Specifying and Processing Co-Reservations in the Grid

Thomas Röblitz
Zuse Institute Berlin, Takustr. 7, D-14195 Berlin, Germany
roeblitz@zib.de

Abstract: Executing complex applications on Grid infrastructures necessitates the guaranteed allocation of multiple resources. Such guarantees are often implemented by means of advance reservations. Reserving resources in advance requires multiple steps – beginning with their description to their actual allocation. In a Grid, a client possesses little knowledge about the future status of resources. Thus, manually specifying successful parameters of a co-reservation is a tedious task. Instead, we propose to parametrize certain reservation characteristics (e.g., the start time) and to let a client define criteria for selecting appropriate values. Then, a Grid reservation service processes such requests by determining the future status of resources and calculating a co-reservation candidate which satisfies the criteria. In this paper, we present the *Simple Reservation Language (SRL)* for describing the requests, demonstrate the transformation of an example request into an integer program using the *Zuse Institute Mathematical Programming Language (ZIMPL)* and experimentally evaluate the time needed to find the optimal co-reservation using *CPLEX*.

1 Introduction

In many scientific disciplines, large scale simulations and workflows for analyzing petascale data sets necessitate the adoption of Grid technologies to meet their demanding resource requirements. The use of Grid resources, however, poses new challenges, because of their heterogeneity, geographic distribution and autonomous management. Especially, the efficient execution of complex applications, e.g., large simulations of black holes with Cactus or distributed observations of astronomic objects with robotic telescope as envisioned in the AstroGrid-D project [GAC07], requires the *co-allocation* of multiple Grid resources. Because Grid resources are autonomously managed, the allocation of them at the same time or in some sequence cannot be *guaranteed* by standard Grid job management schemes, i.e., a broker decides where to submit a job, but has no control on when the job is actually executed.

This problem can be circumvented by acquiring *reservations* for the required resources. A reservation guarantees that a specific resource can be allocated to a request at the desired time. Typically, reservations have a fixed duration and are acquired some time in advance. Considering the vast number of resources eligible for reservation, a user may pose constraints on which resources are selected. In addition, the provider may want to optimize the utilization of their resources. Hence, a broker has to solve a complex optimization problem to find a mapping of application components to resources at specific times.

In this paper, we model the mapping problem as an integer programming problem and use a standard solver to find its best solution. We use the *Zuse Institute Mathematical Programming Language* (ZIMPL) [Koc04] to describe the integer programming problem and the solver *CPLEX* for finding the solution. To simplify the specification of co-reservation requests, we

propose a simple language for defining the requirements of each part along with the objective to be optimized. Throughout the paper, we reuse an example presented in our previous work [RR05]. The example describes a co-reservation request including temporal and spatial relationships between atomic request parts.

> *Reserve 16 CPUs of an IBM p690, 32 CPUs of a PC cluster and a one Gbit/s-network connection between them, each for six hours between 2007/12/12 06:00pm and 2007/12/15 06:00pm. All reservations must start at the same time. Reserve a visualization pipeline for two hours starting four hours after the reservation on the IBM p690 begins and a 100 Mbit/s-network connection between the p690 and the visualization pipeline for the same time.*

Outline. We summarize related work in Section 2. The general procedure for processing co-reservation requests is presented in Section 3. Section 4 describes the *Simple Reservation Language (SRL)* for specifying requests in detail. Thereafter, we demonstrate – using the scenario described above – the transformation of SRL requests into an optimization problem in Section 5. Then, we present an experimental evaluation of the times to find an optimal co-reservation candidate in Section 6 and conclude in Section 7.

2 Related Work

In [FKL+99], Foster et al. present GARA, a resource management architecture for reserving multiple resources in advance. The focus of their work is on the general scheme to guarantee that an application can co-allocate multiple resources at specific times. They also discuss needed extensions of existing local resource management such that they support advance reservations. Our work focuses on the description and processing of co-reservation requests. The needed components for processing requests are similar to those in GARA. Hence, our mechanism could be easily integrated in their framework.

Condor ClassAds, proposed by Raman et al. [RLS98], provides a mechanism for matching atomic requests to resources. Condor ClassAds is a very versatile language to specify both a user's request and the offers of the resource providers. The language we propose for describing reservation requests is conceptually similar to ClassAds, because it is also based on $(attribute, value)$-pairs and allows references between different matching parties. The main reason for not using pure ClassAds was that the processing of them differs from the processing of multi-resource reservation requests. Liu and Foster extend Condor ClassAds by applying constraint programming techniques to the problem of selecting resources for execution of immediate requests in a Grid environment [LF03].

