Thomas Röblitz

Co-Reservation of Resources in the Grid

Dissertation
zur Erlangung des akademischen Grades
doctor rerum naturalium (Dr. rer. nat) im Fach Informatik
eingereicht an der Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

Gutachter:

1. Prof. Dr. Alexander Reinefeld
2. Prof. Dr. Miroslaw Malek
3. Prof. Dr. Hans-Ulrich Heiß

Tag der mündlichen Prüfung: 18. Dezember 2008

Für Renate und Dieter

Acknowledgments

This work has been carried out from 2004 to 2008 at the *Computer Science Research* department at the Zuse Institute Berlin. I would like to thank my supervisor Prof. Dr. Alexander Reinefeld for letting me pursue this research project and giving me the freedom and time to dig into all the obscure details. The first contact with the topic was made through a cooperation with IBM Böblingen in 2002. Particularly, I would like to thank Tony Gargya and his colleagues for many discussions at that time. I am also grateful for the support by Prof. Dr. Hans-Ulrich Heiß, who invited me to present my work in his group which gave impetus to long lasting regular discussions with his staff members Jörg Schneider, Barry Linnert and Julius Gehr. There is almost no single part of the thesis which was not influenced by these discussions. I would also like to thank Krzysztof Rzadca, then at IMAG Grenoble and PJIT Warsaw, for our intense collaboration on the advanced method for determining the future status of resources.

Of course, I wish to thank Florian Schintke – my "room mate" over all the time at Zuse. Not only did he helped a lot with bootstrapping the first work (probing of resource status) and provided good feedback to many raw ideas, he also made life at Zuse very enjoyable (lots of chocolate, XBlast-ing, chess, talking about "Gott und die Welt", taking care of my plants while mountaineering far away, . . .).

I owe a lot to my parents, introducing me to computers at a time when they cost a fortune – I still remember when my father brought the first one – a KC85/3 – home (in spring 1988). Last, but not least, I want to thank my wife Susanna for her unconditional support and patience – it really makes a difference if you sit in the same boat.

Contents

Part I

Introduction

Chapter 1

Co-Reservation of Resources in the Grid

In [SC92], Smarr and Catlett introduced the term *Meta Computing* for the transparent use of high-performance supercomputers distributed at different national laboratories in the US. Meta computing aims at the sharing of resources – hardware and software – and the collaboration of scientists spread over various geographical locations and different administrative domains. By taking an analogy from the *Power Grid*, which provides power as an *Utility* – nearly everywhere and everytime – Foster and Kesselman pioneered the term *Grid Computing* [FK99]. According to Foster [Fos02] a Grid is concerned with:

- *autonomously managed resources,*
- *standards, protocols and interfaces to uniformly access resources,* and
- *coordinating the provision of non-trivial quality-of-service.*

Software frameworks such as the Globus Toolkit [GT] and Unicore [Uni08] virtualize the interfaces of the resources and thus provide a uniform access to them. They, however, do not **coordinate** the execution of applications which range from atomic activities – a single data transfer, a compute job, etc. – to complex scenarios containing multiple activities spanning distributed resources. Coordination is important for all stakeholders – users, resource providers and virtual organizations (VO) – particularly to balance their often conflicting interests. Users are mainly interested in obtaining **guarantees** on the quality-of-service (QoS) for the execution of their applications. Providers aim at a **high utilization** of their resources. Virtual organizations want to ensure a **fair sharing** of the resources among their members – the users – to support their mission, e.g., analyzing the data of large scientific experiments, enabling world-wide collaborations, etc. Common QoS parameters are the response time of activities, the allocated network bandwidth or the cost for using a resource. Desiring guarantees is not unique to Grid computing. For example, in the travel business it is common practice

to obtain reservations for seats of a flight, hotel rooms, car rentals, etc. Such guarantees are particularly useful if the application (travel) is composed of multiple activities serviced by autonomous resources.

> *This thesis proposes CORES – a system architecture and mechanisms for obtaining guarantees on the execution of complex applications in Grid environments.*

1.1 Obtaining Guarantees in Grid Environments

In the first development phase of Grid computing (1995-2005), the focus was on providing a uniform interface to the multitude of resources. During that phase, only very basic coordination mechanisms were employed. These mechanisms applied greedy algorithms for balancing the workload and ensured the control flow of workflows, but did not provide efficient means for guaranteeing QoS parameters. Examples of such mechanisms include the Globus Toolkit component DUROC [CFK99] and the legendary phone call method, i.e., a user asks a resource owner to drain its system in order to let the user's jobs start at some agreed time. Obtaining guarantees on QoS parameters faces two big challenges in Grids, namely, the **autonomy of the resources** and the **lack of global information**. The former results in having limited control over when an activity is actually started by a resource's local management system. The latter does not only concern the problem of having incomplete information on the current state of the resources, but also the intractability to compute every possible schedule depending on future events such as the entry of new activities or the (unexpected) termination of existing ones.

1.2 Goals of a Resource Management System for Obtaining Guarantees

A resource management system for obtaining guarantees on QoS parameters is concerned with questions like

Q1 – *How can users efficiently and easily describe their applications and the required quality-of-service?*

Q2 – *How can the resources determine their availability, but retain as much autonomy as possible at the same time?*

Q3 – *How are candidates for reservations selected in order to satisfy the goals of both the users and the providers?*

Q4 – *How are reservations actually obtained?*

Q5 – *How is a co-reservation system embedded into Grid resource management?*

The main stakeholders in Grid resource management – the users and the providers – aim at different often conflicting goals. For example, users want to pay the lowest price, while providers wish to maximize their profit.

Goals of the Users. Users are mainly concerned with the description of their application scenarios. The description language must be easy to understand and use, yet it must be expressive to support the wide range of scenarios and it must be flexible to support new application scenarios. Notable features of such a description language, are capabilities to express moldable application parts as well as flexible means to specify temporal and spatial relationships between any two pairs of parts and, last but not least, the ability to define criteria for selecting the "best" solution. Apart from these goals of the description of scenarios, users are interested in an efficient processing and high success rate of their requests.

Goals of the Providers. Providers are mainly interested in retaining as much as possible their autonomy and in minimizing side-effects such as decreased utilization and degraded quality-of-service of their background workload. Thus, they require facilities for properly and efficiently advertising their offers – when and at which conditions resources may be reserved – and means to describe their constraints and criteria for accepting requests. In addition, enhanced mechanisms are needed for efficiently and reliably managing the actual allocation of resources to multiple requests.

1.3 General Approaches for Obtaining Guarantees

Depending on the resource management architecture and the goals of the stakeholders different approaches exist for providing guarantees on QoS parameters. First, a central scheduler could serve **all** activities by maintaining a global allocation table. This approach would not only violate the **autonomy of the resources**, but would not scale very well. Second, resources are allocated in **best-effort** manner. That is, an application scheduler submits sub-activities to eligible resources, but the resources' local management systems decide when these activities are executed. The application scheduler may use prediction mechanisms or activity replication to improve the level of guarantee. This approach suffers from the **lack of global information**, i.e., the desired QoS can only be met with a large overhead if at all. Third, the application scheduler and the local resource management systems collaborate by supporting priorities of activities and by pre-empting activities with low priorities to meet the guarantees of highly prioritized activities [BHL+06]. This approach provides a means to trade-off the level of autonomy with the level of guarantee. Fourth, the application scheduler reserves sufficient capacity at the desired resources in advance. In this approach, the local resource management systems retain the autonomy, except that they agree to some schedule decisions in advance and guarantee to abide by them. Fifth, the local resource management systems bilaterally negotiate an efficient execution of complex distributed ap-

plications. While this scenario seems well suited to cope with the two main challenges of Grid computing, it requires non-trivial changes of the scheduling mechanisms at the local resource management systems.

1.4 The Approach of CORES

The CORES framework follows the fourth approach – **reserving resources in advance** – to obtain guarantees on QoS parameters for the execution of complex applications in Grid environments. The approach was chosen, because it balances the goals of the resource providers – autonomy and information hiding – and the goals of the users – level of guarantees on the QoS – best.

CORES provides a generic system architecture for reserving multiple resources in advance. The fundamental assumptions in CORES are that

- *local resource management systems provide means for reserving capacity in advance,* and
- *the means for deriving the future status of resources are available or may be added to the resources' local management system.*

CORES's reservation mechanism contains the following steps.

1. The requests and resources are described.
2. Eligible resources are determined by matching static requirements and properties of the requests and resources.
3. The future status of the resources is calculated by the resources' local management systems.
4. Appropriate sets of candidates are selected for reservation.
5. The resources are actually allocated.

In CORES, the input to the resource reservation algorithms are **flexible** reservation requests. That is, a request specifies ranges of the parameters *start time*, *end time* and *service level*, and a utility function. The reservation system selects the best combination of parameter values to assign requests to resources with a start time, end time and a service level.

1.5 State of the Art

We briefly highlight the most prominent work related to reserving resources in advance. Each chapter contains a detailed analysis of the state of the art.

Over the last ten years, mechanisms to reserve resources in advance have gained more and more attention in the research on Grid computing. First, a generic framework [FKL+99] was depicted by Foster et al. Thereafter, the attention has shifted to address the questions **Q1** – **Q5** (cf. Section 1.2) in more detail.

Raman [RLS98] (Condor ClassAds), Stokes-Rees [SR06] (GRDL), and Wolf [Wol07] (D-GRDL) proposed languages for describing requests and resources (**Q1**).

Methods for predicting future values of key parameters of a resource management system (**Q2**) such as the execution time of jobs or the waiting time of jobs are studied by many authors, i.e., Andrzejak et al. [AC05], Downey [Dow97], Li et al. [LGTW04, Li07], Röblitz et al. [RSR06, RR06] and Smith et al. [SFT98, STF99].

The selection of candidates (**Q3**) was studied in several proposals, e.g., Brandic et al. [BBES05], Naik et al. [NLYW05], Röblitz [Röb08b, Röb08a] and Zeng et al. [ZBN+04].

Methods for actually obtaining reservations (**Q4**) are closely related to transactions in distributed database systems (e.g., SAGAS [GMS87], flexible transactions [ELLR90], etc.) and the composition of web services (e.g., WS-BusinessActivity [FL07]), and have seen little improvements in the context of Grid resource management. The resource independent protocol SNAP for reserving Grid resources was introduced by Czajkowski et al. [CFK+02]. The OGF standard WS-Agreement [ACD+07] defines a protocol for negotiating service level agreements between consumers (users) and providers. MacLaren proposed HARC [Mac07] which puts emphasis on handling failures in the context of reserving multiple Grid resources.

Several proposals were made for embedding reservation systems into Grid resource management systems (**Q5**). In [RR05], we proposed to consider co-reservations as *Virtual Resources*, which may provide enhanced management capabilities such as external scheduling, resource aggregation and fault recovery.

1.6 Organization of the Thesis

The thesis is organized in three parts. The first part contains

- a description of the application scenarios (Chapter 2),
- an overview of three approaches for guaranteeing quality-of-service (Chapter 3),
- a summary of performance models (Chapter 4), and
- a mathematical formulation of the co-reservation problem (Chapter 5).

The main part provides

- a system architecture and life cycle for managing co-reservations (Chapter 6),
- a language for describing requests and resources (Chapter 7),
- an analysis of existing mechanisms for matching static requirements (Chapter 8),

- a versatile method for determining the future state of resources (Chapter 9),
- integer and binary models for mapping requests to resources (Chapter 10),
- a study of efficient methods for allocating multiple resources (Chapter 11), and
- a model for embedding co-reservations into Grid resource management (Chapter 12).

The third part concludes the thesis (Chapter 13) and lists terminology (Appendix A).

Chapter 2

Application Scenarios

We describe two application scenarios – *parallel jobs* and *job chains* – which are in the focus of this thesis. The applications shall be executed on a distributed infrastructure spanning multiple autonomous domains – the Grid. First, we emphasize the essential properties of the scenarios and reference actual projects where the scenarios frequently appear (cf. Section 2.1). Thereafter, we introduce several extensions which apply to both application scenarios (cf. Section 2.2).

2.1 Basic Scenarios

For each application type – parallel jobs and job chains – we present basic considerations of both the users and the resource providers. The main difference between the two application types is their **temporal** structure. Parallel jobs require that (most) parts are executed in parallel, while the parts of job chains are executed in a sequence.

BS1 – Parallel Jobs. The most basic parallel job (cf. Example 2.1) requires two compute resources and a network connection between them. At each resource, the application needs a certain software and hardware environment, in which it executes with known characteristics. The most important hardware aspects are the architecture of the processor, its clock frequency as well as the required main memory and local disk space. The software environment contains the names and versions of the operating system, libraries and software packages. The execution characteristics are given by simple estimates of the runtime for a fixed number of processors. The network connection between the compute parts must feature a minimum bandwidth and a maximum latency to allow an efficient communication between the application parts. The resource providers (not shown in Example 2.1) may grant or deny access based on the affiliation of the user, the requested capacity or (virtually) any policy.

Example 2.1 (Collisions of Black Holes)
Run two instances of the Cactus Code *[ADF+01, GAL+03, Cac06], each at a Linux cluster (kernel version 2.6.1 or higher) with 128 x86-processors, two gigabytes of main memory per processor and 50 gigabytes of local disk space. Both instances require 48 hours execution time. The efficient execution requires a network connection between the two sites with a maximum latency of 50 milliseconds and a minimum bandwidth of 1 Gbit/s.*

A similar application is the simulation of gravitational forces [Aar03, Spr05, Gad06] in the astrophysics community. Co-executing multiple application parts – making archived data available, accessing data from a program, using licenses of a 3^{rd} party software package, etc. – at the same time is a generalization of the parallel job type.

BS2 – Job Chains. The simplest job chain just requires a single compute part, a data source and a network connection for transferring the data from the source to the compute site (cf. Example 2.2). The compute part may be described as those of the parallel jobs. The data part defines a logical file name and its size. In contrast to parallel jobs, the parts are executed in sequence. First, data is restored from an archive system, then transferred to a compute site, where it will be processed. A small extension would involve multiple data processing steps and data transfers in between them, all of which are executed sequentially.

The resource providers (not shown in Example 2.2) may grant or deny access based on the affiliation of the user, the ownership of and permissions to read the data, the requested capacity or (virtually) any policy.

Example 2.2 (Stormtrack Analysis in Climate Research)
The analysis of stormtracks (cf. case study in [GLPS07]) is composed of four steps. First, raw data is staged from archives and filtered to extract information about geopotential heights. Second, the extracted information – depending on the field size and temporal resolution up to tens of GB – is transferred to the compute resource. Third, the stormtrack is calculated by a single sequential application. Last, the results (from a few KB to hundreds of MB) are transferred to a server specified by the user. The analysis shall be finished no later than 2008-03-04 06:00 PM UTC.

Job chains occur in many disciplines, for example, the post-processing of data obtained from LHC experiments in high-energy physics [FGPS07], the detection of gravitational waves in astrophysics [ESR+07], image processing in medical sciences [KPS+07] or the continuous analysis of data recorded by sensor networks in earth science research [KLHW07]. Note, we aim at static job chains, i.e., those whose control-flow is known a-priori.

2.2 Advanced Scenarios

The advanced scenarios extend the basic scenarios in the following seven aspects.

AS1 – Types of Applications. The scenario of Example 2.3 differs from the basic ones, because it does not require all resources at the *same time* nor in a *specific order*, but it needs a certain capacity within a given period of time. The needed capacity may be aggregated from many sites, hence a co-reservation may be used. Using virtualization mechanisms, the aggregated resources can be made accessible as if there were provided at a single site.

> **Example 2.3 (Parameter Sweep Study)**
> *For a parameter sweep study 1 million short (10 minutes) single processor jobs need to be executed until* 2008-04-17 08:00 AM UTC*, but the study cannot start before* 2008-04-14 06:00 PM UTC*.*

AS2 – Types of Resources. In addition to compute resources, data servers and network connections, applications may require the availability of software licenses (cf. Example 2.4) and special purpose machines for visualizing the simulated or analyzed data. Also, other resources such as web servers, database servers or custom application servers may be integrated into the scenarios.

> **Example 2.4 (Computational Fluid Dynamics)**
> *A step of a job chain uses the* Fluent *package thus requiring a license during its execution.*

AS3 – Number of Resources. The number of needed resources increases with additional types of resources, demands for a larger aggregated capacity or longer job chains.

> **Example 2.5 (Visualization of simulation results)**
> *The simulation results (cf. Example 2.1) shall be visualized at the* Studio da Vinci *at Zuse Institute Berlin.*

AS4 – Richer Means to Describe Application Characteristics. A user may provide more information on the characteristics of the whole application and its parts. Example 2.6 illustrates the relation of the application's execution time and the allocated capacity.

> **Example 2.6 (Moldable Parallel Computation)**
> *The parallel application of Example 2.1 requires 64 hours on 64 processors, 96 hours on 32 processors, 160 hours on 16 processors and so forth. In this example, the speed-up of the application complies to Amdahl's law with the parameters* $s = \frac{1}{65}$ *and* $p = \frac{64}{65}$ *(cf. Section 4.1.1).*

AS5 – Arbitrary Constraints and Objectives. Constraints and objectives may guide the assignment of simulation parts to resources in a flexible way, particularly, they may be specified on any subset of the individual parts of an application. For example, users are limited by an available budget (constraint on all parts) and wish to minimize the

total costs[1] (objective on all parts) as well as the end time of the whole application (objective on all parts or the 'last') *or* to trade-off end time with costs. On the other side, resource providers are interested in high utilization and maximizing their profit (objectives on a single part).

> **Example 2.7 (Cost Limit, Overlapping Tasks & Minimal Cost)**
> *The costs for executing the stormtrack analysis (cf. Example 2.2) may not exceed 200 Euro. Also, the first and the second step as well as the third and the forth step may (partially) overlap, that is, data transfers may start as soon as data is available. Finally, the costs shall be minimized if multiple schedules satisfying the budget constraint exist.*

AS6 – Types of Agreements. Depending on the requests and the capabilities of the resource management systems, different levels of guarantees on the allocation of the resources may be needed. For example, merely transferring small files between workflow steps may be executed in best-effort manner, planning a demo (cf. Example 2.8) requires a firm guarantee that the resource will be available, applications involving business processes may require compensation if the desired service level is not provisioned, etc.

> **Example 2.8 (High Level of Allocation Guarantee)**
> *The additional visualization step (cf. Example 2.5) must happen from* 2008-06-23 10:00 AM UTC *to* 2008-06-23 12:00 AM UTC, *because a demo is planned for that time.*

[1] Costs can be monetary or expressed in equivalents of processing hours.

Chapter 3

Coordination Approaches in Grid Resource Management

The application scenarios presented in Chapter 2 require the coordinated use of resources in a Grid. First, we give a brief overview of the resource management in existing Grid environments (cf. Section 3.1). Thereafter, we present the three approaches which build upon the capabilities of the existing resource management functions. In particular, we study what can be achieved with best effort co-allocation (cf. Section 3.2), we outline a scheme based on advance co-reservations (cf. Section 3.3) and we sketch a mechanism for peer-to-peer co-scheduling (cf. Section 3.4).

3.1 Resource Management in Grid Environments

Figure 3.1 shows the levels of resource management in existing Grid environments. On top of the physical resources, operating systems such as UNIX, Linux, Windows, etc., provide basic capabilities, i.e., process creation, process isolation, scheduling of processes, memory management, disk quota enforcement, access to network devices and so forth. At the next level, local resource management systems control the admission of activities to pools of physical resources. Grid-level resource management builds upon local resource management by introducing a virtualization layer (Grid-Middleware) and, at the top level, a coordination layer (Grid-Broker).

