XIA, H., JOHNSSON, L., LIU, B., PATEL, M., REED, D., DENG, W., MENDES, C., SHI, Z., YARKHAN, A., AND DONGARRA, J. 2004. New Grid Scheduling and Rescheduling Methods in the GrADS Project. In *International Parallel and Distributed Processing Symposium, Workshop for Next Generation Software*, IEEE Computer Society Press.

CSIRIK, J., AND WOEGINGER, G. J. 1997. Shelf algorithms for on-line strip packing. *Inf. Process. Lett. 63*, 4, 171–175.

CZAJKOWSKI, K., FOSTER, I., KARONIS, N., MARTIN, S., SMITH, W., AND TUECKE, S. 1998. A Resource Management Architecture for Metacomputing Systems. In *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson and L. Rudolph, Eds. Springer Verlag, 62–82. Lect. Notes Comput. Sci. vol. 1459.

CZAJKOWSKI, K., FOSTER, I., KESSELMAN, C., SANDER, V., AND TUECKE, S., 2002. Snap: A protocol for negotiating service level agreements and coordinating resource management in distributed systems.

DUROC-TEAM, G. The dynamically updated request online coallocator. http://www-fp.globus.org/duroc/.

ELMROTH, E., AND TORDSSON, J. 2005. A Grid Resource Broker Supporting Advance Reservations and Benchmark-based Resource Selection. In *State-of-the-art in Scientific Computing.*, Springer Verlag, vol. 3732 of *Lecture Notes in Computer Science*, 1077–1085.

FAHRINGER, T., PRODAN, R., DUAN, R., HOFER, J., NADEEM, F., NERIERI, F., PODLIPNIG, S., QIN, J., SIDDIQUI, M., TRUONG, H.-L., VILLAZON, A., AND WIECZOREK, M. Askalon: A development and grid computing environment for scientific workflows. *Workflows for eScience, Scientific Workflows for Grids.*

FAHRINGER, T., PRODAN, R., DUAN, R., NERIERI, F., PODLIPNIG, S., QIN, J., SIDDIQUI, M., TRUONG, H.-L., VILLAZON, A., AND WIECZOREK, M. 2005. ASKALON: A Grid Application Development and Computing Environment. In *6th International Workshop on Grid Computing (Grid 2005)*, IEEE Computer Society Press, Seattle, Washington, USA.

FOSTER, I., KESSELMAN, C., LEE, C., LINDELL, R., NAHRSTEDT, K., AND ROY, A. 1999. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service*.

GLOBUS. The globus alliance. http://www.globus.org/toolkit.

GRAAP-WG, G. G. F., 2006. Grid resource allocation agreement protocol. https://forge.gridforum.org/projects/graap-wg.

HAFID, A., VON BOCHMANN, G., AND DSSOULI, R. 1998. A quality of service negotiation approach with future reservations (NAFUR): a detailed study. *Computer Networks and ISDN Systems 30*, 8, 777–794.

JACKSON, D. B., SNELL, Q., AND CLEMENT, M. J. 2001. Core algorithms of the maui scheduler. In *JSSPP '01: Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, Springer-Verlag, London, UK, 87–102.

JSDL-WG, G. G. F. Job submission description language (jsdl). http://www.ggf.org/documents/GFD.56.pdf.

MCGOUGH, A. S., AFZAL, A., DARLINGTON, J., FURMENTO, N., MAYER, A., AND YOUNG, L. 2005. Making the grid predictable through reservation and performance modelling. *The Computer Journal 48*, 3, 358–368.

NETESSINE, S., AND SHUMSKY(2002), R. Introduction to the theory and practice of yield management. *INFORMS Transactions on Education 3*, 1.

NORDUGRID-TEAM. NorduGrid project. http://www.nordugrid.org.

PLATFORM. Lsf. http://www.platform.com/Products/ Platform.LSF.Family.

ROEBLITZ, T., SCHINTKE, F., AND REINEFELD, A. 2006. Resource reservations with fuzzy requests. *Concurrency and Computation: Practice and Experience*.

SHANG, Y., AND WAH, B. W. 1998. A discrete lagrangian-based global-search method for solving satisfiability problems. *J. of Global Optimization 12*, 1, 61–99.

SIDDIQUI, M., AND FAHRINGER, T. 2005. GridARM: Askalon's Grid Resource Management System. In *Advances in Grid Computing - EGC 2005 - Revised Selected Papers*, Springer Berlin / Heidelberg, ISBN 3-540-26918-5, vol. 3470 of *Lecture Notes in Computer Science*, 122–131.

SIDDIQUI, M., FAHRINGER, T., HOFER, J., AND TOMA, I. 2005. Grid resource ontologies and asymmetric resource-correlation. In *2nd International Conference on Grid Service Engineering and Management (GSEM'05)*, Lecture Notes in Informatics, Erfurt, Germany, G. S. of Informatics, Ed.

SIDDIQUI, M., VILLAZON, A., HOFER, J., AND FAHRINGER, T. 2005. GLARE: A grid activity registration, deployment and provisioning framework. In *International Conference for High Performance Computing, Networking and Storage (SuperComputing), SC05*, ACM Press, ISBN 1-59593-061-2/05/0011, Seattle, Washington, USA, ACM, Ed.

SINGH, G., KESSELMAN, C., AND DEELMAN, E. 2005. Performance impact of resource provisioning on workflows. Technical report 05-850, Information Sciences Institute, University of Southern California. http://www.cs.usc.edu/Research/TechReports/05-850.pdf.

SMITH, W., FOSTER, I., AND TAYLOR, V. 2000. Scheduling with advanced reservations. In *14th International Parallel and Distributed Processing Symposium (IPDPS'00)*.

SULISTIO, A., AND BUYYA, R. 2004. A grid simulation infrastructure supporting advance reservation. In *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems*, ACTA Press, MIT Cambridge, Boston, USA.

SUN MICROSYSTEMS, I. Sun grid engine. http://www.sun.com/software/gridware/.

TSCHUDIN, C. 1997. Open Resource Allocation for Mobile Code. In *First International Workshop on Mobile Agents*.

WALDSPURGER, C., AND WEIHL, W. 1994. Lottery scheduling: Flexible proportionalshare resource management. In *First Symposium on Operating System Design and Implementation (OSDI)*, USENIX.

WANG, G., ZHANG, W., MAILLER, R., AND LESSER, V. 2003. *Analysis of Negotiation Protocols by Distributed Search*. Kluwer Academic Publishers, 339–361.

WOLF, L., AND STEINMETZ, R. 1997. Concepts for Resource Reservation in Advance. *Multimedia Tools and Applications 4*, 3 (May), 255–278. Special Issue on State of the Art in Multimedia Computing.

WSRF, G. A. Web services resource framework. http://www.globus.org/wsrf.

YOKOO, M., DURFEE, E. H., ISHIDA, T., AND KUWABARA, K. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering 10*, 5, 673–685.

YU, J., AND BUYYA, R. 2005. A Taxonomy of Workflow Management Systems for Grid Computing. Technical report grids-tr-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, Mar. http://www.gridbus.org/.

ZHANG, W. 2002. Modeling and solving a resource allocation problem with soft constraint techniques. Technical report wucs-2002-13, Department of Computer Science and Engineering, Washington University in St. Louis, May.