In [NLYW05], Naik et al. present an integer programming based approach for assigning requests to heterogeneous resources. In contrast to our work, they consider immediate requests only. That is the resource manager tries to match as many requests to resources at the current time. Scheduling requests at future times is not considered. The VIOLA metascheduler [WWZ05] schedules parallel compute jobs to multiple resources by incrementally increasing the advance booking period until all jobs may be allocated. In contrast to our work, it only supports one criteria – earliest completion time.

3 The Reservation Framework

The framework consists of the three main components: the *Grid Reservation Service (GRS)*, the *Local Reservation Service (LRS)*, and the *Resource Information Service (RIS)*. The general procedure for processing co-reservation requests is as follows.

① A user sends a request described by the *Simple Reservation Language* to the GRS.

② The GRS queries the RIS to determine resources which support advance reservation and match the core characteristics such as type and operating system.

③ The GRS sends *probe* requests to the LRSes of these resources to let them provide detailed information about their future status and reservation fees.

④ The GRS transforms the user request including the probed information into the *ZIMPL* format and thereby instantiates the optimization problem.

⑤ The GRS determines the best co-reservation candidate. A co-reservation candidate maps each request part to a $(resource, starttime)$-pair.

⑥ The GRS tries to acquire the reservations given by the best co-reservation candidate. Thus, for each $(resource, starttime)$-pair the GRS sends a *reserve* request to the LRS of the resource *resource*.

⑦ If some *reserve* requests are denied, the GRS refines the optimization problem and proceeds with step ⑤.

⑧ If all *reserve* requests are admitted or no solution could be found eventually, the GRS sends a response message to the client.

Note, in this paper we focus on the steps ①, ④ and ⑤. Step ② is implemented in todays Grid resource brokers (cf. Section 2). Methods for determining the future status of resources are described in our previous work [RR06, RSR06]. The actual acquiring of the reservations (step ⑥) and the refinement in case of failures (step ⑦) are subject to future work.

4 The Simple Reservation Language

The purpose of the *Simple Reservation Language (SRL)* is twofold. First, it enables clients to easily describe reservation requests without requiring them to know the details of a mathematical programming language. Second, the Grid Reservation Service (GRS) can efficiently pre-process reservation requests, because the SRL uses a small set of attributes and limited value domains only. Despite these restrictions, the language is powerful enough to describe a large variety of requests as will be seen in the following subsections. As outlined in the introduction, we regard the co-reservation problem as an optimization problem. Formally, a co-reservation request must define a set of variables including their domains, a set of constraints on these variables' domains and an objective function which is to be optimized. The SRL follows a less formal approach by letting a user define certain attributes, i.e., the SRL vocabulary, which are transformed into the corresponding mathematical terms.

Table 1: Attribute scopes of the Simple Reservation Language.

Used in	Scope	Description
key & value	TS	Temporal specification of a request
key & value	QOS	QoS specification of a request
key & value	MISC	Miscellaneous attributes of a request
key	CON	Constraints of a request
key	OBJ	Objectives of a request
value	RVC	Attributes of a reservation candidate

4.1 Structure of a Co-Reservation Request

A co-reservation request consists of multiple atomic requests as well as constraints and objectives defining the relationships between any two atomic requests. Each SRL request is a set of $(attribute, value)$-pairs. The domain of a value depends on the attribute and will be discussed in the following subsections. The syntax of an attribute is defined as

```
<attribute> := <id>'.'<scope>'.'<key>.
```

The *key* of an attribute is an alpha-numeric string. For each scope there exist several keys with a pre-defined meaning which we will present along with the discussion of the scopes. In addition, any other string may be used, but its meaning is only defined by the request itself. The *scope* denotes a specific group of attributes. We distinguish three kinds of scopes, based on its appearance in an $(attribute, value)$-pair (cf. Table 1). The component `<id>` associates attributes with a specific part of a co-reservation request. Thus, it is possible to reference attributes of a specific part from within any other part.

4.2 Description of Atomic Requests

The vocabulary of the SRL must cope with the following issues to describe a (meaningful) atomic request:

- When should the resource be reserved?
- Type and quantity of the resource to be reserved.
- Which constraints must be met?
- Which criteria should be optimized?