3.1.1 Machine-level Resource Management

The overall goal of machine-level resource management is to allow access by multiple users and the simultaneous execution of multiple programs using the physical devices of a machine. Therefore, operating systems provide means to **create** processes, files and connections, to **control** and **schedule** processes, to **read** and **write** data to files and connections and to **delete** processes, files and connections. Furthermore, the operating system prevents unauthorized access to data in files and data in the memory

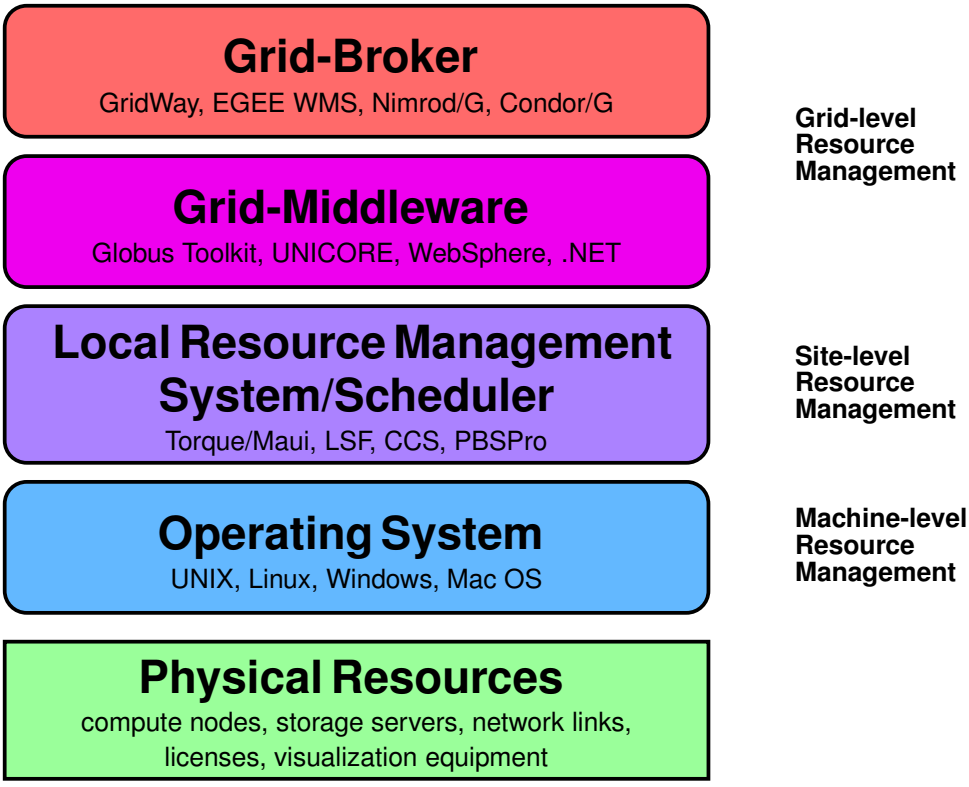

Figure 3.1: Layered resource management in existing Grid environments.

of processes. At the machine-level, operating systems are typically executed as single system images providing a uniform view of the physical devices. In a multi-user and multi-tasking environment, operating systems trade-off the fairness among the users and high utilization. For example, if disk usage quota cannot be enforced, each user/group may be assigned a single device to guarantee their requirements. Similarly, if the maximum parallelism of a session (by creating new processes or threads) cannot be controlled, each compute job is assigned a full node. The usage quota enforcement capabilities of existing operating systems may be extended by virtualization layers such as Xen [Xen08] or VMware [Vmw08].

3.1.2 Site-level Resource Management

Often single machines, belonging to the same site, the same research or business group, or the same administrative domain, are pooled together to employ a uniform admission control policy and enable fair sharing of them. For example, compute nodes may be aggregated in clusters managed by well-known tools such as Condor [LLM88], Torque/Maui [JSC01], PBSPro [PBS04], LoadLeveler [Loa08], Sun Grid Engine [Sge08], CCS [KR98], OAR [CCG+05]. All of these systems provide a similar set of core functions, but some support enhanced management capabilities, too. Core functions include: monitoring the status of machines, providing a single entry point for activities

(e.g., compute jobs), queuing and scheduling of workload, priority-based ordering of activities, supporting heterogeneous machines, management of job steps, job dependencies as well as the setup of job environments, monitoring the status of jobs and accounting of resource usage. On compute resources, often first-come-first-served (FCFS) together with backfilling are used as scheduling policy. Enhanced management capabilities are job check-pointing and advance reservation. Typically, activities can be submitted by any user from the front-end of a resource. Each site, however, autonomously decides when these activities are executed. In particular, many resource management systems make that decision when resources are available, i.e., they do not calculate a schedule in advance. Reservations provide a means to make that decision in advance and provide a guarantee that the associated activities may start at the desired time.

3.1.3 Grid-level Resource Management

Grid-level resource management is split into two layers – the Grid-Middleware and the Grid-Broker (cf. Fig. 3.1).

The Grid-Middleware serves as an abstraction layer virtualizing site-level resource management systems. Grid-Middlewares such as the Globus Toolkit [GLO08] and Unicore [Uni08] provide generic means for authentication and authorization; submission, monitoring and control of activities; transferring data (files) and accessing information about the status of resources. The information about the status of resources covers, however, mainly static properties (cf. schemata such as GLUE [GLU08]). More importantly, information about the past and current workload may be hidden, incomplete or simply outdated. Grid-Middleware also supports a minimal layout of a generic sandbox for executing compute jobs. For example, with Globus, the expression `${GLOBUS_USER_HOME}` allows the user to prepare the job environment independent of its actual account name and path of the user's home directory.

Grid-Brokers determine eligible resources for distributing the workload to sites. Rather than calculating a schedule, workload management in todays Grid environments is greedy as it only considers one activity request at a time. Enhancing workload management not just requires to update existing scheduling policies, but to deal with the lack of global knowledge about the local scheduling policies, to gather detailed information about the current and future status of the resources, and to extend resource sharing models towards economic approaches. Besides workload distribution, Grid-Brokers also take care of preparing the job environment on remote sites, e.g., by initiating file transfers and delegating credentials, and provide basic fault-recovery, e.g., by resubmitting jobs to other sites if they failed elsewhere.

3.2 Best Effort Co-Allocation

The best effort co-allocation approach just uses the existing capabilities of Grid-enabled site-level resource management and the available status information. In particular, we make the following assumptions.

- Activities can be submitted to any resource, but the resource's local management system decides when to start the activity.
- Sites publish their currently free capacity, their total capacity and the number of queued activities.
- A Grid-Broker possesses detailed information about activities only if it has submitted these activities by itself.

Given the above assumptions, we study if and how well the scenarios (cf. Chapter 2) can be implemented.

The basic parallel application scenario (BS1) – limited to two compute resources only – can be implemented by means such as DUROC [CFK99] or KOALA [ME05]. DUROC submits two parts, one to each compute site. Once started each part synchronizes with the other part at a barrier. While the first part waits for the second part, no other job may use the blocked resources. KOALA aims to minimize such waiting by placing the parts at sites with the largest number of idle processors and may preempt low-priority jobs if not enough processors are available at the predicted start time. This approach, however, requires changes to the local scheduling policies.

We assess the quality of best effort co-allocation by using results from advanced prediction methods. In [Li07], Hui proposed a machine learning approach for predicting the queue waiting time of two different compute resources and three workload traces NIK04, SDSC01 and SDSC02. For the sake of simplicity, we assume that the errors of the predicted times are normally distributed with the means μ of 300 (NIK04), 375 (SDSC01) and 690 (SDSC02) minutes as well as the variances σ^2 of 225 (NIK04), 325 (SDSC01) and 490 (SDSC02). We performed three experiments to sample the distribution of the absolute difference between the prediction errors. The resulting distributions have the means 75 (NIK04/SDSC01), 390 (NIK04/SDSC02) and 315 (SDSC01/SDSC02). We can expect that resources are wasted at the site starting first for duration of the difference of the prediction error. The more processors a part requests, the larger is the effective loss in throughput at the site starting first. A similar analysis can be performed for parallel applications requiring more than two parts (AS3) and for job chains (BS2). In certain circumstances, prediction-based coordination may be sufficient, for example, meeting deadlines can be relatively easy if the deadline is well behind the predicted response time. Best effort co-allocation may also cope with larger numbers of resources (AS3), different types of resources (AS2) and applications (AS1). On the other hand, they are not well suited to handle richer means for describing application characteristics (AS4), are not able to satisfy arbitrary constraints and objectives (AS5), particularly, economic ones and may not support different types of agreements well (AS6).

3.3 Advance Co-Reservation

The co-reservation approach requires that the site-level resource management systems allow to reserve fractions of their capacity for a specified period of time. In practice, this period should begin at some time in the future. Using this capability, a central co-reservation service may request reservations such that the relationships between any two pairs of application parts are satisfied. By gathering additional information about the future status of the resources, especially their availability and costs, the co-reservation service may find reservation candidates which not only satisfy the constraints but also the objectives of all stakeholders. The disadvantages of the co-reservation approach are that the schedule must be fixed in advance and that the site-level resource management systems loose some autonomy in scheduling their workload. However, the SLRMSs retain the full autonomy on granting or denying requests for advance reservations. The advance co-reservation approach serves all aspects of the application scenarios very well.

3.4 Peer-to-Peer Co-Scheduling

In the P2P co-scheduling approach, the start times of the application parts are explicitly coordinated between any pair of site-level resource management systems (SLRMS). That is, the coordination is fully decentralized. Since each SLRMS is aware of the structure of a complex application – at least it knows which parts are directly dependent of the part the SLRMS manages – it needs to negotiate the start time of its own part with the SLRMSs of the adjacent parts. Depending on the inherent flexibility of the application and the flexibility in the scheduling of the current workload of an SLRMS such negotiations may span all SLRMSs managing a complex application. That is, P2P co-scheduling requires a distributed consensus protocol. However, the consensus protocol may not simply employ a majority scheme since the minority may not be able to implement the majority decision. Moreover, finding an optimal schedule considering economic metrics may only be possible if the values of these metrics are known a-priori. Finally, implementing the decisions of the consensus protocol may require changes of each site-level resource management system. The advantage of P2P co-scheduling is that it does not require reservations unless the application must be executed during specific periods of time. Thus, coordinating complex applications by P2P co-scheduling may yield smaller impacts on other applications than achieved with a co-reservation scheme.

3.5 Summary

Table 3.1 provides a high-level overview of the approaches, comparing them with respect to the implementation of the application scenarios. In this work, we will exploit the advance co-reservation approach, because basic advance reservation capabilities

are supported by many site-level resource management systems and the approach allows a wider range of scenarios. In specific environments, P2P co-scheduling or best effort co-allocation may be sufficient tough.

Table 3.1: Comparison of the approaches for supporting the complex application scenarios described in Chapter 2.

	Approaches		
Phase	best effort co-allocation	advance co-reservation	peer-to-peer co-scheduling
submit	centrally steered, wait time prediction	centrally steered, status probing	no coordination
start	no coordination	no coordination	explicit, decentralized coordination
runtime	explicit, decentralized coordination	no coordination	no coordination

Chapter 4

Activity Performance Models

We introduce performance models of compute jobs depending on the number of processors (cf. Section 4.1) and the type of the processors (cf. Section 4.2).

4.1 Speed-up Models of Parallel Programs

We present two performance models of parallel applications – Amdahl and Downey – and provide several definitions for calculating the speed-up, the resulting execution time and the required number of processors to achieve a specific speed-up or execution time.

4.1.1 Amdahl's Law

By noticing that the runtime of any parallel program cannot be shorter than the time needed for executing its sequential part – no matter how many processors one might employ – Amdahl formulated his famous speed-up model [Amd67]

$$S_A(n, s, p) = \frac{s + p}{s + \frac{p}{n}}, \tag{4.1}$$

where $n \in \mathbb{N}, n > 0$ is the number of processors, s is the sequential and p is the parallel fraction of the program. Thus, $s + p = 1$ holds for any program. Figure 4.1 illustrates S_A for various parallel fractions and numbers of processors.

The inverse function $S_A^{-1}(sup, s, p)$ calculates the number of processors needed to obtain a given speed-up sup. If the condition $\frac{1}{sup} > s$ holds, the inverse function is defined by Eq. (4.2).

$$S_A^{-1}(sup, s, p) = \frac{p}{\frac{1}{sup} - s}. \tag{4.2}$$

Given a reference execution time dur_{ref} and a reference number of processors np_{ref}, the execution time on n processors is defined as

$$dur(n, s, p, np_{ref}) = dur_{ref} \frac{S_A(np_{ref}, s, p)}{S_A(n, s, p)}. \tag{4.3}$$

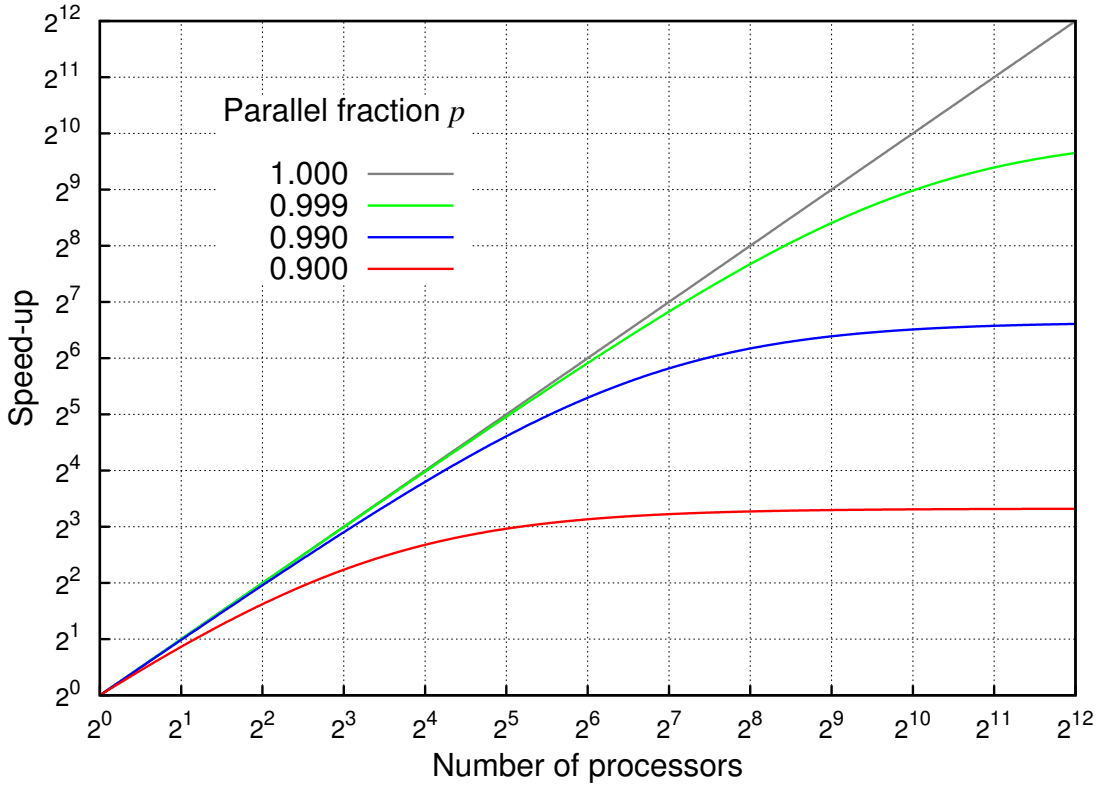

Figure 4.1: Amdahl's law for various parallel fractions p and numbers of processors.

4.1.2 Downey's Performance Model

Amdahl's model has been criticized for being too simple and therefore unusable in practice. When the average degree of parallelism A and its variance σ are known, Downey's speed-up model (cf. Eq. (4.4)) provides a better speedup estimate for a program running on $n \in \mathbb{N}_+$ processors.

$$S_D(n) = \begin{cases} \frac{An}{A+\sigma/2(n-1)} & \sigma \leq 1 \quad \wedge \quad 1 \leq n \leq A \\ \frac{An}{\sigma(A-1/2)+n(1-\sigma/2)} & \sigma \leq 1 \quad \wedge \quad A \leq n \leq 2A-1 \\ A & \sigma \leq 1 \quad \wedge \quad n \geq 2A-1 \\ \frac{An(\sigma+1)}{\sigma(n+A-1)+A} & \sigma \geq 1 \quad \wedge \quad 1 \leq n \leq A+A\sigma-\sigma \\ A & \sigma \geq 1 \quad \wedge \quad n \geq A+A\sigma-\sigma \end{cases} \tag{4.4}$$

4.1.3 Miscellaneous Speed-Up Models

Besides Amdahl and Downey many more speed-up or performance models exist. We briefly list a few of them.

Gustafson [Gus88] adjusts Amdahl's law by the observation that the achievable speed-up may also depend on the execution time of the application. Based on the

observation that any parallel program contains a computation and a communication part, Gruber et al. propose the Γ-model [GVV+03]. The Γ-model characterizes parallel applications by the quotient of the computation part and the communication part and the machines by the quotient of their maximum local processor performance and the per processor network communication bandwidth. Gruber et al. argue that all these parameters can be easily obtained. Thus, compute clusters can be tailored to the applications. In the "database" performance model, the execution time and speed-up of a parallel application is derived by evaluating workload traces. Those workload traces must comprise information about the executable (name and version), the number of used processors, the network between the processors, the application parameters, and so forth.

4.2 Processors' Performance in a Grid

In a Grid, compute resources may have different types or versions of a processor installed. Thus, the required execution time of a program may be adjusted by the broker beforehand. For the sake of simplicity, we assume the performance of a processor is characterized by a single metric. The higher is the metric's value, the higher is the processor's performance. Given a reference execution time dur_{ref} on a reference processor with the performance pp_{ref} and the performance pp of a target processor, a sequential program requires the execution time $dur^{\circ}(pp)$ (cf. Eq. (4.5)).

$$dur^{\circ}(pp) = dur_{ref}\,\frac{pp_{ref}}{pp} \tag{4.5}$$

For parallel programs, we integrated the heterogeneity of processors in a Grid into Amdahl's law [Amd67] by combining Equations (4.1) and (4.5). First, we adapted the speed-up S_A defined in Eq. (4.1) to take the performances of the target processor pp and the reference processor pp_{ref} into account. The adapted speed-up $S_A^{\star}$ is defined in Eq. (4.6).

$$S_A^{\star}(n, s, p, pp, pp_{ref}) = \frac{1}{s + \frac{p}{n}}\,\frac{pp}{pp_{ref}} \tag{4.6}$$

The inverse function ${S_A^{\star}}^{-1}$ calculates the number of processors needed to obtain a given speed-up sup. If the condition $\frac{pp}{sup \cdot pp_{ref}} > s$ holds, the inverse function is defined by Eq. (4.7).

$${S_A^{\star}}^{-1}(sup, s, p, pp, pp_{ref}) = \frac{p}{\frac{pp}{sup \cdot pp_{ref}} - s} \tag{4.7}$$

Figure 4.2 illustrates Eq. (4.6). The execution time $dur^{\star}$ of a parallel program is calculated as the product of the reference execution time dur_{ref} and the ratio of the original speed-up S_A to the adapted speed-up $S_A^{\star}$. The adapted duration is defined by Eq. (4.8).

$$dur^{\star}(n, s, p, pp, np_{ref}, pp_{ref}, dur_{ref}) = dur_{ref}\,\frac{S_A(np_{ref}, s, p)}{S_A^{\star}(n, s, p, pp, pp_{ref})} \tag{4.8}$$

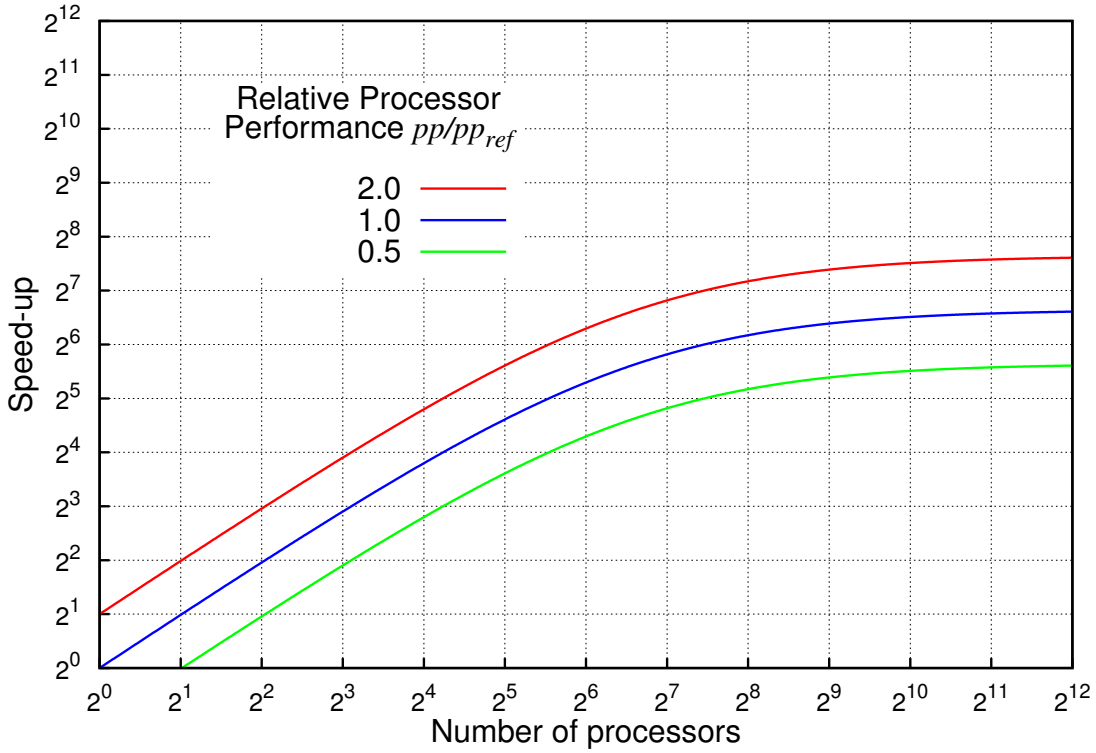

Figure 4.2: Amdahl's law in a Grid scenario for compute resources with different relative processor performances pp/pp_{ref} and an application with a parallel fraction $p = 0.99$.