Temporal specification. The scope `TS` defines three attributes `est` (earliest start time), `let` (latest end time) and `duration` for specifying, when the resource should be reserved. The values can be given as epoch seconds or UTC[1] string.

[1] UTC stands for Coordinated Universal Time.

Quality-of-Service Specification. The scope `QOS` defines attributes for specifying what type of resource and what quantity or quality of this resource should be reserved. These attributes are `QOS.type`, `QOS.cpus`, `QOS.netbw`, `QOS.diskspace`. The GRS does not handle the differences among the corresponding resource types internally. It must just ensure to contact the right local reservation service for each type of resource. The values for `QOS.type` are `compute`, `network`, `storage` and `visualize`. The values for the corresponding quantity attributes are within the usual domains.

Miscellaneous Attributes. The scope `MISC` may contain any attribute which does not fit into the previous categories. For example, it may be used to associate a user id or certificate with a request which could be necessary for authentication and authorization.

Constraint Specification. The scope `CON` is used to define constraints on any attributes of all scopes except `CON` and `OBJ`. The key of a `CON`-attribute can be any alpha-numeric string. Two different attributes must not use the same key. The syntax of the value of a `CON`-attribute is defined as follows.

```
<con value> := <expr><op><expr>
<expr>      := <attribute>|<quantity>
<op>        := '<'|'<='|'=='|'!='|'>='|'>'
```

The term `<attribute>` refers to any attribute (except for the scopes `CON` and `OBJ`). For example, the cost `IBM.RVC.cost` for reserving the requested resources could be limited. The term `<quantity>` specifies a quantity such as 100 Mbit/s.

Objectives Specification. The scope `OBJ` defines the objective of the request. The objective is the weighted sum of all sub-objectives. A client must assign a weight to each sub-objective such that their sum equals one. Because the range of values of the sub-objective attributes may be very different and not known a-priori, each attribute must be normalized. The syntax of the actual sub-objective specification is as follows.

```
<obj> := {'min'|'max'},'<attribute>','<weight>
```

The term `<attribute>` refers to any attribute (except for the scopes `CON` and `OBJ`). For example, to execute the application on the IBM p690 at the earliest moment, the sub-objective is `min,IBM.RVC.begin,0.6`.

Reservation Candidates. The scope `RVC` contains attributes of a **R**eser**V**ation **C**andidate, which are determined in the probing phase (cf. step ③).

4.3 Description of Co-Reservation Requests

A *co-reservation request* consists of multiple atomic requests, additional constraints and objectives defining the relationships among the atomic parts. Relationships among different parts are enabled by the identifier component `<id>` of an attribute's key. Thus, attributes of the

part <id> can be referenced in the constraints and objectives of any other part. For example, the *same time*-constraint in the introduction's scenario can be specified as `IBM.RVC.begin == PCC.RVC.begin`. This scheme is very flexible. Also, workflow-like applications may be modeled, i.e., that one part should start only after another has finished. In that case the constraint would be `VIS.RVC.begin >= IBM.RVC.end`. Additionally, objectives may reference attributes in different parts. For example, the above workflow application may desire that its whole execution time is minimized. Such goal can be specified by the sub-objective `min,VIS.RVC.end-PCC.RVC.begin,10`. In Fig. 1, the main parts of the scenario of the introduction are described using the Simple Reservation Language.

```
IBM.TS.est   = Dec 12 18:00:00 2007
IBM.TS.duration = 21600
IBM.TS.let   = Dec 15 18:00:00 2007
IBM.QOS.type = compute
IBM.QOS.cpus = 16
IBM.QOS.arch = "IBM p690"
IBM.CON.cost = IBM.RVC.cost <= 10000
IBM.OBJ.cost = min,IBM.RVC.cost,8
IBM.OBJ.start = min,IBM.RVC.begin,10
PCC.TS.duration = 21600
PCC.QOS.type = compute
PCC.QOS.cpus = 32
PCC.QOS.arch = "PC cluster"
PCC.CON.time = PCC.RVC.begin
               == IBM.RVC.begin
VIS.TS.duration = 7200
VIS.TS.let   = Dec 15 18:00:00 2007
VIS.QOS.type = visualize
VIS.QOS.cpus = 4
VIS.QOS.arch = "SGI Onyx"
VIS.CON.time = PCC.RVC.begin+14400
               == VIS.RVC.begin
```

Figure 1: A co-reservation request described in SRL (in extracts).