Finally, Eq. (4.9) defines the inverse function of $dur^{\star}$, which calculates the number of processors needed to execute an application within a given duration dur.

$$dur^{\star -1}(dur, s, p, pp, np_{ref}, pp_{ref}, dur_{ref}) = S_A^{\star -1}\left(\frac{dur_{ref} \cdot S_A(np_{ref}, s, p)}{dur}, n, s, p, pp, pp_{ref}\right) \tag{4.9}$$

Chapter 5

Mathematical Formalization of the Co-Reservation Problem

The co-reservation problem $CoRP$ of assigning partially related requests to resources is described as abstract optimization problem. First, the sets of requests and resources are introduced. Then, we define the variables and their domains. Next, we present a core concept exploited in CORES – the properties of requests and resources. Then, we use these properties in several types of constraints: single assignment constraints, multi assignments constraints, and temporal and spatial relationships. Finally, we define the criteria for finding optimal solutions.

5.1 Problem Notation

We denote the co-reservation problem $CoRP$ by the set of problem instances $\{CoRP_i\}$. Each instance is a tuple $\langle R_i, S_i, T_i, D_i, Q_i, SIC_i, P_i, SAC_i, TR_i, SR_i, MAC_i, O_i\rangle$ of sets, which are empty by default. In the failure free case, the set $CoRP$ contains a single element only. Additional elements may be created through refining the original problem instance iteratively (cf. Section 10.6). A solution is found by solving all instances (elements of $CoRP$) individually and selecting the solution of the instance with the "best" objective value as global solution. Note, in the following of this chapter we will omit the subscript i.

5.2 Requests, Resources, Variables and Domains

Definition 1 (Requests and Resources). *The sets $R = \{r_1, \ldots, r_L\}$ ($L \in \mathbb{N}$, $r_l \in \mathbb{N}$) and $S = \{s_1, \ldots, s_K\}$ ($K \in \mathbb{N}$, $s_k \in \mathbb{N}_+$) denote the finite set of L requests and the finite set of K resources, respectively.* ♢

Definition 2 (Variables). *For each request $r \in R$, the variables $V_s(r)$ (resource), $V_t(r)$ (start time), $V_d(r)$ (duration) and $V_q(r)$ (service level) are searched for.* ♢

Definition 3 (Domain of $V_s(r)$). *The domain $\mathrm{dom}(V_s(r))$ of the variable $V_s(r)$ is the union of the sets $\{0\}$ and $S(r) \subseteq S$.* ♢

The set $\{0\}$ is used if the request is not assigned to any resource. The set $S(r)$ contains the **eligible** resources of the request r. That is, each pair (r, s) with $s \in S(r)$ satisfies the static requirements of both, the request r and the resource s.

Definition 4 (Domains of $V_t(r)$, $V_d(r)$ and $V_q(r)$). *The domains $\mathrm{dom}(V_t(r))$, $\mathrm{dom}(V_d(r))$ and $\mathrm{dom}(V_q(r))$ of the variables $V_t(r)$, $V_d(r)$ and $V_q(r)$, respectively, are defined as finite subsets of the natural numbers $\mathbb{N}$, i.e.,*

$$\forall r \in R: \quad \mathrm{dom}(V_t(r)) \subset \mathbb{N} \;\wedge\; \mathrm{dom}(V_d(r)) \subset \mathbb{N} \;\wedge\; \mathrm{dom}(V_q(r)) \subset \mathbb{N}\,. \qquad ♢$$

The sets T, D and Q denote the unions of the the domains of the start times, the durations and the service levels, respectively. They are formally defined as follows

$$T = \bigcup_{l \in [1,L]} \mathrm{dom}(V_t(r_l))\,, \quad D = \bigcup_{l \in [1,L]} \mathrm{dom}(V_d(r_l))\,, \quad Q = \bigcup_{l \in [1,L]} \mathrm{dom}(V_q(r_l))\,.$$

5.3 Assignments and their Combinations

Definition 5 (Assignment). *A tuple $\langle r, s, t, d, q\rangle$ with $r \in R$, $s \in S(r)$, $t \in \mathrm{dom}(V_t(r))$, $d \in \mathrm{dom}(V_d(r))$ and $q \in \mathrm{dom}(V_q(r))$ is called an assignment of the request r to the resource s at the start time t for the duration d with the service level q.* ♢

The expression $r \triangleright s$ denotes any assignment of the request r to the resource s. Any assignment involving the request r and the resource s are denoted by the term $r\triangleright$ and $\triangleright s$, respectively.

Definition 6 (Combination of Assignments). *We call the set $CA = \{\langle r_i, s_i, t_i, d_i, q_i\rangle\}$, which contains one element per request r_i, $i = 1, \ldots, L$, a combination of assignments.* ♢

A combination of assignments CA represents a solution candidate of the co-reservation problem.

Definition 7 (Mapped Combination of Assignments). *The set $CA^S \subseteq CA$ denotes the combination of assignments, which involves any resource in S, but the virtual resource 0, i.e.,*

$$CA^S = \{\langle r_i, s_i, t_i, d_i, q_i\rangle \,|\langle r_i, s_i, t_i, d_i, q_i\rangle \in CA \;\wedge\; s_i \neq 0\}\,. \qquad ♢$$

5.4 Properties

Before we formally define the constraints, we introduce properties as a means to describe the features of an entity in a given assignment. For example, the required duration of a request may depend on the service level (bandwidth, number of processors) offered in an assignment. Similarly, the requested reservation fee may depend on the

requester's affiliation or the offered service level. In a real system, properties are derived from the descriptions of the requests, the resources and – in particular – through the prediction of the future status (step ③ in Fig. 6.2 on page 38).

Definition 8 (Property). *A property $p^{id}_{x \triangleright y}$ is a mapping*

$$p^{id}_{x \triangleright y} : T \times D \times Q \longrightarrow \mathbb{R}\,,$$

with $id \in \mathbb{N}$ being an identifier and $x \triangleright y$ referring to an assignment or a set of assignments. ♢

In the following, we often use strings as identifiers, for example, stt for the start time of a reservation, $cost$ for its reservation fee, fit for its fitness in a resource's local schedule.

The sets $P_{r\triangleright}$ and $P_{\triangleright s}$ contain all properties of the request r and the resource s, respectively. The set of all properties of an assignment $r \triangleright s$, denoted by $P_{r\triangleright s}$, is defined as the union $P_{r\triangleright} \cup P_{\triangleright s}$. The set

$$P = \bigcup_{r_l \in R,\ s_k \in S(r_l)} P_{r_l \triangleright s_k}$$

contains all properties of all assignments. The parameters of an assignment $\langle r, s, t, d, q \rangle$ may be denoted by specific properties as well. For example, the start time of the request r may be defined through the property

$$p^{stt}_{r\triangleright}(t, d, q) = t\,.$$

Similarly, its duration can be defined as

$$p^{dur}_{r\triangleright}(t, d, q) = d\,.$$

5.5 Constraints

We define several types of constraints:

- *single assignment constraints*, which a single assignment must fulfill,
- *temporal relationships*, which must be fulfilled by pairs of assignments,
- *spatial relationships*, which must be fulfilled by two to three assignments, *and*
- *multi assignments constraints*, which restrict up to L assignments.

For all types (except spatial relationships), we distinguish *equality* and *inequality* constraints. Spatial relationships always use equality as relation. We denote equality and inequality constraints by the superscripts $^{=}$ and $^{\geq}$, respectively.

Definition 9 (Single Assignment Constraint). *A constraint on a single assignment $r \triangleright s$ is a mapping*

$$sac^{=,\geq}_{r\triangleright s,h} :\ T \times D \times Q \longrightarrow \mathbb{R}\,,$$

with $h \in \mathbb{N}$ enumerating the constraints of the assignment $r \triangleright s$. ♢

The actual mapping of a constraint on the assignment $r \triangleright s$ is constructed as some arbitrary combination of the properties in $P_{r \triangleright s}$. Common combinations are the weighted sum, euclidean norm or minimum and maximum.

Example 5.1 (Deadline of a Job Chain)
The result of a sequential 3-step job chain must be available no later than some deadline $p^{dl}_{r_3\triangleright}$. Under the assumption that request r_3 represents the third step, this requirement can be expressed as follows

$$sac^{\geq}_{r_3\triangleright,1}\,(t,d,q) \;=\; p^{dl}_{r_3\triangleright}(t,d,q) - p^{stt}_{r_3\triangleright}\,(t,d,q) - p^{dur}_{r_3\triangleright}\,(t,d,q)\;.$$

Definition 10 (Boolean Value of a Single Assignment Constraint). *The boolean value* $\mathsf{bool}()$ *of a constraint* $sac^{cop}_{r\triangleright s}$ *is defined as follows*

$$\mathsf{bool}\,(sac^{cop}_{r\triangleright s}(t,d,q)) = \begin{cases} 1 & cop \text{ equals } '=' \;\wedge\; sac^{=}_{r\triangleright s}(t,d,q) = 0\,, \\ 1 & cop \text{ equals } '\geq' \;\wedge\; sac^{\geq}_{r\triangleright s}(t,d,q) \geq 0\,, \\ 0 & \text{elsewise}\,. \end{cases}$$

♢

The set of all constraints on the single assignment $r \triangleright s$ is denoted by $SAC_{r\triangleright s}$. The set

$$SAC = \bigcup_{r_l\in R,\; s_k\in S(r_l)} SAC_{r_l\triangleright s_k}$$

contains all single assignment constraints of all assignments.

We model **temporal relationships** between two assignments $r_a \triangleright s_a$ (short A) and $r_b \triangleright s_b$ (short B) by special equality and inequality constraints.

Definition 11 (Temporal Relationship). *A temporal relationship between two assignments A and B is a mapping*

$$tr^{=,\geq}_{A:B,h} : (T \times D \times Q)^2 \longrightarrow \mathbb{R}\,,$$

with $h \in \mathbb{N}$ enumerating the temporal relationships between the assignments A and B. ♢

The actual mapping of a temporal relationship $tr_{A:B}$ is constructed as some arbitrary combination of the (temporal) properties in $P_A \cup P_B$.

Example 5.2 (Precedence Relations of a Job Chain)
The first step (request r_1) of the above sequential 3-step job chain (cf. Example 5.1) shall precede the second step (request r_2). This requirement can be expressed as follows

$$\begin{aligned} tr^{\geq}_{r_1\triangleright:r_2\triangleright,1}\,(t_1,d_1,q_1,t_2,d_2,q_2) \quad = \quad & p^{stt}_{r_2\triangleright}\,(t_1,d_1,q_1,t_2,d_2,q_2) - \\ & p^{stt}_{r_1\triangleright}\,(t_1,d_1,q_1,t_2,d_2,q_2) - \\ & p^{dur}_{r_1\triangleright}\,(t_1,d_1,q_1,t_2,d_2,q_2)\;. \end{aligned}$$

Definition 12 (Boolean Value of a Temporal Relationship). *The boolean value* $\mathsf{bool}()$ *of a temporal constraint* $tr^{cop}_{A:B,x}$ *is defined as follows*

$$\mathsf{bool}\left(tr^{cop}_{A:B,x}(t_a, d_a, q_a, t_b, d_b, q_b)\right) = \begin{cases} 1 & cop \text{ equals } '=' \ \wedge\, tr^{=}_{A:B,x}(t_a, d_a, q_a, t_b, d_b, q_b) = 0\,, \\ 1 & cop \text{ equals } '\geq' \ \wedge\, tr^{\geq}_{A:B,x}(t_a, d_a, q_a, t_b, d_b, q_b) \geq 0\,, \\ 0 & \text{elsewise}\,. \end{cases}$$

♢

The sets $TR^{=}_{A:B}$ and $TR^{\geq}_{A:B}$, respectively, contain all temporal equality and inequality constraints between the assignments A and B. The set of all temporal constraints between the assignments A and B is denoted by $TR_{A:B} = TR^{=}_{A:B} \cup TR^{\geq}_{A:B}$. The set

$$TR = \bigcup_{\substack{r_{l1} \in R, \\ s_{k1} \in S(r_{l1})}} \; \bigcup_{\substack{r_{l2} \in R, \\ s_{k2} \in S(r_{l2})}} TR_{r_{l1} \triangleright s_{k1} : r_{l2} \triangleright s_{k2}}$$

contains all temporal relationships between any two assignments.

Spatial relationships are used to co-locate two requests at the same site, e.g., input data with a compute part, and to ensure connectivity for a network request and the participants on both ends. We distinguish two types of spatial relationships – one including non-network resources only and one for linking network resources and non-network resources. The former is denoted by the superscript nnt, the latter by the superscript net.

Definition 13 (Non-Network Spatial Relationship). *We model spatial relationships between two assignments* A *and* B *involving non-network resources only by special equality constraints denoted by* $sr^{\mathsf{nnt}}_{A:B,h}$*, which is a mapping*

$$sr^{\mathsf{nnt}}_{A:B,h} : (T \times D \times Q)^2 \longrightarrow \mathbb{R}\,,$$

with $h \in \mathbb{N}$ non-network *spatial relationships between the assignments* A *and* B*.* ♢

Definition 14 (Network Spatial Relationship). *A spatial relationship linking two assignments* A *and* B *of non-network resources and an assignment* C *of a network resource is a mapping*

$$sr^{\mathsf{net}}_{A:B:C,h} : (T \times D \times Q)^3 \longrightarrow \mathbb{R}^2\,,$$

with $h \in \mathbb{N}$ network *spatial relationships between the assignments* A*,* B *and* C*.* ♢

The actual mapping of the spatial relationships $sr^{\mathsf{nnt}}_{A:B,h}$ and $sr^{\mathsf{net}}_{A:B:C,h}$ is constructed by a specific combination of the spatial properties in $P_A \cup P_B$ and $P_A \cup P_B \cup P_C$, respectively. The construction uses, in particular, the properties $p^{left}_{\triangleright s}$ and $p^{right}_{\triangleright s}$, which denote the *left* and the *right* network end-point of an assignment involving the resource s. For assignments involving point-to-point *network* resources, these properties evaluate to different values because they connect two sides. For assignments involving *non-network* resources, they evaluate to the same value.

Definition 15 (Boolean Value of Non-Network Spatial Relationships). *The boolean value* bool() *of a* non-network *spatial relationship* $sr^{\mathsf{nnt}}_{A:B,h}$ *is defined as follows*

$$\mathsf{bool}\left(sr^{\mathsf{nnt}}_{A:B,h}(t_a,d_a,q_a,t_b,d_b,q_b)\right)=\begin{cases}1 & sr^{\mathsf{nnt}}_{A:B,h}(t_a,d_a,q_a,t_b,d_b,q_b)=0\,,\\ 0 & \text{elsewise}\,.\end{cases}$$

♢

Definition 16 (Boolean Value of a Network Spatial Relationship). *The boolean value* bool() *of a* network *spatial relationship* $sr^{\mathsf{net}}_{A:B:C,h}$ *is defined as follows*

$$\mathsf{bool}\left(sr^{\mathsf{net}}_{A:B:C}(t_a,d_a,q_a,t_b,d_b,q_b,t_c,d_c,q_c)\right)=\begin{cases}1 & sr^{\mathsf{net}}_{A:B:C}(t_a,d_a,q_a,t_b,d_b,q_b,t_c,d_c,q_c)=(0,0)\,,\\ 0 & \text{elsewise}\,.\end{cases}$$

♢

The sets $SR^{\mathsf{nnt}}_{A:B}$ and $SR^{\mathsf{net}}_{A:B:C}$, respectively, contain all spatial *non-network* and *network* relationships between the assignments A and B as well as among the assignments A, B and C. The set

$$\begin{aligned}SR \quad = \quad & \bigcup_{\substack{r_{l1}\in R,\\ s_{k1}\in S(r_{l1})}} \bigcup_{\substack{r_{l2}\in R,\\ s_{k2}\in S(r_{l2})}} SR^{\mathsf{nnt}}_{r_{l1}\triangleright s_{k1}:r_{l2}\triangleright s_{k2}} \\ \cup \quad & \bigcup_{\substack{r_{l1}\in R,\\ s_{k1}\in S(r_{l1})}} \bigcup_{\substack{r_{l2}\in R,\\ s_{k2}\in S(r_{l2})}} \bigcup_{\substack{r_{l3}\in R,\\ s_{k3}\in S(r_{l3})}} SR^{\mathsf{net}}_{r_{l1}\triangleright s_{k1}:r_{l2}\triangleright s_{k2}:r_{l3}\triangleright s_{k3}}\end{aligned}$$

contains all non-network and network spatial relationships.

Example 5.3 (Transfer of Data)
Transferring a data set from an archive (request r_1) to a supercomputer (request r_3) requires a network resource (request r_2) connecting the archive and the supercomputer. The spatial properties of seven resources are given by the table

	Resource						
Property	s_1	s_2	s_3	s_4	s_5	s_6	s_7
p^{left}	1	1	9	1	3	2	2
p^{right}	1	9	9	3	3	3	2

.

The following assignments are possible $A_1 = r_1 \triangleright s_1$, $A_2 = r_1 \triangleright s_7$, $B_1 = r_2 \triangleright s_2$, $B_2 = r_2 \triangleright s_4$, $B_3 = r_2 \triangleright s_6$, $C_1 = r_3 \triangleright s_3$ and $C_2 = r_3 \triangleright s_5$. Thus, the following 12 spatial constraints sr^{net} implement the desired relationship, where $\mathbf{tdq}$ *abbreviates the parameters $t_1, d_1, q_1, t_2, d_2, q_2, t_3, d_3, q_3$.*

$$\begin{aligned}&sr^{\mathsf{net}}_{A_1:B_1:C_1,1}(\mathbf{tdq})=(0,0),\quad sr^{\mathsf{net}}_{A_1:B_1:C_2,2}(\mathbf{tdq})=(0,1),\quad sr^{\mathsf{net}}_{A_1:B_2:C_1,3}(\mathbf{tdq})=(0,1)\\ &sr^{\mathsf{net}}_{A_1:B_2:C_2,4}(\mathbf{tdq})=(0,0),\quad sr^{\mathsf{net}}_{A_1:B_3:C_1,5}(\mathbf{tdq})=(1,1),\quad sr^{\mathsf{net}}_{A_1:B_3:C_2,6}(\mathbf{tdq})=(1,0)\\ &sr^{\mathsf{net}}_{A_2:B_1:C_1,7}(\mathbf{tdq})=(1,0),\quad sr^{\mathsf{net}}_{A_2:B_1:C_2,8}(\mathbf{tdq})=(1,1),\quad sr^{\mathsf{net}}_{A_2:B_2:C_1,9}(\mathbf{tdq})=(1,1)\\ &sr^{\mathsf{net}}_{A_2:B_2:C_2,10}(\mathbf{tdq})=(1,0),\quad sr^{\mathsf{net}}_{A_2:B_3:C_1,11}(\mathbf{tdq})=(0,1),\quad sr^{\mathsf{net}}_{A_2:B_3:C_2,12}(\mathbf{tdq})=(0,0)\end{aligned}$$

A **Multi Assignments Constraint** models constraints between multiple assignments.

Definition 17 (Multi Assignments Constraint). *A constraint between multiple assignments $MA = (ma_1, \ldots, ma_L)$ is a mapping*

$$mac^{=,\geq}_{MA,h} : (T \times D \times Q)^L \longrightarrow \mathbb{R}\,,$$

with $h \in \mathbb{N}$ enumerating the constraints between the assignments MA and L being the number of requests in R. ♢

The actual mapping of a constraint $mac^{=,\geq}_{MA}$ is constructed as some arbitrary combination of the properties in $P_{MA} = \bigcup_{i=1}^{L} P_{ma_i}$. Note, a mapping may constrain all assignments ($|MA| = L$), but it does not need to do so. Which assignments are constrained by a specific *multi assignments constraint* is defined by the actual mapping.