5 Transforming SRL Requests into ZIMPL

In the following, we exemplify how SRL requests are transformed into the *Zuse Institute Mathematical Programming Language* (ZIMPL) [Koc04]. The result can then be used by standard integer programming solvers to find the best reservation candidate. Listing 1 shows the complete implementation of the scenario described in the introduction. It consists of seven parts, which we will now explain in detail.

Request Structure. The number of atomic request parts is read from a configuration file in line 2. The example co-reservation request contains 5 parts (one IBM p690, one PC cluster, one network link IBM–PC, one SGI visualization resource and one network link SGI–IBM).

Reservation Candidates. The set of reservation candidates is initialized in line 5. This set is transformed into a multi-dimensional array (lines 6–8) to make individual metrics of a candidate accessible. Note, by separating the reservation candidates for each part (first index in array rvc), we ensure that a request is matched to a resource with the required type.

Model Variables. Each atomic request $r \in R$ must be assigned to a resource $s \in S$ at a certain start time $t \in T$. Thus, we model the problem with $R \times S \times T$ *binary* variables (line 11). The variable xb[r, s, t] is set to 1 iff the atomic request r is assigned to resource s at start time t.

Matching and Cost Constraints. We constrain the number of binary variables set to 1 such that each request part is only once assigned to a resource (line 14). The total cost for all parts of the co-reservation are limited in line 15.

Listing 1: Implementation of the example scenario as *integer program* in ZIMPL.

```
1  # request parts
2  set P := { read "req.dat" as "<1n>" comment "#" };
3
4  # reservation candidates
5  set PRTMV  := { read "rvc.dat" as "<1n,2n,3n,4n,5n>" comment "#"};
6  set RVC    := proj(PRTMV, <1,2,3>);
7  set METRIC := proj(PRTMV, <4>);
8  param rvc[RVC*METRIC] := read "rvc.dat" as "<1n,2n,3n,4n> 5n" comment "#" default 100;
9
10 # model variables
11 var xb[RVC] binary;
12
13 # matching, type and cost constraints
14 subto once: forall <p> in P: (sum <p,i,j> in RVC: xb[p,i,j]) == 1;
15 subto cost: (sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,1]) <= 10000;
16
17 # temporal relationships
18 subto temp1: sum <1,i,j> in RVC: xb[1,i,j]*j == sum <2,m,n> in RVC: xb[2,m,n]*n;
19
20 subto temp2: sum <1,i,j> in RVC: xb[1,i,j]*j == sum <3,m,n> in RVC: xb[3,m,n]*n;
21
22 subto temp3: sum <1,i,j> in RVC: xb[1,i,j]*(j+14400) == sum <4,m,n> in RVC: xb[4,m,n]*n;
23
24 subto temp4: sum <4,i,j> in RVC: xb[4,i,j]*j == sum <5,m,n> in RVC: xb[5,m,n]*n;
25
26 # spatial relationships
27 set NET  := proj({ read "net.dat" as "<1n,2n>" comment "#" }, <1>);
28 set ENDS := { <1>, <2> };
29 param net[NET*ENDS] := read "net.dat" as "<1n,2n> 3n" comment "#";
30
31 subto spat1: sum <1,i,j> in RVC: xb[1,i,j]*i ==
32              sum <m> in NET: sum <3,m,n> in RVC: xb[3,m,n]*net[m,1];
33
34 subto spat2: sum <2,i,j> in RVC: xb[2,i,j]*i ==
35              sum <m> in NET: sum <3,m,n> in RVC: xb[3,m,n]*net[m,2];
36
37 subto spat3: sum <1,i,j> in RVC: xb[1,i,j]*i ==
38              sum <m> in NET: sum <5,m,n> in RVC: xb[5,m,n]*net[m,1];
39
40 subto spat4: sum <4,i,j> in RVC: xb[4,i,j]*i ==
41              sum <m> in NET: sum <5,m,n> in RVC: xb[5,m,n]*net[m,2];
42
43 # objective function
44 set OBJS := { <1>, <2>, <3> }; # 1-cost, 2-time & 3-bandwidth
45 param CF[P*OBJS] := read "coeff.dat" as "<1n,2n> 3n" comment "#";
46
47 minimize objective:  sum <p> in P:
48   ( sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,1]*CF[p,1]
49   + sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,2]*CF[p,2]
50   - sum <p,i,j> in RVC: xb[p,i,j]*rvc[p,i,j,3]*CF[p,3] );
```

Temporal Relationships. We require that the start times of part one (IBM p690) and part two (PC cluster) must be equal (line 18). Also, the start times of part one and part three (network IBM–PC) must be equal (line 20). The next constraint (line 22) requires that the SGI visualization part begins exactly four hours after the computational parts begin (e.g., the IBM part). The last temporal constraint (line 24) ensures that the network between the IBM and the SGI is reserved from the same start time as the SGI part. Note, due to the fixed durations of all parts we do not need to put constraints on the end times of the reservations.