Example 5.4 (Limiting the Total Reservation Cost)
The total reservation cost of the 3-step job chain (cf. Examples 5.1, 5.2 and 5.3) must not exceed the maximum budget $p^{budget}_{r_1\triangleright}$. The reservation costs of the steps are given by the properties $p^{cost}_{r_1\triangleright}$, $p^{cost}_{r_2\triangleright}$ and $p^{cost}_{r_3\triangleright}$. Then, the budget constraint is expressed as follows

$$mac^{\geq}_{r_1\triangleright:r_2\triangleright:r_3\triangleright,1}\,(t_1, d_1, q_1, t_2, d_2, q_2, t_3, d_3, q_3) \;=$$

$$p^{budget}_{r_1\triangleright}\,(t_1, d_1, q_1, t_2, d_2, q_2, t_3, d_3, q_3) - p^{cost}_{r_1\triangleright}\,(t_1, d_1, q_1, t_2, d_2, q_2, t_3, d_3, q_3) - p^{cost}_{r_2\triangleright}\,(t_1, d_1, q_1, t_2, d_2, q_2, t_3, d_3, q_3) - p^{cost}_{r_3\triangleright}\,(t_1, d_1, q_1, t_2, d_2, q_2, t_3, d_3, q_3)\;.$$

Note, we arbitrarily associated the budget property with request r_1.

Definition 18 (Boolean Value of a Multi Assignments Constraint). *The boolean value* $\mathsf{bool}()$ *of a constraint $mac^{cop}_{MA,h}$ is defined as follows*

$$\mathsf{bool}\left(mac^{cop}_{MA,h}(t_1, d_1, q_1, \ldots, t_L, d_L, q_L)\right) \;=$$

$$\begin{cases} 1 & cop \text{ equals } '=' \;\wedge\; mac^{=}_{MA,h}(t_1, d_1, q_1, \ldots, t_L, d_L, q_L) = 0\,, \\ 1 & cop \text{ equals } '\geq' \;\wedge\; mac^{\geq}_{MA,h}(t_1, d_1, q_1, \ldots, t_L, d_L, q_L) \geq 0\,, \\ 0 & \text{elsewise}\,. \end{cases}$$

♢

The set of all constraints between the assignments in MA is denoted by MAC_{MA}. The set

$$MAC = \bigcup_{MA \in \mathcal{P}(CA)} MAC_{MA}$$

contains all constraints on all multi assignments.

Feasible Solution. We call a candidate solution CA^S a *feasible* solution if the following conjunction holds (tdq_X abbreviates t_X, d_X, q_X for $X \in CA^S$).

$$\left(\bigwedge_{\langle r,s,t,d,q\rangle \in CA^S} \bigwedge_{sac \in SAC_{r \triangleright s}} \mathsf{bool}\left(sac(t,d,q)\right)\right) \wedge \left(\bigwedge_{A,B \in CA^S \wedge A \neq B} \bigwedge_{tr \in TR_{A:B}} \mathsf{bool}\left(tr(tdq_A, tdq_B)\right)\right) \wedge$$
$$\left(\bigwedge_{A,B \in CA^S \wedge A \neq B} \bigwedge_{sr \in SR^{\mathsf{nnt}}_{A:B}} \mathsf{bool}\left(sr(tdq_A, tdq_B)\right)\right) \wedge$$
$$\left(\bigwedge_{A,B,C \in CA^S \wedge A \neq B \wedge B \neq C \wedge A \neq C} \bigwedge_{sr \in SR^{\mathsf{net}}_{A:B:C}} \mathsf{bool}\left(sr(tdq_A, tdq_B, tdq_C)\right)\right) \wedge$$
$$\left(\bigwedge_{MA \in \mathcal{P}(CA^S)} \bigwedge_{mac \in MAC_{MA}} \mathsf{bool}\left(mac(t_1, d_1, q_1, \ldots, t_{|MA|}, d_{|MA|}, q_{|MA|})\right)\right)$$

The above condition only takes the combinations $CA^S \subseteq CA$ into account, because the assignments in $CA \setminus CA^S$ involve the virtual resource 0. That is, the corresponding requests are not mapped to a resource, and their constraints need not be taken into account.

5.6 Optimization Criteria

We distinguish objectives of the requests and the resources. While their definition is essentially the same, the differentiation is made to formulate some conditions on their construction (see below).

Definition 19 (Objective of a Co-Reservation Request). *A co-reservation request's objective on a combination of assignments CA^S is a pair $\left(o^f_{R,CA^S,h}, o^\omega_{R,CA^S,h}\right)$ of*

$$\begin{aligned} \textit{a mapping} \quad & o^f_{R,CA^S,h} : (T \times D \times Q)^L \longrightarrow [-1,1] \\ \textit{and a weight} \quad & o^\omega_{R,CA^S,h} \in [0,1]\,, \end{aligned}$$

with $h \in \mathbb{N}$ enumerating the request's objectives on a combination of assignments CA^S. ♢

The mapping $o^f_{R,CA^S,h}$ of an objective $o_{R,CA^S,h}$ is constructed as some arbitrary combination of the properties in P_{CA^S}. The set O_{R,CA^S} contains all objectives of the co-reservation request R on the combination of assignments CA^S.

Definition 20 (Objective of a Resource). *An objective of a resource s_k on a combination of assignments CA^S is pair $\left(o^f_{s_k,CA^S,h}, o^\omega_{s_k,CA^S,h}\right)$ of*

$$\begin{aligned} \textit{a mapping} \quad & o^f_{s_k,CA^S,h} : (T \times D \times Q)^L \longrightarrow [-1,1] \\ \textit{and a weight} \quad & o^\omega_{s_k,CA^S,h} \in [0,1]\,, \end{aligned}$$

with $h \in \mathbb{N}$ enumerating the resource's objectives on a combination of assignments CA^S. ♢

The mapping $o^f_{s_k,CA^S,h}$ of an objective $o_{s_k,CA^S,h}$ is constructed as some arbitrary combination of the properties in P_{CA^S}. The sets O_{s_k,CA^S} $(k = 1,\ldots,K)$ contain all objectives of the resource s_k on the combination of assignments CA^S.

Let $FS = \{CA^S_i\}$ denote the set of all feasible solutions. The weights of the objectives must satisfy the conditions

$$\forall CA^S \in FS : \sum_{o \in O_{R,CA^S}} o^\omega = 1, \qquad \text{and} \qquad \forall s \in S, \quad \forall CA^S \in FS : \sum_{o \in O_{s,CA^S}} o^\omega = 1\,.$$

Let the set $O(CA^S)$ contain all objectives on a combination of assignments CA^S, i.e., $O(CA^S) = O_{R,CA^S} \cup O_{s_1,CA^S} \cup \ldots \cup O_{s_K,CA^S}$.

Definition 21 (Global Optimization Criteria). *The criteria $B : FS \longrightarrow \mathbb{N}$ defines an order on the feasible solutions FS by applying some function to the objectives $O(CA^S)$.* ♢

We call a feasible solution CA^S *optimal* if the following condition holds.

$$B\left(CA^S\right) = \min_{fs \in FS} B(fs)$$

Criteria for calculating B could be the weighted sum, a prioritization of the objectives or the Pareto-set of the solutions.

Note, the reservation system can specify the criteria B such that the objectives of either side – the requests or the resources – are preferred or balanced.

Example 5.5 (Min End Time & Max Fitness)
The user wants the results of the job chain as soon as possible. In contrast, the resources want to maximize the fitness of the assignments. The user's goal is expressed as follows

$$\begin{aligned} o^f_{R,CA^S,1}\left(t_1,d_1,q_1,t_2,d_2,q_2,t_3,d_3,q_3\right) \;=\; & \\ & p^{stt}_{r_3\triangleright}\left(t_1,d_1,q_1,t_2,d_2,q_2,t_3,d_3,q_3\right) + \\ & p^{dur}_{r_3\triangleright}\left(t_1,d_1,q_1,t_2,d_2,q_2,t_3,d_3,q_3\right) \\ o^\omega_{R,CA^S,1} \;=\; 1\,. & \end{aligned}$$

The goal of the resources is expressed by the following objectives

$$\begin{aligned} \forall s_k \in S : & \\ & o^f_{s_k,CA^S,1}\left(t_1,d_1,q_1,t_2,d_2,q_2,t_3,d_3,q_3\right) \;= \\ & \qquad (-1)\cdot p^{fit}_{\triangleright s_k}\left(t_1,d_1,q_1,t_2,d_2,q_2,t_3,d_3,q_3\right) \\ & o^\omega_{s_k,CA^S,1} \;=\; 1\,. \end{aligned}$$

Note, the fitness property is multiplied by -1 to invert the optimization sense (minimize → maximize).

Part II

Specifying, Processing & Using Co-Reservations

Chapter 6

General System Architecture

First, we describe the life cycle of a co-reservation (cf. Section 6.1). Then, we present the general system architecture (cf. Section 6.2). Last, we give an overview of the processing steps (cf. Section 6.3).

6.1 The Life Cycle of a Co-Reservation

A co-reservation is composed of m atomic reservations ($m \in \mathbb{N}_+$). First, we describe the life cycle of an atomic reservation. Then, we define the life cycle of a co-reservation.

Life Cycle of an Atomic Reservation. Figure 6.1 shows the life cycle of an atomic reservation. Any atomic reservation begins in the state specified and ends in one of the states failed, done or canceled. During the processing of a request a reservation may iterate multiple times over the states resource candidates, reservation candidates, candidate selected and denied. A granted reservation may either be active or inactive.

The state changes in the middle pillar and the state inactive correspond to the simplest failure-free life cycle of an atomic reservation. Successful requests for reserving resources in advance stay inactive until their begin time is reached.

The processing re-iterates over the states resource candidates, reservation candidates and candidate selected if the original candidate could not be acquired (upward arrows starting from state denied) or if a change of an inactive reservation was requested (upward arrows starting from state inactive).

The state canceled may be reached if a granted reservation (states active and inactive) is terminated. An active reservation becomes done if its end time is reached.

An atomic reservation fails (state failed) if no alternatives to denied candidates are found or if a resource failure occurs (dashed arrows in Fig. 6.1).

Life Cycle of a Co-Reservation. Let z_j denote the state of the j-th atomic reservation ($j = 1, \ldots, m$). The tuple $Z_k := \langle z_1, \ldots, z_m \rangle$ describes the k-th state in the life cycle of a co-reservation with m parts ($k \in \mathbb{N}$).

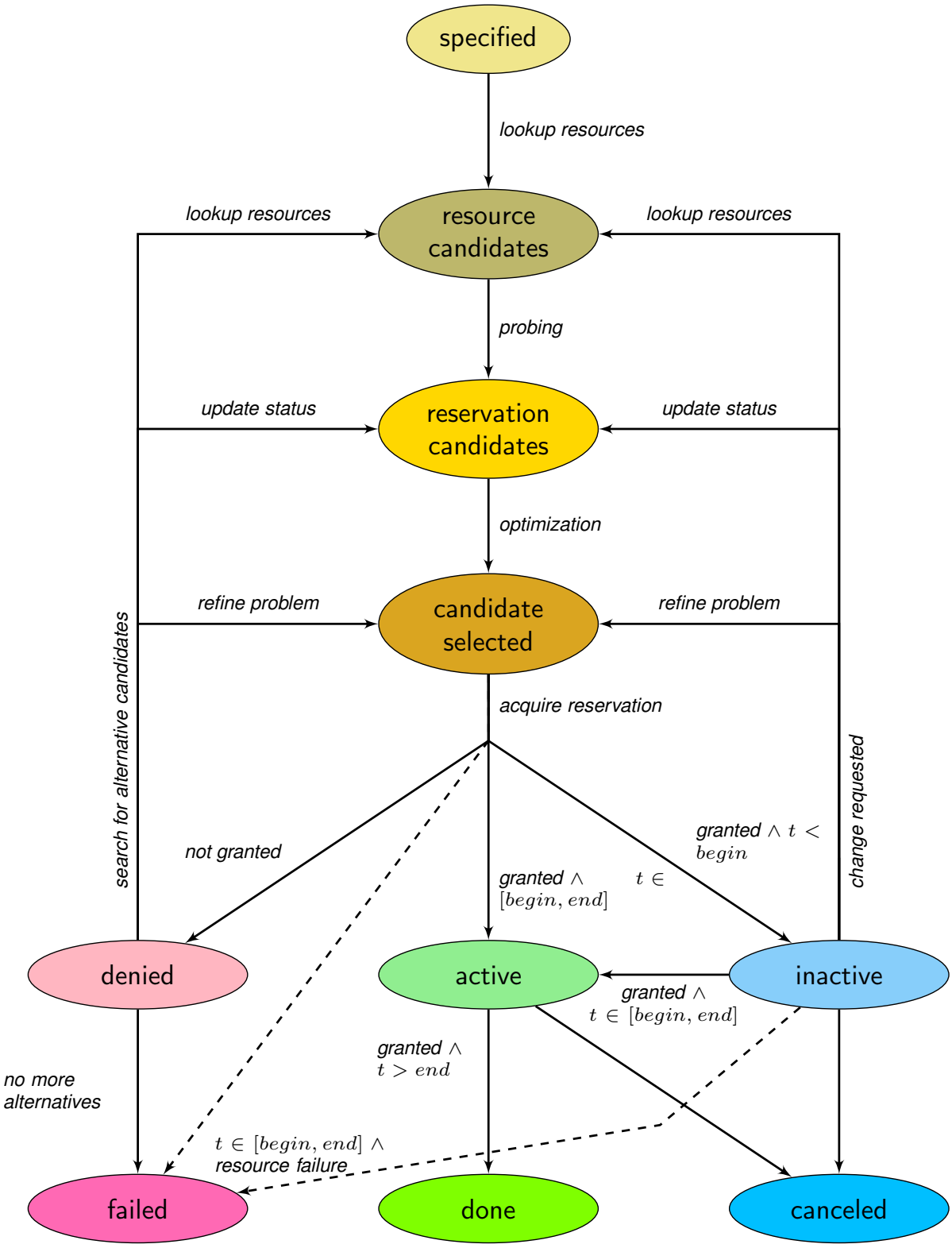

Figure 6.1: Life cycle of an atomic reservation.

Any co-reservation begins its life cycle in the state $Z_0 := \langle \mathsf{specified}, \ldots, \mathsf{specified} \rangle$. Different parts of a co-reservation may be in different states at the same time. Hence, the life cycle of a co-reservation can be described as state sequence $ZS := Z_0 Z_1 Z_2 \ldots$, where, without loss of generality, two succeeding states Z_k and Z_{k+1} only differ in the state change of one atomic reservation.

A co-reservation ends if all parts are in one of the terminal states failed, done and canceled. Note, different parts may be in different terminal states.

6.2 System Architecture

Figure 6.2 shows the three main components of the co-reservation framework CORES: the *Grid Reservation Service* (GRS), the *Resource Catalog* (RC) and the *Local Reservation Service* (LRS). The interplay of these components is depicted by arrows, whose numbers and labels refer to the description of the processing steps presented in Section 6.3.

Note, while Fig. 6.2 only shows a single instance of the GRS, the RC and the client, multiple instances of each of these entities may exist. For example, different clients may use different GRS instances, which compete for the same resources. However, a single co-reservation request is processed by a single GRS instance only.

Grid Reservation Service. The Grid Reservation Service is the central component which receives co-reservation requests and coordinates their processing. It may operate in two modes: (1) single request and (2) bucket of requests. In the former, it processes incoming requests serially. Thus, the client receives the response as early as possible. In the latter, it gathers requests and processes them together to achieve a higher resource utilization and fair sharing among the clients.

Introducing the GRS as the central component does not exclude the existence of multiple instances of GRSs. In particular, each large organization or even small groups of researchers may deploy their own instance which can incorporate domain specific knowledge about both the applications of the researchers and the resources they wish to use. Because the GRS coordinates the processing of co-reservation requests, each of the following chapters covers some of its aspects. Particularly, the steps for mapping requests to co-reservation candidates and for allocating resources to candidates are presented in Chapter 10 and 11, respectively.

Resource Catalog. The Resource Catalog (RC) stores static information about the resources. Each resource registers itself with at least one RC providing information such as its type, its capacity, its performance metrics, authorization requirements, resource management capabilities, etc. An RC offers a querying interface which is used by the GRS to determine reservable resources matching static requirements of a co-reservation request. The description of resources and their matching with request parts are described in Chapters 7 and 8.

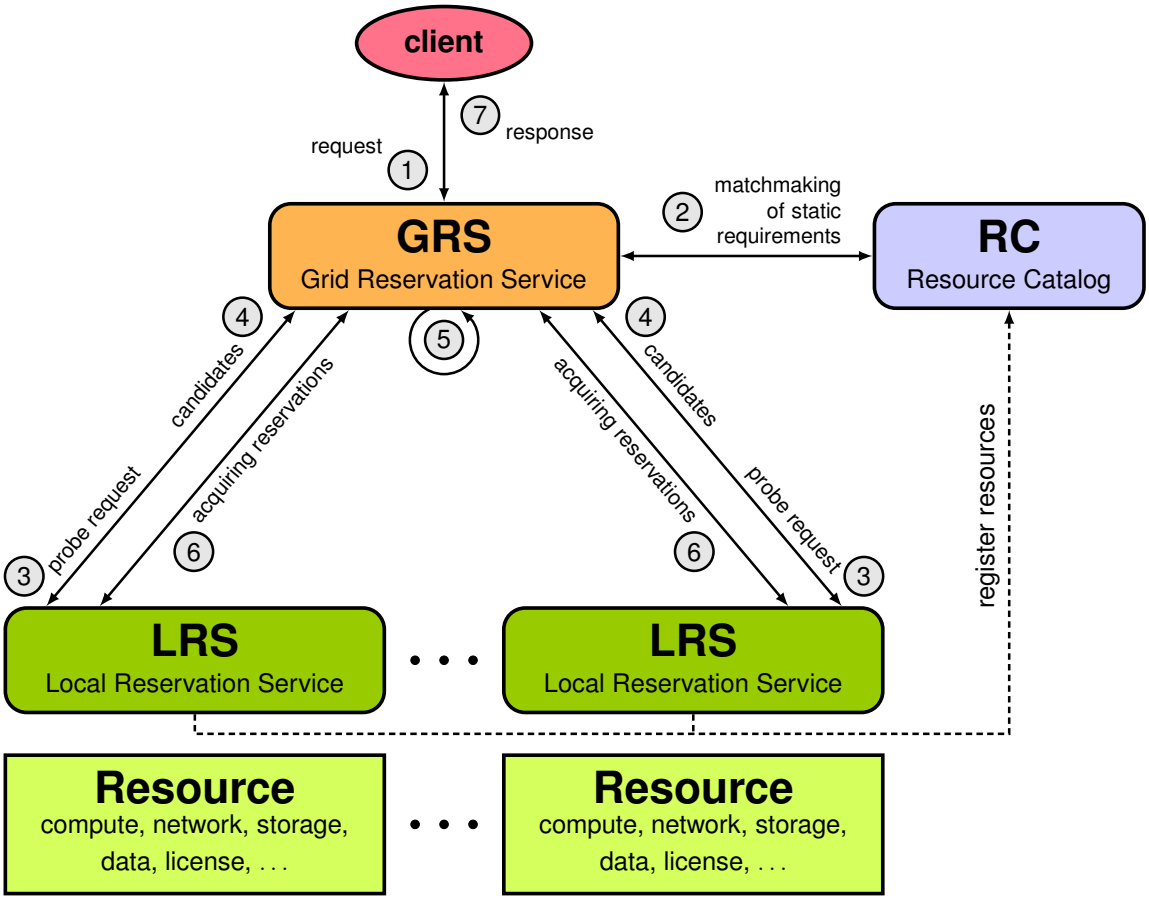

Figure 6.2: Components of the reservation framework and their interplay in the processing of a co-reservation request.

Local Reservation Service. The Local Reservation Service (LRS) provides a generic interface to the resource management system of a resource and enhances its functionality to support the reservation mechanism. By providing a generic interface it simplifies the communication with the GRS. The proposed co-reservation mechanism needs information about the future status of the resources. Because, current resource management systems provide only very limited information, the LRS integrates mechanisms for deriving the information. The advantage of deriving such information at the resource is, that all – especially confidential – local information is available and each resource can more easily control what information is made available to the GRS. We describe several methods for *probing* the future status of resources in Chapter 9.

6.3 Processing Steps

The following processing steps cover the life cycle of a co-reservation except for the states done and canceled and the state changes originating in state the inactive.

① The co-reservation request is described in the *Simple Reservation Language (SRL)* and send to the Grid Reservation Service (GRS).

The SRL is presented in Chapter 7.

② On receipt of a request, the GRS queries resource catalogs to determine eligible resources which match the requested type as well as static characteristics such as the required operating system.

This step is discussed in Chapter 8.

③ The eligible resources are asked to provide detailed status information covering the time period of the co-reservation request.

This step is described in Chapter 9.

④ The GRS compiles the status information, the constraints and the objectives of all involved parties into a single instance of an optimization problem and solves it by using standard tools.

This step is discussed in Chapter 10.

⑤ To acquire the reservations represented by the solution, the GRS sends reserve messages – one for each co-reservation part – to the LRSs of the selected resources.

This step is discussed in Chapter 11.

⑥ If some reservations were not granted, the optimization problem is refined and the mechanism continues with step ④.

This step is described in Section 10.6.

⑦ If all reservations were admitted or no solution could be found, the reservation system generates a corresponding response message.

Chapter 7

Description of Requests and Resources

The processing of co-reservation requests begins with a description of the requirements and objectives of all involved parties – the users and the providers. We propose the *Simple Reservation Language* (SRL) for specifying requests and resources. Designing a description language must answer the general questions:

- What kind of information should be described?, *and*
- How should the information be structured?

From a user's point of view a request description must define:

- Who is requesting a reservation?,
- What types of resources and which quantity *or* Quality-of-Service of each resource shall be reserved?,
- When should the reservation begin and how long will it last?, *and*
- Which reservation parameters are preferred if the reservation system may select among many?

From the provider's point of view a resource description must define:

- Who is allowed to acquire reservations?,
- What type of service and which service level (Quality-of-Service) does the resource provide?,
- Which service manages reservation requests?, *and*
- Which constraints and objectives shall be met by any reservation?

Chapter Outline. In Section 7.1 we discuss the requirements on such a language. The state of the art in description languages for requests and resources is presented in Section 7.2. Thereafter, we present the *Simple Reservation Language* in detail (cf. Section 7.3). The chapter is summarized in Section 7.4.

7.1 Requirements

The co-reservation procedure involves several participants – the consumers and the providers – with diverse and often conflicting goals. We derive requirements of a **uniform** language, which lets all participants describe their functional and non-functional requirements. The language needs to support a wide range of scenarios – from rigid atomic requests and resources to flexible multi-part requests (cf. Chapter 2).

7.1.1 Functional Requirements

A co-reservation consists of multiple atomic parts. First, we present the requirements considering atomic parts only. Thereafter, we discuss additional requirements to compose multiple parts into a co-reservation. In each category, we describe issues related to *properties*, *constraints* and *objectives*.

Properties of Atomic Parts

A description language must provide means to describe the fixed as well as the variable properties of an atomic part. **Fixed properties** define information which does not depend on the matching. In contrast, the value of a **variable property** depends on the parameters offered by a matching candidate.

Example 7.1 (Fixed properties)
*An atomic reservation **request** may have a type, for example compute, storage or network, which does not depend on the matching. Also, the name of the requester is the same in all possible matchings.*

*Similarly, the type of a **resource** does not depend on the matching. Also, properties describing hardware features, e.g., architecture of a processor or the network technology, are the same for all matchings.*

Example 7.2 (Variable properties)
The duration of a reservation depends on the quality-of-service offered in a matching. For example, the transfer time of 10 gigabyte of data depends on the available bandwidth as illustrated in the table below.

	Data Sizes				
Bandwidth	*1 GB*	*500 GB*	*1 TB*	*1 PB*	*10 PB*
10 Mbit/s	*15′*	*5 d*	*10 d*	*25 y*	*250 y*
100 Mbit/s	*1′40″*	*12 h*	*24 h*	*2.5 y*	*25 y*
1 Gbit/s	*10″*	*1 h*	*2h*	*3 m*	*2.5 y*
10 Gbit/s	*1″*	*6′*	*12′*	*10 d*	*3 m*
40 Gbit/s	*1″*	*1′30″*	*3′*	*2.5 d*	*1 m*

A similar relation exists for the execution time of a program, which depends – among many other parameters – on the clock frequency of the processor and on the number of processors being used. A basic relation is defined by Eq. (4.6). In a commercial scenario,

the price a consumer is willing to pay may depend on the parameters of the matching candidate. For example, the higher is the offered bandwidth or the earlier would the request finish, the higher is the available budget.

Constraints of Atomic Parts

While properties are used to describe the parameters of a request or a resource, constraints are used to pose restrictions on possible matchings between requests and resources. Thus, the set of **feasible** matchings is described by the properties and the constraints.

Because the properties of a part may be variable and depend on the matching party, it must be possible to formulate constraints on all attributes of a matching.

Example 7.3 (Constraints of atomic parts)
*A **request** for a compute resource may pose constraints on the type of the resource, i.e., compute, on the architecture of the processor, the operating system, the clock frequency of the processor. Furthermore, it may limit the number of used processors if the scalability of the program is bounded.*

*A **resource** will restrict the types and sizes of requests it serves, or grant access to certain users or virtual organizations (VO) only. It may also impose limits on the aggregated share assigned to all reservations or to all allocations of a certain user or VO by enforcing a threshold on the* fit *value, which is a variable property of a resource.*

Objectives of Atomic Parts

Assuming that a request and a resource can participate in many matchings, both the users and the providers may prefer certain matchings over others. The description language must provide means to describe these preferences called **objectives**. If a request consists of multiple objectives, it must be possible to declare their (relative) importance.

Example 7.4 (Objectives of atomic parts)
Common user objectives are to minimize the completion time of a compute task or to minimize the costs of using a resource.

In contrast, a resource provider may want to maximize the utilization of its resources or minimize the power consumption.

Properties of Multiple Parts

A request consisting of multiple parts may need to define global properties such as the total budget or a global identifier. Similarly, multiple resources can be grouped together. Such a group may be described with global properties like an identifier, the total capacity or an abstract capacity in the case of composite resources.

Constraints of Multiple Parts

We observe three sources for constraints of multiple parts. First, temporal relationships between pairs of matchings are needed. For example, a distributed parallel simulation may require that all parts begin at the same time. In a job chain scenario with known execution times for each request part, temporal constraints define the order in which the parts are executed. Second, spatial relationships between pairs of matchings need to be described. For example, if two compute parts shall be connected by a dedicated link, we need two spatial constraints which specify that compute resource one is connected to one endpoint of the link while compute resource two is connected to the other endpoint. Third, aggregated parameters may be restricted. For example, if the total costs for all parts must not exceed a certain budget.

Objectives of Multiple Parts

Objectives of multiple parts are needed to specify global preferences. For example, a user may want to minimize the costs of a request or minimize the total completion time of a job chain. The reservation system may prefer combinations with a high aggregated fitness value, i.e., those for which subsequent reserve messages are likely to succeed.

Auxiliary Request Parts

Auxiliary request parts describe resources which shall not be reserved, but influence the other matchings. For example, data resources can be requested with an auxiliary part. Such part may match multiple instances of a data item, e.g., replica of a file. Considering the network bandwidth between the data resources and eligible compute sites, the co-reservation mechanism can chose the best combination of data, network and compute resources, but will only reserve the compute and network resources[1]. Additionally, auxiliary request parts can be used to describe existing reservations which shall be replaced or augmented by new parts. Note, replacing an existing reservation with a new one may require additional support from the resource's local management system.

7.1.2 Non-functional Requirements

Non-functional requirements define criteria on the quality of a system. A description language defines a format for exchanging information between different components. In the co-reservation framework, information is exchanged between users and the GRSs, but also between the GRSs and the resource catalogs and the LRSs. Hence, the language must be **easy to use** by human beings and **portable** to simplify the exchange of information. Since, the number of feasible matchings may be very high and

[1] In fact, we may wish to reserve data resources in the sense that the accessed data is made available on-line before it is read by a job, i.e., staged from tape to disk for low latency access.

the state of the resources in a Grid may change quickly, it must provide all information to facilitate an **efficient** processing. In particular, no communication with a user should be needed to determine the preferred set of candidates. The language must be **extensible** both, in the type of the resources or requests and in the set of properties, constraints and objectives it supports. Traditionally, most resource management systems consider only a small set of types of resources. These types are compute, storage and network. Applying Grid technology to areas different from high-performance computing necessitates the representation of additional types of resources such as web services or robotic telescopes.

Specifying requests and resources in the same language, acknowledges the symmetry of the matching process, simplifies the deployment of the language and improves resolving errors. Therefore, requests and resources shall be described by **uniform** means.

7.2 State of the Art

Languages for describing requests and resources play a fundamental role to resource management. In this section, we will review approaches from batch systems, Grid resource management systems and generic resource description frameworks. We divided the languages in three categories, depending on if they may be used to describe (1) requests only (cf. Section 7.2.1), (2) resources only (cf. Section 7.2.2) or (3) both requests and resources (cf. Section 7.2.3). In each category, we present a prominent language in detail and relate other approaches to it. Despite the long history of describing resources and requests, we will focus at languages developed over the past 20 years only. We summarize the as-is state of the features of the presented languages in Table 7.2.

7.2.1 Request Description Languages

In the Globus Toolkit (GT) [GT], resource allocation requests are described with the GT *Resource Specification Language* (RSL). The language groups relations of an atomic part by the operator `&` and supports the composition of complex requests by the operator `+`. A relation is either a binding of a value to a key or a comparison of a key with a value. The former is expressed by assignment statements and mainly used for describing parameters of the job execution such as the executable, the program arguments, needed directories, etc. The latter is described by boolean expressions and used to define constraints on the matching resources.

Recently, the NorduGrid project [NG] proposed *xRSL* – an extended version of Globus RSL. xRSL extends RSL by introducing new keys and a distinction between the *User-side RSL* and the *GM-side RSL*[2]. A user only needs to provide keys of the user-side part. Keys of the GM-side are added by a special user interface of the NorduGrid

[2]GM stands for Grid Manager.

middleware.

The Open Grid Forum (OGF) [OGF] conducts standardization efforts to define a common language for submitting jobs. The OGF Job Submission Description Language (JSDL) [JSD] is an XML[3] format to describe properties and requirements of a Grid job. Essentially, JSDL provides the same features as xRSL does.

The UNiform Interface to COmputing REsources (UNICORE) [Uni08] was developed to provide access to supercomputers at high-performance computing (HPC) centers. UNICORE's job model supports single task and workflow applications. Each task is described by its fixed resource requirements, its input and output data and the actions it will perform. All these information are encapsulated in the *Abstract Job Object* (AJO).

The above languages, designed to solely describe job requests, can be seen as generalized batch job description languages. For example, the cluster batch system TORQUE [TOR08] uses a specific list of pre-defined attributes to describe a job. Names of attributes may be different among cluster batch systems. A Grid-level description language defines common names for attributes. During the submission of a job, these attribute names need to be mapped to the corresponding name of the target system.

7.2.2 Resource Description Languages

The Monitoring and Discovery Service (MDS) [MDS] provides means to store and query information on the characteristics and current load of, compute and storage resources. It is a building block of the Globus Toolkit [GT] since its first version was released in 1997. MDS includes components for caching and aggregating information, the Index Service, and for notification of certain conditions, the Trigger Service. Because of its flexible architecture and use of well-known web standards such as XML for the representation of information and XSLT[4] for transforming the information format of different information providers, MDS can manage information of diverse entities. The information is stored in (key,value)-attributes, which may be grouped into classes defined by a schema. An example for a widely used schema is the GLUE schema [GLU08]. The Grid Laboratory Uniform Environment (GLUE) schema was initially developed as a joint effort by the EU projects DataGrid [EDG08] and DataTAG [EDT08] and the then international Virtual Data Grid Laboratory (iVDGL). The GLUE schema aims to ease the sharing of compute and storage resources managed by different Grid middlewares or associated with different Grid environments. In particular, it provides a uniform representation of the resources, thus simplifying the design of Grid brokers and information indexes. The schema defines several classes of (key,value)-attributes, which cover specific aspects of a resource. For example, the class *ComputingElement* encapsulates a batch queue of a local resource management system. Table 7.1 shows selected attributes of the class ComputingElement (cf. version 1.3 of the GLUE schema specification [GLU07]).

[3] XML stands for the standard *eXtensible Markup Language* of the W3C (http://www.w3.org/).

[4] XSLT stands for the W3C recommendation of XSL Transformations (http://www.w3.org/TR/xslt).

Table 7.1: Selected attributes of the GLUE schema class *ComputingElement*.

Name	Description	Value
Info.TotalCPUs	Total number of processors	128
Benchmark.SI00	SPECint2000 value of a processor	1569
Processor.ClockSpeed	Clock frequency (MHz) of a processor	2600
MainMemory.RAMSize	Main memory size (megabyte) of a node	512
State.TotalJobs	Total number of current jobs	144
State.RunningJobs	Number of running jobs	42
State.WaitingJobs	Number of waiting jobs	100

The UNICORE resource model [Uni08] covers the capabilities and capacity of a HPC system. It also provides rules for enacting tasks, e.g., setting environment variables and providing configuration parameters. The capacity of a system is described with a small set of attributes such as the number of nodes, the number of processors per node, the main memory per node, the queues, the CPU time limit, etc. Access to a resource is only granted to users which are registered with the UNICORE User DataBase.

7.2.3 Symmetric Description Languages

Condor [LLM88] is a resource management for clusters built from workstations or dedicated machines. The core of Condor is its very flexible resource and request description mechanism *Condor ClassAds* [RLS98]. Both, the resources and the requests are described by classified advertisements (short: ClassAd) using the same syntax and semantics. Each ClassAd describes an entity by attributes, requirements and a ranking criteria. An attribute is a (key,value)-pair such as the type of the entity or its name. Requirements are complex boolean expressions, whose clauses refer to attributes of matching candidates. A ranking criteria is an arithmetic function, which may reference values of the matching candidates, too. For example, a job request may prefer the resource with the fastest processor. Matching two entities is a symmetric process, that is, entity A matches B and vice versa. In particular, both entities may specify requirements and ranking criteria, which must be satisfied. Due to the generic model Condor ClassAds builds upon, it satisfies many of the requirements listed in Section 7.1. Therefore, the SRL will inherit the core elements of Condor ClassAds and extend it where necessary (cf. Section 7.3).

The RedLine Description Language [LF03b] is a generalization of Condor ClassAds, more specifically, it replaces (key,value)-attributes by equality constraints and the requirements attribute by a set of constraints. Thus, a RedLine ClassAd consists of constraints only. Bi-lateral matching is performed by the conjunction of the constraints of

the matching candidates. Multi-lateral matching is defined by two constructs *ISA* and *ISA SET*, which facilitate the description of a multi-part request or resource.

In his dissertation *"A REST Model for High Throughput Scheduling in Computational Grids"*, Stokes-Rees [SR06] proposes a set theory formalization of Condor ClassAds. While Condor ClassAds needs to handle tri-state logic[5], this is avoided in the proposed Grid Resource Description Language (GRDL) [SR06]. Thus, the language is easier to implement and to use. In contrast to many other approaches, the GRDL supports the heterogeneity in Grids by applying transformations on attributes. For example, the available storage capacity of a resource, given in units of one terabyte, can be converted into units of one gigabyte if a request uses this base unit.

The D-Grid Resource Description Language (D-GRDL) [Wol07] is also based on XML, but enables hierarchies of resource classes as well as aggregations of multiple resources. For example, a compute node may aggregate several software packages. The D-GRDL allows arbitrary constraints in disjunctive normal form. Since it only aims at identifying a set of eligible resources, but not at selecting the "best" among them, it does not provide means for expressing preferences. Although, only the requests may constrain the matching, we presented the D-GRDL under *symmetric* languages, because it allows to describe both the requests and the resources.

The Resource and Service Description (RSD) [BGKR98] language is a uniform approach to describe requests and resources. It features fixed properties, implicit constraints on properties of atomic parts and a means to describe complex environments and requests. RSD instances are composed of nodes, e.g., atomic parts, and edges, which are network links between the nodes. Arbitrary attributes may be associated with both the nodes and the edges. The language was designed to describe large parallel HPC applications and static meta-computers. It does not, however, support arbitrary constraints among the different parts of an application or objectives to select a set of matching resources.

7.2.4 Requirement Matrix and Use in Grid Projects

Table 7.2 summarizes the presented approaches for describing requests and resources. Additionally, the table shows the use of these approaches in recent and on-going resource management projects.

7.3 The Simple Reservation Language

Based on the requirements listed in Section 7.1 and the survey of existing description languages in Section 7.2, we propose the *Simple Reservation Language* (SRL). The basic concept is inherited from languages such as Condor ClassAds [RLS98], that is, both, the requests for co-reservations and the resources are described with the same language. The main differences to these languages are: (1) the introduction of variable properties,

[5] See Section 8.2.1 on page 62 on handling the issue of tri-state logic.

Table 7.2: Summary of the description languages with respect to the requirements and their use in resource management systems and recent projects. Abbreviations/Symbols: PRP - Properties (F - Fixed / V - Variable), CON - Constraints (I - Implicit / A - Arbitrary), OBJ - Objectives, AUX - Auxiliary parts, EASE - Ease of use, PORT - Portability, EFF - Efficiency, EXT - Extendability, UNI - Uniformity, + - positive, ○ - neutral, − - negative.

	Functional Requirements							Non-Functional Requirements					
Description Language	Atomic Parts			Multiple Parts			AUX	EASE	PORT	EFF	EXT	UNI	Projects
	PRP	CON	OBJ	PRP	CON	OBJ							
Request Languages													
RSL	F	I	no	no	no	no	no	○	+	−	+	−	GT-based
xRSL	F	I	no	no	no	no	no	○	+	−	+	−	NorduGrid
JSDL	F	I	no	no	no	no	no	−	+	−	+	−	AstroGrid-D
Unicore	F	I	no	no	no	no	no	○	○	−	○	−	EU Deisa, D-Grid
PBS/Torque	F	I	no	no	no	no	no	○	−	−	−	−	PC Clusters
Resource Languages													
MDS	F	I	no	F	I	no	yes	○	+	○	−	−	GT-based
GLUE	F	I	no	F	I	no	yes	○	+	○	−	−	GT-based
Unicore	F	I	no	no	no	no	no	○	○	−	○	−	EU Deisa, D-Grid
Symmetric Languages													
ClassAds	F	A	yes	F	A	yes	yes	+	+	○	+	+	Condor-based, EGEE
RedLine	F	A	yes	F	A	yes	yes	+	+	○	+	+	Research
GRDL	F	A	yes	F	A	yes	yes	+	+	○	+	+	LHCb Grid
RSD	F	I	no	F	I	no	yes	+	○	○	+	+	Research

(2) a more flexible way to specify constraints (like in RedLine [LF03b]), (3) a specific naming scheme for *(key,value)*-attributes and (4) a precise definition of the semantics of a core set of attributes.

As outlined in the introduction, we regard the co-reservation problem as an optimization problem. Formally, the descriptions 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 and a provider define certain attributes, i.e., the SRL vocabulary, which are transformed into the corresponding mathematical expressions. Figure 7.1 illustrates the use of the SRL. First, the consumers (users) and providers (resources) describe their requirements. Next, the descriptions are pre-processed, that is, adding implicit constraints (added red line) and adding inherited attributes.

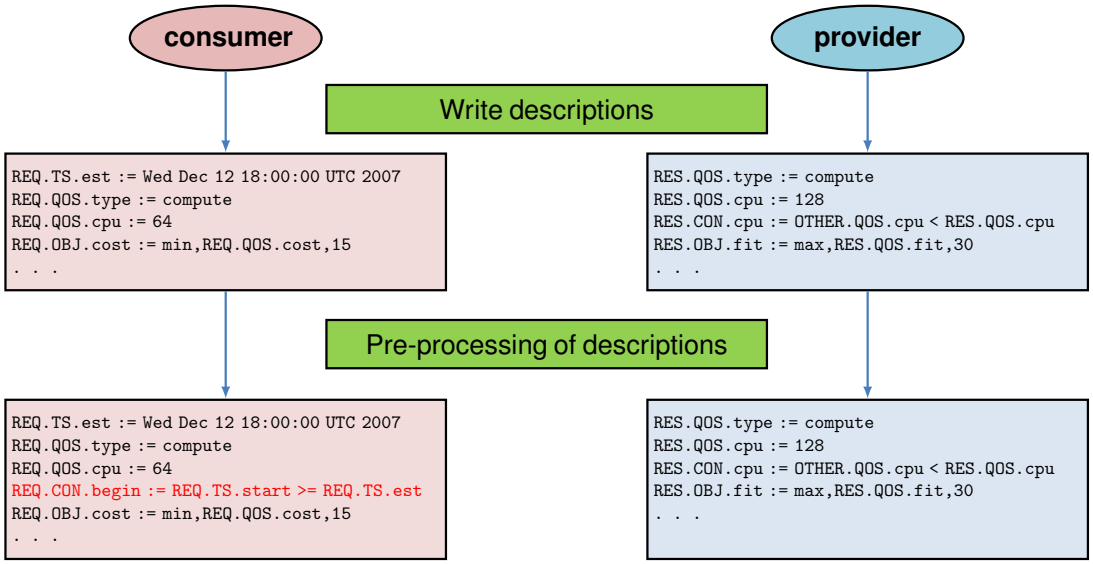

Figure 7.1: Overview of the main processing steps of SRL descriptions.