Table 2: Parameters of the experimental evaluation.

Parameter	Values
no. of resources	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20
no. of candidates	7, 12, 23, 34, 67, 133, 265, 397
corresponding time gap	11h, 6h, 3h, 2h, 1h, 30m, 15m, 10m

Spatial Relationships. We assume a fully connected network with bi-directional links. Thus, n resources (compute and visualization) are connected by n^2 links. The network configuration is initialized from line 27 to line 29. The actual spatial relationships are defined for the network link between the IBM p690 part and the PC cluster (lines 31/32 and 34/35) and for the network link between the IBM p690 part and the SGI visualization pipeline (lines 37/38 and 40/41). For the sake of simplicity, we use the number of a resource as its location identifier (cf. multiplication with i on the left side of each comparison).

Objective Function. We use the weighted sum as global objective function (lines 47–50). It aggregates three sub-objectives – minimum cost, minimum start time and maximum available bandwidth. Because the value ranges of the metrics differ significantly (cost: 0-20000, start time: 0-237600, bandwidth: 0-1000), we scale them by appropriate factors (the maximum of each value range). These factors are the coefficients initialized in line 45.

6 Experimental Evaluation of the Scalability

Many optimization problems, in particular integer problems, suffer from a large search space. We studied the impact of the number of eligible resources and the number of reservation candidates on the time needed to find the optimal co-reservation candidate. In the absence of workload traces for co-reservations we randomly generated the reservation candidates (step ③). For each parameter pair (no. of resources, no. of candidates), we generated 10 experiments and calculated the average time for finding the optimal co-reservation candidate. Table 2 lists the parameters of all experiments, which were sequentially executed on a SUN Galaxy 4600 16 core system with 64 Gbytes of RAM. Each experiment used a single processor core only.

Fig. 2 shows the solving time against the number of reservation candidates. Each curve represents the experiments with a specific number of resources. Additionally, the graph shows two approximations of the solving time. For the experiments with one resource, the solving time increases exponentially with the number of candidates. For the experiments with 20 resources, the solving time increases quadratically with the number of candidates.

Whether the solving time is acceptable in real world scenarios depends on several parameters. First, a client may want a response as soon as possible. Second, the calculated future status (cf. step ③) may only be valid for a certain time. Thereafter, the "best" co-reservation candidate is sub-optimal or reservation attempts (cf. step ⑥) simply fail. Third, the longer the book-ahead time (earliest start time) of the co-reservation is, the longer solving times may be acceptable.

The experimental results provide two means for limiting the solving time – (1) the GRS may ask the resource providers for a limited number of reservation candidates and (2) use less eligible resource than found through the resource information service query (cf. step ②).