7.3.1 Syntax of the Simple Reservation Language

An SRL description consists of *(key,value)*-attributes with each key being a string of three components: an identifier `<id>`, a scope `<scope>` and a name `<name>`. Definition 22 formally introduces attributes.

Definition 22 (Attribute). *The syntax of an attribute is defined by the following EBNF.*

```
attribute   ::= key ':=' value
key         ::= id'.'scope'.'name[elem]
id          ::= namestring | 'ROOT' | 'OTHER' | '*'
scope       ::= 'TS' | 'QOS' | 'MISC' | 'CON' | 'OBJ'
name        ::= namestring
```

```
elem        ::= '{'ak'}' | '[1]' | '[2]'
value       ::= scopestring
namestring  ::= letter { numlet }
ak          ::= string
scopestring ::= datespec | qosspec | miscspec | constraint |
                objective
```

The alternatives of `scopestring` *are defined in Def. 23, 24, 25, 26 and 27.* ♢

The identifier `<id>` of an attribute names the part of the entity[6]. Thus, it is possible to reference attributes of a specific part from within any other part. The identifier `ROOT` refers to global attributes. The identifier `OTHER` references an attribute of the matching party. Using the identifier `*` with a certain scope `<scope>` and name `<name>`, the expression `*.scope.name` evaluates to the set of attribute keys of an entity, where the scope and the name of each element of the set matches the scope `<scope>` and the name `<name>`.

Each scope denotes a specific group of attributes. We distinguish three kinds of scopes, based on its appearance in a *(key,value)*-attribute. Scopes which can appear in the key are *left-hand side* scopes. The ones which can appear in the value are *right-hand side* scopes. Scopes which may appear on both sides are *left-/right-hand side* scopes. The scopes defined in the SRL, are shown in Table 7.3. Global attributes of the scopes `TS`, `QOS` and `MISC` are inherited by all parts of an entity. If a part requires a different value of the attribute, it may overwrite it by using the same scope and name in the key.

Table 7.3: Attribute scopes of the Simple Reservation Language.

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

The name of an attribute is an alpha-numeric string. For each scope there exist several keys with a pre-defined semantics which we will present in Section 7.3.2. Apart from the pre-defined names, any other string may be used, but its semantics is only defined by the described relationships to other attributes.

The optional extension to a name – `{ak}` or `[1/2]` – is used to reference named fractions of the value or elements of a pair. The fraction is named by the string `ak`.

[6]We use the term *entity* if a statement applies to both, the resources and the requests.

The first element of a pair is referenced with the extension `[1]`, the second with `[2]`. The non-terminals `letter` and `numlet` refer to any letter (a-z and A-Z) and any digit, letter and underscore, respectively.

Definition 23 (Temporal Scope). *We define the syntax of* `datespec` *– the value domain of the temporal scope – as follows*

```
datespec  ::=  datespec aop datespec | '(' datespec ')' |
               absdate | '+' reldate | duration | key
aop       ::=  '+' | '-' | '*' | '/'
absdate   ::=  epochseconds | UTCstring
reldate   ::=  posint timeunit {':'  reldate}
duration  ::=  posint timeunit {':'  duration}
timeunit  ::=  'd' | 'h' | 'm' | 's'                    ♢
```

The non-terminal `epochseconds` specifies the date as seconds since the epoch, i.e., defined as "00:00:00 1970-01-01 UTC" with UTC being the *Coordinated Universal Time*. `UTCstring` is used to define the date in a more natural, i.e., human readable, format.

The relative date (cf. non-terminal `reldate`) begins with a plus sign and may be given by a colon separated list of days, hours, minutes or seconds (cf. non-terminal `timeunit`). All these parts can take any positive integer (cf. non-terminal `posint`). During the pre-processing of descriptions, the relative dates and the durations are translated into epoch seconds and seconds, respectively.

Definition 24 (Quality-of-Service Scope). *The syntax of* `qosspec` *– the value domain of the quality-of-service scope – is defined as follows*

```
qosspec ::= qosspec aop qosspec | (' qosspec ')' | restype |
            aarray | pvector | vstring
restype ::= 'compute' | 'storage' | 'network' | 'data' |'any'
aarray  ::= aelem { ':'  aelem }
pvector ::= pair { ':'  pair }
aelem   ::= vstring '=>' vstring
pair    ::= vstring '/' vstring                         ♢
```

The non-terminal `restype` specifies one of the supported resource types. The non-terminal `vstring` may be any string excluding line feeds.

Definition 25 (Miscellaneous Scope). *The syntax of* `miscspec` *– the value domain of the miscellaneous scope – is defined as follows*

```
miscspec    ::= vstring                                 ♢
```

The non-terminal `vstring` may be any string excluding line feeds.

Definition 26 (Constraint Scope). *The syntax of* `constraint` *– the value domain of the constraint scope – is defined as follows*

```
constraint  ::= boolexpr | 'not' boolexpr |
                boolexpr boolop boolexpr
boolexpr    ::= expression op expression
expression  ::= key | sop key | number
sop         ::= 'sum' | 'min' | 'max' | 'prod'
op          ::= '==' | '!=' | '>' | '>=' | '<=' | '<' | in
boolop      ::= 'and' | 'or'                                    ♢
```

The non-terminal `number` may be any number, i.e., integer, real, boolean. Note, the operators `!=`, `>`, `<` and `in` are primarily used in the matching of static requirements (cf. Chapter 8). If they may be used for specifying dynamic requirements depends on the employed optimization techniques (cf. Chapter 10).

Definition 27 (Objective Scope). *The syntax of* `objective` *– the value domain of the objective scope – is defined as follows*

```
objective   ::= sense ',' objexpr ',' weight
sense       ::= 'min' | 'max'
objexpr     ::= key | number | objexpr aop objexpr
aop         ::= '+' | '-' | '*' | '/'                           ♢
```

The non-terminal `weight` may be any real number. The non-terminal `number` may be any number, i.e., integer, real, boolean.

Example 7.5 (Attributes)
The following eight attributes illustrate the use of the SRL for describing a co-reservation request containing two parts `REQ1` *and* `REQ2`*.*

```
REQ1.TS.est   := Tue Dec 12 18:00:00 UTC 2006
REQ1.TS.dur   := 21600
REQ2.TS.let   := Fri Dec 15 18:00:00 UTC 2006
REQ1.QOS.type := compute
REQ1.QOS.cpu  := 16
ROOT.CON.cost := sum *.MISC.cost <= 10000
REQ2.CON.time := REQ1.TS.start == REQ2.TS.start
REQ1.OBJ.cost := min,REQ1.MISC.cost,8
```

The attributes `REQ1.TS.est` *and* `REQ1.TS.dur` *define the earliest start time and the duration of part* `REQ1`*, respectively. The attribute* `REQ2.TS.let` *sets the latest end time, i.e., deadline, of the part* `REQ2`*. The attributes* `REQ1.QOS.type` *and* `REQ1.QOS.cpu` *define the type of the requested resource and the amount to be reserved, respectively. A global constraint on the total cost is given with the attribute* `ROOT.CON.cost`*. The attribute* `REQ2.CON.time` *defines a constraint on the start times of the parts* `REQ1` *and* `REQ2`*. Note, it does not matter which identifier is used for the key of a constraint, because the optimization problem combines all constraints in a single conjunction. The last attribute defines an optimization criteria of the part* `REQ1`*.*

7.3.2 Pre-defined Types and Attributes

The Simple Reservation Language defines a small set of common attributes and assigns a specific semantics to them. Thus, the writing and processing of SRL descriptions is simplified and less prone to ambiguous interpretations. In the following, we describe the pre-defined attributes of the scopes `TS`, `QOS` and `MISC`. Note, there exist no pre-defined attributes of the scopes `CON` and `OBJ`.

Temporal Scope `TS`

The scope `TS` contains six pre-defined attribute keys – x`.TS.est`, x`.TS.dur`, x`.TS.durref`, x`.TS.let`, x`.TS.start` and x`.TS.end` – where x represents any part of a description. The semantics of these keys is as follows.

The earliest start time of a reservation is given by the value of x`.TS.est`. The latest end time of a reservation is given by the value of x`.TS.let`. The corresponding attributes are used in the probing and candidate selection steps. For the latter step, the attributes are translated into constraints, i.e., x.`TS.est` $\leq$ x.`TS.start` and x.`TS.let` $\geq$ x.`TS.end`.

The actual start and end time of a reservation are denoted by the keys x.`TS.start` and x.`TS.end`, respectively. Note, these attributes are typically set in the candidate selection step, such that they do not violate the constraints on the earliest start time and the latest end time. They may, however, also be fixed in the description of a request or a resource.

The value of the key x.`TS.dur` defines the actual duration of the reservation. It may be fixed to a certain value or a function of parameters such as QoS attributes. The value of the key x.`TS.durref` defines the duration with respect to reference QoS attributes. This attribute may be used to simplify the specification of moldable requests, whose duration depend on the acquired QoS level.

Quality-Of-Service Scope `QOS`

The scope `QOS` has one common pre-defined attribute key – x.`QOS.type` – and several type-dependent pre-defined keys.

The values of the key x.`QOS.type` are `compute`, `storage`, `network`, `data` and `any`. In Table 7.4, we list the type-dependent pre-defined attribute keys for these values (except for `any`). The type `any` may be used for a service whose type is not among the ones listed in Table 7.4. For example, the set of attributes can easily be extended to reserve more abstract resources such as web server capacity or database querying capacity by introducing attributes such as x.`QOS.webresponsetime` or x.`QOS.dbmsquerytime`, respectively.

For each resource type – except `network` – we define two attributes `domain` and `url` which correspond to the IP domain the resource is associated with and the service which manages the resource, respectively. Since a network link connects two resources,

we define attributes for both 'sides' of it – `domainleft` and `domainright` as well as `urlleft` and `urlright`.

Table 7.4: Pre-defined attribute keys (`name` in x.`QOS.name`) for the several service types.

Name	Semantics	Value Format
		`compute` *type*
`arch`	CPU architecture	String
`os`	operating system	(`string/version number`)-pair
`swenv`	software environment	Vector of (`string/version number`)-pairs
`np`	number of CPUs	Positive integer
`nplb`	lower bound on `np`	Positive integer
`npub`	upper bound on `np`	Positive integer
`npref`	reference `np`	Positive integer
`perf`	CPU performance	(`benchmark name/benchmark value`)-pair
`perfref`	reference CPU perf.	(`benchmark name/benchmark value`)-pair
`spm`	speed-up model	String, e.g., `amdahl`, `downey`, `gamma`
`spp`	speed-up parameter	`aarray` (cf. Def. 24) mapping strings to numbers, which are accessed with the syntax x.`QOS.spp{str}`
`ram`	RAM memory	Positive integer plus a byte unit (MB, GB, ...)
`disk`	disk space	Positive integer plus a byte unit (GB, TB, ...)
		`storage` *type*
`disk`	storage space	Positive integer plus a byte unit (GB, TB, ...)
`bwmax`	max bandwidth	Positive integer plus a byte/time unit (GB/s, TB/s, ...)
		`network` *type*
`bwmax`	max bandwidth	Positive integer plus a byte/time unit (MB/s, GB/s, ...)
`bwavail`	available bandwidth	Positive integer plus a byte/time unit (MB/s, GB/s, ...)
`latency`	latency	Positive integer plus a time unit (μs, ms, s, ...)
		`data` *type*
`lfn`	logical file name	`URL` pointing to a replica catalog
`pfn`	physical file name	`URL` of the location from where the file can be obtained
`size`	size of the file	Positive integer plus a byte unit (MB, GB, ...)

Miscellaneous Scope `MISC`

The scope `MISC` contains four pre-defined attribute keys – x.`MISC.owner`, x.`MISC.vo`, x.`MISC.serviceurl` and x.`MISC.reserve`. The key x.`MISC.owner` is used to associate an owner with a description, e.g., a user for a request or an institute name with a resource. The owner may be identified by several means such as a local UNIX account, an X.509

certificate, etc. The key x.MISC.vo associates a virtual organization with a request or a resource. The key x.MISC.serviceurl is used for identifying the service which manages the access to a resource, e.g., a local resource management system or a Globus GRAM interface. The key x.MISC.reserve is used to specify if the request part is to be reserved or not. This feature may be used for combining reservations with best-effort allocations, for specifying parts which are already reserved or simply for specifying parts which may not be reserved.

7.3.3 Pre-processing SRL Descriptions

Pre-processing is used to transparently enable complex relationships among attributes and to set default values if applicable. Thus, writing a request or resource description is further simplified. We distinguish three kinds of pre-processing operations: (1) setting a default value, (2) adding constraints and (3) adding variable properties. For example, if a request part does not specify an earliest start time (x.TS.est), the attribute is set to the current time. Examples for adding constraints were discussed above (cf. Section 'Scope TS'). The pre-processing operations for the scope TS are summarized in Table 7.5.

Table 7.5: Pre-processing operations for attribute keys of the scope TS.

Attribute	Pre-processing Operation	
x.TS.est := val	add constraint	x.CON.tsest := x.TS.start >= val
x.TS.let := val	add constraint	x.CON.tslet := x.TS.end <= val
x.TS.dur := $expr$	add constraint	x.CON.tsdur := x.TS.start + x.TS.dur == x.TS.end

In the scope QOS, pre-processing is only applied to attribute keys of the service type compute. The pre-processing operations concerning the scope QOS and service type compute are summarized in Table 7.6.

Table 7.6: Pre-processing operations for attribute keys of the scope QOS.

Attribute	Pre-processing Operation	
x.QOS.nplb := val	add constraint	x.CON.qosnplb := x.QOS.np >= val
x.QOS.npub := val	add constraint	x.CON.qosnpub := x.QOS.np <= val

Finally, there are a few pre-processing operations which involve attributes of both the scope TS and the scope QOS. These operations define a relationship between the duration and the QoS-level of a reservation. In the following, we present two operations concerning the execution time of a parallel program that could be run on the reserved resources. The operations depend on the used speed-up model, e.g., Amdahl, and if

we assume homogeneous or heterogeneous site characteristics, i.e., processor characteristics. Note, similar operations may be defined for network connections if they are used to transmit an a-priori known data volume.

If Grid resources possess processors with the same performance, the pre-processing operation applies Amdahl's law (Eq. (4.3)) and adds the variable property

```
x.TS.dur  :=    x.TS.durref
              *  (x.QOS.spp{seq} + x.QOS.spp{par} / x.QOS.npref)
              /  (x.QOS.spp{seq} + x.QOS.spp{par} / x.QOS.np)
```

In a Grid using heterogeneous processors, the pre-processing operation applies the adapted Amdahl's law (Eq. (4.6)) and adds the variable property

```
x.TS.dur  :=    x.TS.durref
              *  (x.QOS.perfref[2] / x.QOS.perf[2])
              *  (x.QOS.spp{seq} + x.QOS.spp{par} / x.QOS.npref)
              /  (x.QOS.spp{seq} + x.QOS.spp{par} / x.QOS.np)
```

Similar operations can be defined for other speed-up models, e.g., Downey or Gamma.

7.3.4 Evaluation of the Simple Reservation Language

We evaluate the Simple Reservation Language (SRL) with respect to the functional and non-functional requirements.

Functional Requirements. The SRL supports fixed and variable properties of individual parts and an entity as a whole (via identifier `ROOT`). Furthermore, constraints and objectives may be specified on the same granularity. Auxiliary parts are supported through the attribute key x.`MISC.reserve`. Finally, allowing references to attributes of any part within an entity itself and to the matching party, provides a sufficient degree of expressiveness. It may be further enhanced by introducing higher-level constructs such as for-loops. Hence, the SRL satisfies the functional requirements listed in Section 7.1.1.

Non-Functional Requirements. The SRL is a uniform approach to describe all entities involved in managing co-reservations. The language is portable, because the small set of syntax rules and the use of pure ASCII characters enables its use on any system environment. Because the language does not restrict the definition of attribute names (except for those with pre-defined semantics) and supports any resource type with the type `any`, it can easily be extended to describe parts belonging to none of the pre-defined types. The efficient processing of SRL instances is facilitated by the objectives for selecting a co-reservation candidate. Finally, the syntax, in particular, the use of the scopes, the provision of attributes with pre-defined semantics and the corresponding pre-processing operations ease the use of the language.

7.4 Summary

While we introduced attribute keys with pre-defined semantics, we were, intentionally, not providing nor requesting any guidelines on writing descriptions. Especially, we did not classify certain attributes as mandatory or optional. Clearly, the more information a description contains, the easier a resource manager can process it. Leaving important information, such as the type of a part, unspecified, may result in a less efficient processing, due to a larger search space, or undesired results, because more candidates match the description. The language also provides a powerful feature for new users or resources to test their description without actually acquiring any reservation. If the attribute `reserve` is set to false, the whole processing is performed except for the step which actually secures a reservation.

Note, we did not introduce specific operators for non-numeric attributes such as the software environment, the operating system or the architecture. Constraints on these attributes may use the usual comparison operators !=, ==, >=, >, < and >=. Note, the applicable operators used in the optimization step (cf. Chapter 10) may depend on the capabilities of the solver. For example, standard LP solvers only support the use of ==, >= and <=.

Chapter 8

Finding Eligible Resources

Upon reception of a co-reservation request the *Grid Reservation Service* (GRS) determines eligible resources taking the requirements of the request and the resources into account. That is, the GRS matches descriptions of atomic request parts with descriptions of resources and vice-versa. The actual matching operation is carried out by querying the *Resource Catalog* (cf. step ② in Fig. 6.2 on page 38). The query considers the static requirements only. Formally, the matching can be modeled as a constraint satisfaction problem, where the variables of one side are filled in with the characteristics of the other. For example, to find compute resources operated under Linux a request may write the following constraint in SRL notation

```
IBM.CON.os := OTHER.QOS.os == "Linux".
```

Likewise, a resource may constrain the requests by the affiliation of the user as in

```
RES.CON.aff := OTHER.MISC.owner in {AstroGrid-D, C3-Grid}.
```

The broker component of many of todays Grid resource management systems apply constraint satisfaction mechanisms to match static requirements of requests and resources. Therefore, we will give an overview of the existing approaches and evaluate them with respect to their applicability in CORES. The main differences between resource brokerage in todays Grid environments and CORES are: (1) resource brokerage effectively ends, while CORES starts with the matching step, and (2) resource brokerage must determine a single matching resource, while CORES derives a list of candidate resources. In Chapter 10, we will see, however, that the matching step is also required to filter the resources further in order to shrink the size of the search space.

The contributions of this chapter are the requirements on determining eligible resources (Section 8.1) and the presentation of existing approaches on matching descriptions and resource discovery (Section 8.2).

8.1 Requirements

Naturally, a mechanism for matchmaking atomic requests with resources shares some of the requirements of the Simple Reservation Language (SRL, cf. Chapter 7). More

precisely, a matchmaking procedure must answer the questions – from the viewpoint of the user

- Who is requesting a reservation?, *and*
- What types of resources and which (maximum) quantity *or* Quality-of-Service of each resource shall be reserved?,

and from the viewpoint of the resource provider

- Who is allowed to acquire reservations?, *and*
- What type of service and which maximum service level (Quality-of-Service) does the resource provides?

In the following, we introduce requirements regarding the representation of static information, symmetric matching and the types of constraints.

Representation of Static Information. Static information is represented in several different types – numeric, textual and mixed/composite – and units – megabytes, gigabytes, terabytes, ..., Mbit/s, Gbit/s, etc. Thus, the matchmaking must be capable of handling different representation types and of converting the values of attributes. For example, a compute resource may specify the available main memory as 8 GB ($8 \cdot 2^{30}$ bytes), but the user requests 1024 MB ($1024 \cdot 2^{20}$ bytes) by writing the SRL constraint

```
REQ.CON.MINMEM := OTHER.QOS.ram >= 1024 MB.
```

Before the matchmaking procedure can compare the pure numbers they must be converted into the same units, e.g., gigabytes, megabytes or any other common base unit for the size of the memory. Also, the matchmaking mechanism must be aware of relationships between certain terms in a domain – e.g., Linux is some form of UNIX which in turn is an operating system – and shall handle composite expressions appropriately – e.g., software requirements such as *zlib 1.1.4 or newer*.