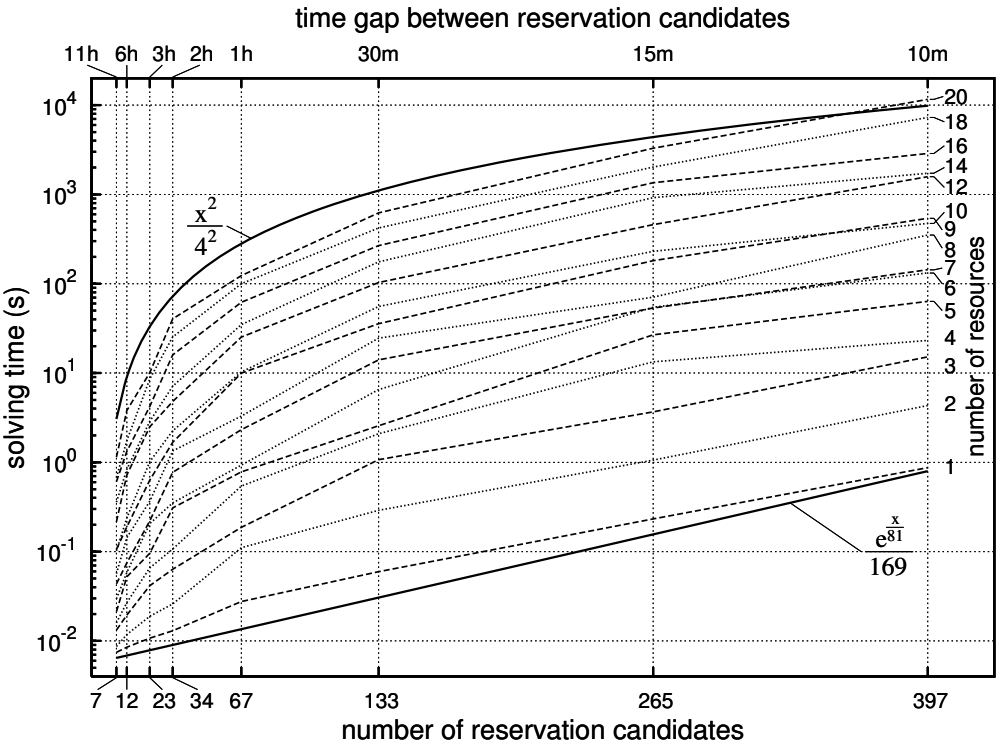

Figure 2: Solving time for several numbers of resources against the number of reservation candidates.

7 Conclusion

Resource allocation for complex applications requires guarantees to ensure desired quality-of-service levels. Such guarantees are typically implemented by reserving the requested resources in advance. We demonstrated an approach for specifying and processing co-reservation requests. The requests are specified in a simple yet powerful language and transformed into an integer program. Our approach is motivated by the observation, that finding the best co-reservation is an optimization problem. The use of a mathematical programming language makes extensions and refinements easy to implement. Moreover, there already exist well-known tools for solving optimization problems described in such languages. Of course, that flexibility comes at some cost namely the time needed to find a solution. The performance experiments revealed means to limit the solving time, i.e., by limiting the number of reservation candidates and/or by using less eligible resources. Nevertheless, the flexibility of the approach facilitates many extensions such as specifying data dependencies and optimizing data transfers. Also, moldable requests may be supported to optimize the utilization of the resources.

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research within the D-Grid initiative under contract 01AK804C and the European Network of Excellence CoreGRID, Institute on Resource Management and Scheduling, under contract IST-2002-004265.

References

[FKL+99] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *Proceedings of the International Workshop on Quality of Service*, pages 27–36. IEEE Press: Piscataway, NJ, 1999.

[GAC07] AstroGrid-D project homepage. http://www.gac-grid.org/, November 2007.

[Koc04] Thorsten Koch. *Rapid Mathematical Programming*. PhD thesis, Technische Universität Berlin, 2004. ZIB-Report 04-58.

[LF03] Chuang Liu and Ian Foster. A Constraint Language Approach to Grid Resource Selection. Technical Report TR-2003-07, Department of Computer Science, University of Chicago, March 2003.

[NLYW05] Vijay K. Naik, Chuang Liu, Lingyun Yang, and Jonathan Wagner. On-line Resource Matching in a Heterogeneous Grid Environment. In *Proceedings of the IEEE International Symposium on Cluster computing and Grid 2005 (CCGrid05), Cardiff, Wales, UK*, volume 2, pages 607–614, May 2005.

[RLS98] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, Illinois, USA*, pages 140–146. IEEE Computer Society Press, July 1998.

[RR05] Thomas Röblitz and Alexander Reinefeld. Co-Reservation with the Concept of Virtual Resources. In *Proceedings of the IEEE International Symposium on Cluster computing and Grid 2005 (CCGrid05), Cardiff, Wales, UK*, volume 1, pages 398–406, May 2005.

[RR06] Thomas Röblitz and Krzysztof Rzadca. On the Placement of Reservations into Job Schedules. In *Proceedings of the 12th International Euro-Par Conference 2006, Dresden, Germany*, pages 198–210, 2006.

[RSR06] Thomas Röblitz, Florian Schintke, and Alexander Reinefeld. Resource Reservations with Fuzzy Requests. *Concurrency and Computation: Practice and Experience*, 18(13):1681–1703, November 2006.

[WWZ05] Oliver Wäldrich, Philipp Wieder, and Wolfgang Ziegler. A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources. In *Proceedings of the 6th International Conference on Parallel Processing (PPAM 2005), Poznan, Poland*, volume 1, pages 782–791, September 2005.