Symmetric Matching. The matchmaker must support symmetric matching, by ensuring that the constraints of both the requesters and the resources are satisfied. For example, a resource may limit access to its reservation facility by writing the SRL constraint

```
RES.CON.VO := OTHER.MISC.vo in \{AstroGrid-D, C3-Grid\}.
```

Thus, all pairs $(request, resource)$ match each other if they satisfy both the constraints `RES.CON.VO` and `REQ.CON.MINMEM`.

Equality Constraints. An equality constraint is used to restrict a property of the matching candidate to a single value. For example, if an application requires a specific number of processors, the client would write the SRL constraint

```
REQ.CON.NUMCPU := OTHER.QOS.np == 64.
```

The matchmaker must support the comparison operator ==.

Inequality Constraints. An inequality constraint is used to bound a property of the matching candidate. The earlier example of requiring a minimum memory size of 1024 MB implements an inequality constraint. Similarly, a resource may specify its maximum number of requested processors by the SRL constraint

```
RES.CON.JOBSIZE := OTHER.QOS.np <= 128.
```

The matchmaker must support the following comparison operators: <= (less than or equal), < (less than), != (not equal), > (bigger than) and >= (bigger than or equal).

Membership Constraints. Often several non-contiguous values may be acceptable for an entity. Because the acceptable values are non-contiguous, inequality constraints may not be used for modeling such requirements. Membership constraints require that the property of an entity is a member of a set. The constraint `RES.CON.VO` restricting the access to a resource to users from two virtual organizations constitutes a membership constraint. The matchmaker must support the membership operator `in`.

Constraints on Non-numeric Properties. Frequently, constraints are given for non-numeric properties such as the architecture of a processor, the operating system of a compute resources or its software environment. First, the matchmaker must ensure that the values of such properties are converted into a common vocabulary. For example, if a user requires a compute system running the operating system `Linux 2.6.16`, but the resource uses all letters in upper case `LINUX 2.6.16` some transformation is needed to enable matchmaking. Second, the matchmaker must use an appropriate representation of the domain of the properties to implement the correct semantics of the comparison operators. For example, if an application requires a compute resource with the `Linux` operating system, the following constraint could be defined

```
REQ.CON.OS := OTHER.QOS.os == Linux.
```

All resources operating under any version of Linux satisfy this constraint, but not those running Windows, Mac OS, AIX, etc.

Complex Constraints. In many situations, basic constraints containing only a single comparison or membership test are not sufficient. Therefore, the matchmaking mechanism must handle complex constraints composed of simple ones by using the standard boolean operators `and` and `or`. Note, in some cases constraints on mixed/composite attributes may be modeled by complex constraints. For example, the previously mentioned software requirement *zlib 1.1.4 or newer* could be written as

```
REQ.CON.SW := zlib in OTHER.QOS.swenv and OTHER.QOS.zlib >= 1.1.4,
```

where `OTHER.QOS.swenv` is a set of software packages provided by a resource.

Unsupported SRL Features. Because, the matchmaker tries to find eligible resources for each atomic request relationships between requests need not to be considered. Also, variable properties such as the future availability or price of a resource are not taken into account by the matchmaker unless these properties are static.

8.2 State of the Art

We discuss prominent approaches to the problem of matching requests to resources and vice-versa. In general, the problem is solved by addressing (1) the description of requests and resources, and (2) the actual matching of these descriptions. We already discussed the state of the art of description approaches in Section 7.2. Here, we focus on the matching mechanisms. Specifically, we discuss matchmaking in Condor [LLM88] and derived systems, matchmaking in todays Grid resource managers and ontology-based resource discovery.

8.2.1 Matchmaking in Condor and Derived Systems

Condor is a resource management system for clusters built from workstations or dedicated machines. The core of Condor is its very flexible resource and request description mechanism *Condor ClassAds* (cf. Section 7.2.3). In Condor, matching two entities is a symmetric process. In particular, both entities may specify requirements and ranking criteria. The actual matchmaking procedure [Ram01] works in two phases – the *setup* phase and the *match* phase. In the setup phase, the algorithm analyzes the constraints to determine sets of external references, i.e., referenced attributes of the entity being matched. Then, these sets are used to convert the representation of offers to rectangles. The setup phase is completed by aggregating, indexing and storing these rectangles. In the match phase, the following steps are performed.

1. The constraints are converted to the rectangle representation.
2. The matching resources are determined by a window query.
3. The highest ranked resource is selected and removed from the index.

Because of the generic model Condor ClassAds builds upon, Condor matchmaking satisfies most of the requirements listed in Section 8.1. The only missing feature is the ability to reason about non-numeric or composite properties, i.e., *Linux is a type of Unix* and the like.

The originally bi-lateral matchmaking of Condor has been extended by gang matching to support complex applications [RLS03]. Other improvements to Condor ClassAds and its matchmaking include the application of constraint satisfaction techniques by Liu and Foster [LF03b] and the use of set theory by Stokes-Rees [SR06]. By modeling the matchmaking as constraint satisfaction problem, Liu and Foster extended the expressiveness of ClassAds with *RedLine* [LF03b]. Similar to gang matching [RLS03], RedLine supports matching of complex application requests to multiple resources. That is, the matching is only successful if all parts are satisfied. Additionally, RedLine facilitates the querying of policies and supports different levels of information representation. Policies of interest are, for example, when resources are available or what capacity may be used. Different levels of information representation cope with the problem that users as well as providers may describe their requests and offers with textually different but conceptually convertible terms, e.g., Linux is a kind of UNIX. The prototype of RedLine exploits node consistency to reduce the size of univariate domains and uses backtracking to solve the remaining constraint satisfaction problem.

In his thesis [SR06], Stokes-Rees mainly tackles two major issues of Condor ClassAds and matchmaking: (1) tri-state logic and (2) scalability. While the first issue may arise in any environment where some information being referenced by one party is missing in the descriptions of the other party, the second issue is specific to infrastructures constructed by large virtual organizations such as the LHCb experiment [LHC08] of the *Large Hadron Collider*. Stokes-Rees takes the unique approach of applying *set theory* to the matchmaking problem. The first issue – tri-state logic – is solved by introducing *requirements*. Requirements specialize characteristics by associating matching operators with them. In brief, a characteristic satisfies a requirement if the characteristic's value set is a subset of the range of a requirement. The tri-state logic issue is removed by defining the range of a requirement with unspecified values simply as the set containing the entire value space of the requirement's dimension. The second issue is addressed by introducing *resource templates* which facilitate the clustering of resources with similar characteristics and thereby support an efficient matching of requests and resources. Resource templates are implemented by using partial matching operators. The rationale behind this is that the particularities of many Grid resources are not important in distributing high-throughput computing jobs.

8.2.2 Matchmaking Mechanisms in Grid Resource Discovery

In many Grid environments, resource discovery builds upon Condor matchmaking – the most known are the EGEE Workload Management System [EGE08] and Condor-G [FTL+01]. The GridWay meta-scheduling framework [Tea07b] provides job description means similar to Condor ClassAds. In particular, it lets users specify a *require-*

ment and a *rank* expression. The actual matchmaking mechanisms employed by GridWay are, however, only briefly described in a few documents [HML03, Tea07a]. In early versions of GridWay [HML03], the resource discovery is implemented by querying instances of the LDAP-based Globus Toolkit Monitoring and Directory Service (GT MDS). In more recent versions [Tea07a], GridWay supports several information drivers which adapts to different information services. Furthermore, new service directories may be integrated by developing new drivers. Nimrod/G [BAG00] provides tools for managing distributed parameter sweep simulations. Particularly, it contains a scheduling component which distributes workload to idle compute resources in a Grid. The actual resource discovery is implemented by using XPath[1] to query the latest versions of the GT MDS. Also, the GridBus broker [VBW04] queries information services such as GT MDS to discover suitable resources for a job. The *Broker Module* of the GridLab Resource Management System (GRMS) [GRM05] encapsulates the matchmaking mechanism which uses the information gathered through the *Resource Discovery Module* (RDM). While the RDM is currently bound to the GT MDS too, the GRMS architecture, like GridWay, allows to integrate other information services without interfering with the actual matchmaking procedure. The Grid Workflow Execution Service (GWES) [Hoh07] maps requests written in the D-GRDL [Wol07] to XQuery[2] and executes them on an XML database storing the descriptions of the resources. GridARM [SF05], the resource management system of Askalon [FPD+05], provides a flexible mechanism for resource discovery. Its request-resource correlator (RRC) employs an ontology engine (OE) and a resource discovery (RD) module for matching requests and resources. First, the OE transforms the request into resource filters by decomposing complex requests into smaller parts and adapting them to the query language supported by the available resource registries. Then, these filters are applied on the resource registries, e.g., GT MDS, and the results are congregated.

8.2.3 Ontology-based Matching

Often, resource properties are encoded with strings such as "`Linux`" for the operating system, "`IA64`" for the architecture of a processor or "`compute`" for the type of the resource. The domain of a single property can be structured by an ontology which allows to reason about relations between the elements of the domain. For example, "`Linux Kernel 2.6.18`" belongs to the kernel version 2.6 of the Linux operating system. Generally, ontologies can be represented with graphs, where nodes represent classes of elements. Assuming such structure is given, semantic web technology may be used to determine if two expressions are in the requested relation (==, <, >, ...) or not.

In [TDK03], Tangmunarunkit et al. apply semantic web technology on the resource matching problem. Their approach builds on ontologies for defining vocabularies to describe the properties of requests and resources. The actual matchmaking procedure

[1] XPath is the W3C recommendation of the XML Path Language (http://www.w3.org/TR/xpath).
[2] XQuery is the W3C recommend. of the XML Query Language (http://www.w3.org/TR/xquery).

is defined through inference rules which define conditions on matchings. According to [TDK03], the main features of a matchmaker are the support of *bilateral constraints*, the ability to describe *matching preference* and the support for *multi-lateral matchmaking* and the application of *integrity checking*. The former two features are readily supported by approaches such as Condor ClassAds and the like. Although co-reservations require multiple resources, multi-lateral matchmaking is not sufficient in a co-reservation context. Multi-lateral matchmaking only covers the second step of the co-reservation mechanism, i.e., the discovery of eligible resources for each part of the co-reservation (cf. step ② of Fig. 6.2 on page 38). The actual matching or assignment of parts to resources is performed in the step determining the best co-reservation candidate (cf. step ④ of Fig. 6.2 on page 38). In [TDK03], the matchmaking is performed by the following four actions.

1. The resource providers periodically advertise their resources sending an *advertisement* message described in the resource's ontology vocabulary to the matchmaker.

2. The clients describe their requests using the request's ontology vocabulary and send them to the matchmaker too.

3. The matchmaker applies the inference rules to all descriptions.

4. The matchmaker returns the ordered matching pairs to the client.

8.3 Summary

Table 8.1 summarizes the existing matching approaches. Interestingly, none of the Grid resource discovery mechanisms supports *symmetric* matching. Furthermore, they only provide basic mechanisms for information representation (due to limited sets of attributes) and for specifying non-numeric and complex constraints. A notable exception is the D-Grid Resource Description Language [Wol07] which provides similar capabilities as the non-Grid approaches. All approaches of the categories *Condor and derived systems* and *Ontology-based matching* support symmetric matching and richer capabilities than most of the discussed Grid resource discovery mechanisms. From these approaches, the Grid Resource Description Language (GRDL) [SR06] and the mechanism proposed by Tangmunarunkit [TDK03] implement all the requirements. Whether they may be used, however, also depends on the actual environment and the availability of implementations. For example, the software provided with [TDK03] is no longer maintained and the resource broker developed for [SR06] was not publicly released. Thus, Condor matchmaking [Ram01] or the D-GRDL [Wol07] may be used, albeit they do not fulfill all requirements.

Table 8.1: Comparison of the matching approaches. Abbreviations/Symbols: sym - symmetric matching, req - considers request's requirements only; ++ - requirement very well supported, + - requirement well supported, ∘ - requirement basically supported, ? - no precise information available *or* not applicable.

Tool / Paper	Requirements							Matching Method
	Information Representation	Matching Type	Equality Constraints	Inequality Constraints	Membership Constraints	Non-numeric Constraints	Complex Constraints	
Condor and derived systems								
Condor	+	sym	++	++	++	∘	++	interval comparison
RedLine	++	sym	++	++	?	∘	++	constraint satisfaction
GRDL	++	sym	++	++	++	++	++	set operators
Grid resource discovery								
GridWay	∘	req	+	+	?	∘	∘	LDAP, XPath
Nimrod/G, GridBus	∘	req	+	+	?	∘	∘	XPath
GRMS	∘	req	+	+	?	∘	∘	LDAP
GWES/D-GRDL	++	req	++	++	?	++	++	XQuery
Askalon/GridARM	∘	req	+	+	?	∘	+	LDAP
Ontology-based matching								
Tangmunarunkit	++	sym	++	++	?	++	++	inference rules

Chapter 9

Determining Reservation Candidates

Possessing detailed information about the future status of the resources facilitates (1) an efficient processing of reservation requests, (2) the managing of non-HPC metrics such as cost and (3) the expression of resources' preferences among the reservation candidates. In specific environments, it may be possible to address these goals individually. In versatile Grid infrastructures, however, resource mechanisms must cope with multiple – often conflicting – goals. For example, a simple heuristic may be to use the latest possible start time for placing a reservation. While this heuristic may yield good results with respect to the reservation success rate, application centric goals – minimal finish time or cost – and resource centric goals – maximal utilization or least impact on waiting jobs – will only be met by accident. The approach of CORES for *probing* the future status of resources – i.e., determining reservation candidates – is guided by three key decisions:

D1 – *the reservation requests are moldable*,

D2 – *the resource providers calculate the candidates by themselves*, and

D3 – *the reservation system may interpolate intermediate candidates*.

By enabling moldable requests (D1), as shown in Fig. 9.1 and Table 9.1, CORES facilitates the efficient processing of requests – compared to a *trial-and-error* scheme. Letting the providers calculate the reservation candidates (D2) allows to integrate arbitrary properties, e.g., reservation costs, cancelation fees, etc., and acknowledges the autonomy of the resources. Using interpolation for determining intermediate candidates (D3) is a means to reduce the overhead for determining candidates.

Chapter Outline. Section 9.1 lists the requirements of determining reservation candidates. Section 9.2 discusses the state of the art in predicting the future state of resources. Section 9.3 presents several distributions of time-qos-slots. Thereafter, we develop methods for deriving the future status of resources (cf. Section 9.4). In Section 9.5, we briefly describe mechanisms for determining intermediate reservation candidates. We provide extensive experimental results in Section 9.6 and close the chapter with a summary in Section 9.7.

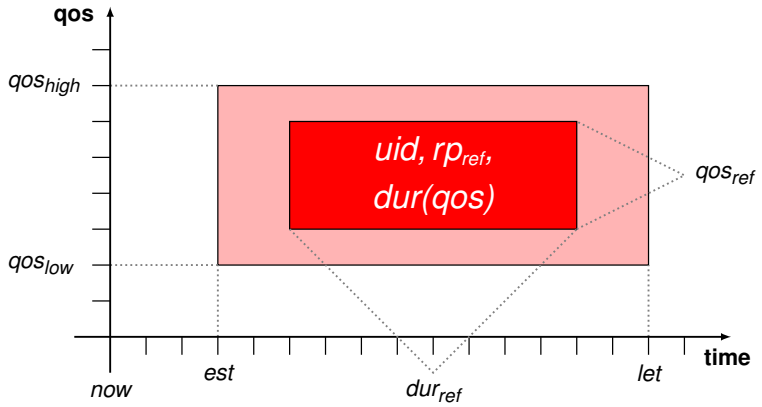

Figure 9.1: Moldable reservation request for a single resource. The parameters are described in Table 9.1.

Table 9.1: Parameters of a moldable reservation request.

Name	Description	Example
est	earliest start time	`2007/01/20 06:00`
let	latest end time	`2007/01/22 20:00`
dur_{ref}	reference duration in seconds	`7200`
qos_{low}	lower bound on the QoS	`2`
qos_{high}	upper bound on the QoS	`7`
qos_{ref}	reference QoS	`3`
uid	requester identification	`XerTWQ4`
rp_{ref}	reference resource performance	`SPECint2000/1500`
$dur(qos)$	QoS-dependent duration	`model=>amdahl:seq=>0.1:par=>0.9`
$prop$	list of properties	`cost, cancelation, fitness`

9.1 Requirements

Deriving reservation candidates must satisfy several goals depending on the point of view of a stakeholder.

R1 – Information for Efficiently Processing Requests

The reservation system desires information that enables an efficient processing of reservation requests. For example, Fig. 9.2 shows a compute resource with running jobs (green boxes marked RJ_1, RJ_2 and RJ_3) and a previously granted reservation (violet box marked RSV_1). For a given moldable reservation request (with a minimum of two

processors), selected time-qos-slots[1] are shown with boxes in red. It is easy to see, that no candidate may begin before the left-most red box. Without that information, a reservation system would need multiple tries to find an available slot. Especially, if it tries to acquire the earliest possible time-qos-slot.

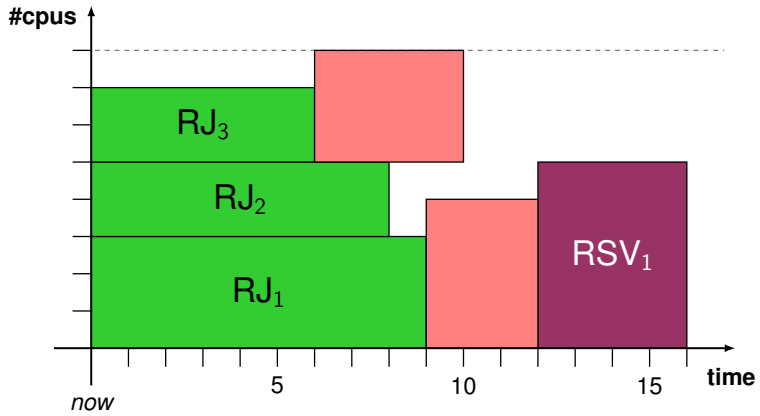

Figure 9.2: Available time-qos-slots (red boxes) for placing a moldable reservation request.

R2 – Expression of Providers' Preferences

The providers are mainly concerned with a high utilization of their resources and a fair sharing of them by their clients. Taking the waiting jobs (yellow boxes marked WJ_1, WJ_2 and WJ_3) into account, the set of eligible slots (red boxes) may be different to Fig. 9.3. That is, the mechanisms must allow the providers to tailor the information – which slots are preferred – at their needs.

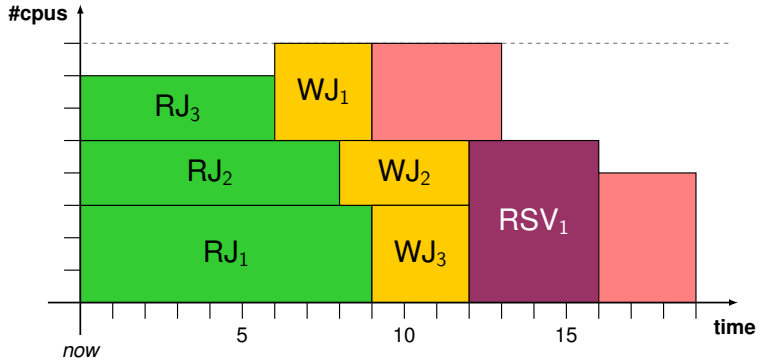

Figure 9.3: Available time-qos-slots (red boxes) for placing a moldable reservation request taking waiting jobs into account.

[1] A time-qos-slot captures the start time, the end time and the service level of a reservation candidate but its properties.

R3 – Arbitrary Properties of Time-QoS-Slots

Besides the intrinsic properties of time-qos-slots – their start time, end time and service level – a stakeholder may select slots according to other properties or criteria. For example, some slots may fit better into a schedule than others or some slots may be cheaper than others. Instead of limiting CORES to a number of currently used properties, arbitrary properties shall be supported.

R4 – Efficient Calculation of Reservation Candidates

For two reasons, the candidates itself must be computable in an efficient manner. First, a high overhead for calculating the candidates would hinder the adoption of the scheme by local resource management systems. Second, information on reservation candidates quickly ages due to the dynamic behavior of the resources in a Grid.

9.2 State of the Art

We review approaches for determining the future status of resources. Although we focus on compute resources, we will also briefly describe related work on other types of resources.

The state of a compute resource can be described by measuring the available capacity of a number of parameters such as the number of processors, main memory, disk storage, etc. Determining these parameters for future points in time is a complex task involving knowledge on the current workload, the scheduling policies, the workload that will be submitted until the requested time and arbitrary changes to the state by canceling jobs or failed resources.

In the following, we distinguish two research areas related to the subject of this chapter: (1) the prediction of job parameters or system utilization for compute resources, and (2) the prediction of workload or utilization parameters in non-compute resources.

9.2.1 Predicting the Future Status of Compute Resources

In [SFT98], Smith et al. propose mechanisms for predicting the run time of applications. First, categories of similar jobs are derived by applying genetic algorithms. The actual run time of a finished job is attached to all categories the job belongs to. Then, the run time of a new job is derived by determining the categories the job belongs to and calculating a run time and a confidence interval for each category. The run time is chosen from the category with the smallest confidence interval. The space needed for recording the run times of finished jobs can be limited. The run time predictions are used in [STF99] to predict the waiting time of a new job. The prediction calculates the schedule of the known workload plus the new job including their predicted run times. The authors study their method with different scheduling configurations. They found

that the built-in error for predicting waiting times is 34 to $43\,\%$ for least work first (LWF) scheduling and 3 to $10\,\%$ for backfill scheduling. These results were derived knowing the run time a priori. The errors are caused by jobs that have not been submitted to the system when the waiting time of a job was predicted. In the simulations without knowing the run time a priori, the error is 10 to $29\,\%$ larger for LWF scheduling and 40 to $74\,\%$ larger for backfill scheduling. Compared to the simulations with the run time estimates of the users, the error is 41 to $123\,\%$ smaller for LWF scheduling and 146 to $286\,\%$ smaller for backfill scheduling. Furthermore, they found a trend that better run time predictions lead to better predictions of waiting times. However, the best run time predictions do not automatically yield the best waiting time predictions.

In [LGTW04], Li et al. propose a mechanism for predicting the waiting time of a potential job[2]. Their mechanism is composed of two steps. First, the execution time of the job is predicted using historical data. The ratio of the average prediction error to the actual run time varies between $14\,\%$ and $35\,\%$ for different workload traces. The corresponding ratios of the user specified run time estimates to the actual run time are up to two orders of magnitude larger. In the second step, the mechanism appends the potential job to the waiting queue and simulates the schedule until the potential job is assigned a start time. An important difference between the mechanisms proposed in [LGTW04, STF99] and the requirements of CORES, is that Li's and Smith's methods derive a single value – the predicted wait time – for a given job request, while the reservation mechanism of CORES requires information on the values of a metric over the whole range $[est, let] \times [np_{low}, np_{high}]$.

Ernemann et al. [EHY02] propose a probing mechanism which considers flexible start times of compute jobs and lets the resources evaluate utility functions defined by both the requester and the resource owner. The resources' utility functions calculate the sum of the surface of the request (number of processors times the run time) and the idle surface before, after and during the candidate's execution. In other words, the utility functions determine a measure of the fragmentation generated by the additional job. The larger is the fragmentation, the lower is utility function's value. The proposed job model, however, does not support jobs waiting in a queue. Indeed, all jobs are assigned a specific start time in advance, i.e., all job requests are advance reservations. The proposed probing mechanism does not fulfill all requirements. Particularly, it does not consider flexible durations and service levels (requirements **R1** and **R3**).

Smith et al. [SFT00] study the impact of supporting advance reservations on parallel computers. The reservation request specifies a start time for which the algorithm tries to make a reservation. If no resources are available at the requested time, the local scheduler responds with a list of available time slots. Through evaluation, Smith et al. [SFT00] conclude that backfilling, stopping and restarting jobs and more accurate execution times decrease the impact of reservations on jobs. As in [EHY02], a resource does not specify its preferences for alternative start times. Moreover, stopping and restarting of jobs should not be used, because these features are not available on all systems.

[2] A *potential* job is yet to be submitted to a resource.

In [SCJG00], Snell et al. use probing to determine which time slots are available for scheduling parallel multi-site jobs. Their approach, however, lacks support for moldable requests. Also, it provides only very limited capabilities for the resources to let them express their preferences among the available time slots.

9.2.2 Predicting the Future Status in Non-compute Resources

Mechanisms for reserving resources in networks and multimedia environments have been studied extensively, e.g., Yuan et al. [YTA03], Chen et al. [CL01], Burchard [Bur04] and Nahrstedt et al. [NHK96],

Yuan et al. [YTA03] introduce a probing mechanism to efficiently determine available network characteristics (bandwidth, delay, jitter and loss) along an a-priori known path connecting an end user and a data provider. Their approach extends earlier work on reserving network resources for multimedia applications by introducing probe requests, which are flexible in a single parameter – the service level – only. CORES requires the flexibility in all three parameters – the start time, the duration and the service level. Furthermore, the support for expressing resource's preferences is very limited, because a resource may only accept or deny a specific candidate.

Chen and Lee [CL01] propose an advance reservation model for network transfers. The start time of a reservation can be chosen from a flexible interval. By booking extra resources for the length of the flexible interval, the model allows to postpone the decision on the actual start time until the flexible interval begins. Thus, it does not need to exhibit the future status of the resources. It only grants or denies requests. The future status of a resource is only determined and used internally for optimizing the schedule of requests during overloaded periods. While their approach has some merits for single resource requests, it is not applicable in the context of co-reservations. Instead of booking extra resources, a resource provider should be able to express its preferences by carefully choosing metrics' values of time-qos-slots. For example, low fitness values and high reservation costs for certain start times could indicate peak loads which are then avoided in the candidate selection phase.

In his dissertation [Bur04], Burchard exploits the *multi-protocol label switching* standard (MPLS) to facilitate advance reservations of bandwidth in computer networks. The proposed mechanisms support requests which are flexible in the start time and the service level (bandwidth). The result of the mechanisms is, however, only a single time-qos-slot that fits "best" the flexible request. While this may be enough for single resource reservations, it does not suffice for co-reservations. Also, the approach does not support arbitrary properties of time-qos-slots such as cost, fitness, etc. Time-qos-slots are determined by evaluating information about the currently active workload and the already known advance reservations. Through analytical analysis and experimental evaluation, the author discovered that arrays are better suited than trees as data structures for determining time-qos-slots.

Nahrstedt et al. [NHK96] propose a mechanism to specify the end-to-end service level in a video on demand application scenario. The achievable service level is de-

termined by processing (relatively) short videos, periodically measuring the desired service level and calculating the sustainable performance. The mechanism does not need to know the hardware and software configuration of the environment – i.e., the processing and sending capacities at the servers, the network characteristics and network load, the clients' hardware and software capabilities. Instead, it analyzes time series of measurements of the end-to-end service level at the client, for example, the frame rate of a transmitted MPEG-1 stream. However, the algorithm is based on the assumption of lightly loaded networks. Thus, it is only applicable for determining the achievable service level under ideal conditions. In contrast, a mechanism for reserving resources in advance needs information about the achievable service levels under various sub-optimal conditions.

9.3 Distributions of Time-QoS-Slots

A *time-qos-slot* is constructed from a time window $[start, end]$ and a service level qos. Time-qos-slots are often illustrated as areas in QoS-time graphs (cf. Figures 9.1, 9.2, 9.3).

Definition 28 (The Set of Time-QoS-Slots). *The set of all time-qos-slots TDQ is defined as the cross product of the sets T (start times), D (durations)[3] and Q (service levels). Formally, TDQ is defined by $TDQ = \{\langle t, d, q\rangle \mid t \in T, d \in D, q \in Q\}$.* ♢

Even for small ranges of the start time T, the durations D and the service levels Q the space of all time-qos-slots TDQ may simply be too large for calculating the properties of each element. The overhead for calculating these properties is reduced by selecting certain elements of the space. These elements form a **distribution** of time-qos-slots.

Definition 29 (Distribution of Time-QoS-Slots). *We define a distribution of time-qos-slots TDQ_{dist} as a subset of the space of all time-qos-slots TDQ, i.e., $TDQ_{dist} \subseteq TDQ$.* ♢

Notation 1 (Subsets of a Distribution of Time-QoS-Slots).
We denote the set of all service levels occurring in TDQ_{dist} by the term TDQ^{Q}_{dist} as abbreviation of the expression

$$\{q \mid \exists t \in T,\, \exists d \in D :\ \langle t, d, q\rangle \in TDQ_{dist}\}\ .$$

The set of start times with any duration but the same service level q is denoted by the term $TDQ_{dist}(\cdot, \cdot, q)$ as abbreviation of the expression

$$\{t \mid \exists d \in D :\ \langle t, d, q\rangle \in TDQ_{dist}\}\ .$$ ⋈

The distributions are determined by the resource providers when they are asked for the future status wrt. a moldable reservation request (cf. Table 9.1). Hence, the two main goals for deriving a distribution are: (1) the efficiency of its calculation and (2) the accuracy of the reservation candidates capturing the future status. We will present the following three distributions in detail:

[3] The end time of a time window $[start, end]$ is calculated as the sum of the start time and the duration.

- **corner distribution** – which only contains the extreme values of the space TDQ,
- **even distribution** – time-qos-slots are evenly distributed over the space TDQ, and
- **static workload based distribution** – most of the time-qos-slots are concentrated at a specific time of the known workload.

Note, for the sake of simplicity we omit the dimension of the duration in the following illustrations.

9.3.1 Corner Distribution

The **corner** distribution only contains the "corners" of the space TDQ.

Definition 30 (Corner Distribution). *Given a moldable reservation request (cf. Table 9.1) and a space of time-qos-slots TDQ, we define the corner distribution to include the following four tuples*[4]

$$
\begin{aligned}
cd_{L,L} &= \langle \min T, dur(\min Q), \min Q \rangle \\
cd_{L,U} &= \langle \min T, dur(\max Q), \max Q \rangle \\
cd_{U,L} &= \langle \max T - dur(\min Q), dur(\min Q), \min Q \rangle \\
cd_{U,U} &= \langle \max T - dur(\max Q), dur(\max Q), \max Q \rangle
\end{aligned}
$$

The first letter of the subscripts represents the temporal space, whereas the second letter represents the service level space. The subscript letters L and U represent the lower and upper bounds of the individual spaces. The function dur determines the required *duration at a given service level.* ♢

Example 9.1 (Corner Distribution)
A moldable request of a parallel computation is given with the following parameters: $est = 3600$ (earliest start time), $let = 39600$ (latest end time), $dur_{ref} = 1800$ (reference duration), $qos_{low} = 16$ (minimum number of processors), $qos_{high} = 128$ (maximum number of processors), $qos_{ref} = 116$ (reference number of processors) and $dur(qos) =$ `{model=>amdahl:par=>0.99}` (speed-up model). The time-qos-slots of the corner distribution are $\langle 3600, 16 \rangle$ ($cd_{L,L}$), $\langle 3600, 128 \rangle$ ($cd_{L,U}$), $\langle 37800, 16 \rangle$ ($cd_{U,L}$), and $\langle 39156, 128 \rangle$ ($cd_{U,U}$).

While the corner distribution can be calculated with constant time complexity *O(1)*, it alone will be of small use because of the very limited number of elements. Hence, using the corner distribution requires additional means – such as interpolation (cf. Section 9.5) – to determine intermediate time-qos-slots and their properties.

[4] With the dimension of the durations, the distribution would contain eight elements.

9.3.2 Even Distribution

The **even** distribution extends the corner distribution by adding time-qos-slots such that they are evenly distributed over the space TDQ.

Definition 31 (Even Distribution). *Given a moldable reservation request (cf. Table 9.1) and a space of time-qos-slots TDQ, the even distribution TDQ_{dist} satisfies the following conditions:*

- *the number of time-qos-slots is the same at each service level, i.e.,*

$$\begin{aligned}\forall q_1\ \forall q_2 : \quad & q_1 \in TDQ^Q_{dist}\ \wedge\ q_2 \in TDQ^Q_{dist} \\ & \Longrightarrow\ \ |TDQ_{dist}(\cdot,\cdot,q_1)|\ =\ |TDQ_{dist}(\cdot,\cdot,q_2)|\ ,\end{aligned}$$

- *the start times are evenly distributed over the space T, i.e.,*

$$\begin{aligned}\forall q\ \forall t\ \exists k : \quad & q \in TDQ^Q_{dist}\ \wedge\ t \in TDQ_{dist}(\cdot,\cdot,q)\ \wedge\ k \in \mathbb{N} \\ & \Longrightarrow\ \ t\ =\ \min T + \left\lfloor k \frac{\max T - dur(q) - \min T}{\left|TDQ_{dist}(\cdot,\cdot,q)\right| - 1} \right\rfloor\ ,\end{aligned}$$

- *the service levels are evenly distributed over the space Q, i.e.,*

$$\begin{aligned}\forall q\ \exists k : \quad & q \in TDQ^Q_{dist}\ \wedge\ k \in \mathbb{N} \\ & \Longrightarrow\ \ q\ =\ \min Q + \left\lfloor k \frac{\max Q - \min Q}{\left|TDQ^Q_{dist}\right| - 1} \right\rfloor\ .\end{aligned}$$

♢

Note, the corner distribution is a special case of the even distribution.

Example 9.2 (Even Distribution)
A moldable request of a parallel computation is given with the following parameters: $est = 3600$ (earliest start time), $let = 39600$ (latest end time), $dur_{ref} = 1800$ (reference duration), $qos_{low} = 16$ (minimum number of processors), $qos_{high} = 128$ (maximum number of processors), $qos_{ref} = 116$ (reference number of processors) and $dur(qos) =$ `{model=>amdahl:par=>0.99}` (speed-up model). The time-qos-slots of the even distribution are shown in Fig. 9.4.

The even distribution can be calculated with quadratic time complexity $O(n^2)$, with n being the maximum of the number of service levels and the number of slots at a single service level. The even distribution allows to trade-off the accuracy of the properties with the time complexity. That is, the larger is the distribution, the better is the accuracy.

9.3.3 Static Workload Based Distribution

The corner distribution and the even distribution assume that all time-qos-slots are equally important. In real scenarios, however, this assumption may lead to inefficient distributions. Considering the time Φ_{max} at which the current workload of a resource will have been processed, we can deduce the following observations. First,

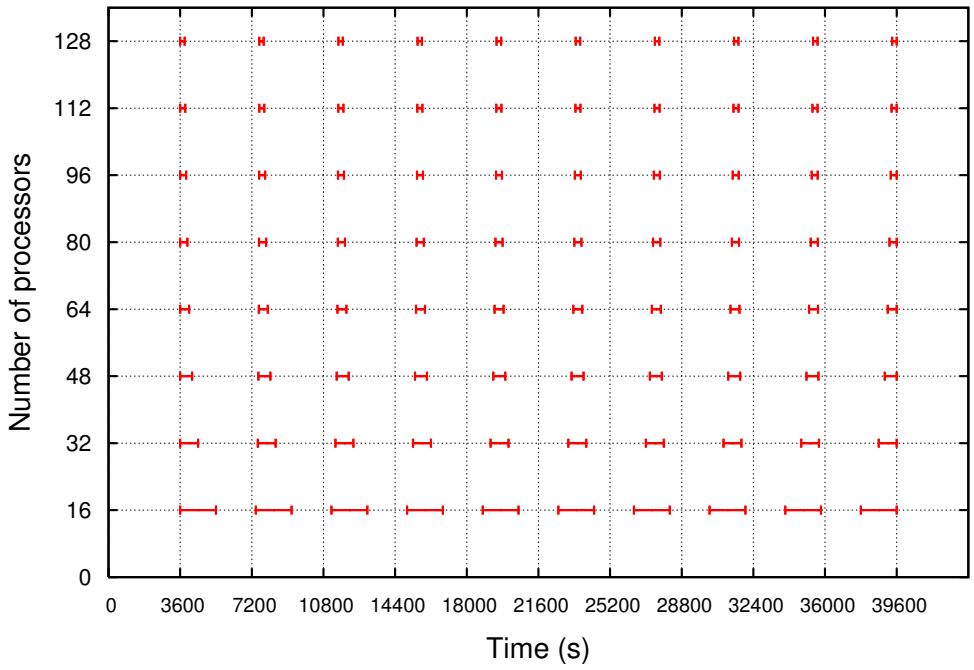

Figure 9.4: Even distribution of time-qos-slots for a moldable request of a parallel computation with the parameters: $dur(qos) =$ `model=>amdahl:par=>0.99`, $qos_{low} = 16$, $qos_{high} = 128$, $qos_{ref} = 16$, $dur_{ref} = 1800$, $est = 3600$, $let = 39600$. The horizontal bars illustrate the potential allocation time (begin, duration, end) of a time-qos-slot.

most changes in the available resource capacity will occur before the time Φ_{max}. Second, there exists a time Φ_{min} ($\Phi_{min} \leq \Phi_{max}$) until which the available resource capacity does not increase significantly. Third, the duration of a reservation does not linearly depend on the service level. For example, most parallel programs contain some sequential fraction $seq > 0$. Hence, smaller service levels (e.g., numbers of processors) have a greater impact on the required duration of a reservation. Thus, the service levels of time-qos-slots should follow a geometric distribution favoring smaller service levels.

Definition 32 (Static Workload Based Distribution). *Given a moldable reservation request (cf. Table 9.1), a space of time-qos-slots TDQ and a workload with the times Φ_{min} and Φ_{max}, the static workload based distribution TDQ_{dist} satisfies the following conditions:*

- *the $M > 1$ service levels TDQ^Q_{dist} form a (nearly) geometric series, i.e.,*

$$\forall q\ \exists m: \quad q \in TDQ^Q_{dist} \ \wedge\ m \in \mathbb{N} \ \wedge\ m \leq M$$
$$\Longrightarrow \quad q = \left\lceil S^{-1}\left(S(\min Q) \cdot \left(\frac{S(\max Q)}{S(\min Q)}\right)^{\frac{m-1}{M-1}}\right)\right\rceil$$

  with S and S^{-1} being the speed-up function and its inverse, respectively,

- *the number of time-qos-slots is the same at each service level, i.e.,*

$$\forall q_1\ \forall q_2: \quad q_1 \in TDQ^{Q}_{dist}\ \wedge\ q_2 \in TDQ^{Q}_{dist} \implies |TDQ_{dist}(\cdot,\cdot,q_1)|\ =\ |TDQ_{dist}(\cdot,\cdot,q_2)|\ ,$$

– *for each service level q, time-qos-slots are placed at the time bounds est (L) and let (U)*

$$\forall q \in TDQ^{Q}_{dist}: \quad stwd_{L,q} := \langle \min T\ ,\ dur(q)\ ,\ q \rangle$$

$$\forall q \in TDQ^{Q}_{dist}: \quad stwd_{U,q} := \langle \max T - dur(q)\ ,\ dur(q)\ ,\ q \rangle$$

– *for each service level q, K time-qos-slots are placed evenly in the interval $[\Phi_{min}, \Phi_{max}]$*

$$\forall q\ \forall t\ \exists k: \quad q \in TDQ^{Q}_{dist}\ \wedge\ t \in TDQ_{dist}(\cdot,\cdot,q)\ \wedge\ k \in \mathbb{N}\ \wedge\ k < K \implies t\ =\ \Phi_{min} + \left\lfloor k \tfrac{\Phi_{max} - dur(q) - \Phi_{min}}{K-1} \right\rfloor$$

The function dur determines the required *duration at a given service level.* ♢

Example 9.3 (Static Workload Based Distribution)
A moldable request of a parallel computation is given with the following parameters: $est = 3600$ (earliest start time), $let = 39600$ (latest end time), $dur_{ref} = 1800$ (reference duration), $qos_{low} = 16$ (minimum number of processors), $qos_{high} = 128$ (maximum number of processors), $qos_{ref} = 116$ (reference number of processors) and $dur(qos) =$ `{model=>amdahl:par=>0.99}` (speed-up model). Furthermore, the workload parameters are $\Phi_{min} = 10800$ and $\Phi_{max} = 28800$. Figures 9.5 and 9.6 illustrate the (nearly) geometric series of service levels and the distribution of time-qos-slots, respectively.

The computational complexity of the static workload based distribution is $O(n^2)$ – the same as of the even distribution if the same number of time-qos-slots are determined plus some constant overhead for calculating Φ_{min} and Φ_{max}. By placing most of the time-qos-slots within the interval $[\Phi_{min}, \Phi_{max}]$ the distribution acknowledges the characteristics of the currently known workload. Moreover it pays attention to the speed-up model of a moldable request by geometrically distributing the service levels within the interval $[qos_{low}, qos_{high}]$.

9.3.4 Other Distributions

The above presented distributions are easy to calculate, but neither consider the variability of the resource's workload nor the "shape" of a property. Here, we briefly describe two distributions which address these issues.

Adaptive Workload Based Distribution. The **adaptive workload based** distribution is an enhanced version of the static workload based distribution. It determines significant events of the currently known workload – end time of active requests, start and end time of existing reservations, start and end time of planned waiting requests – and derives time-qos-slots for them. From these events, only the most significant can be selected. Example selection criteria for compute resources are: (1) the minimum change

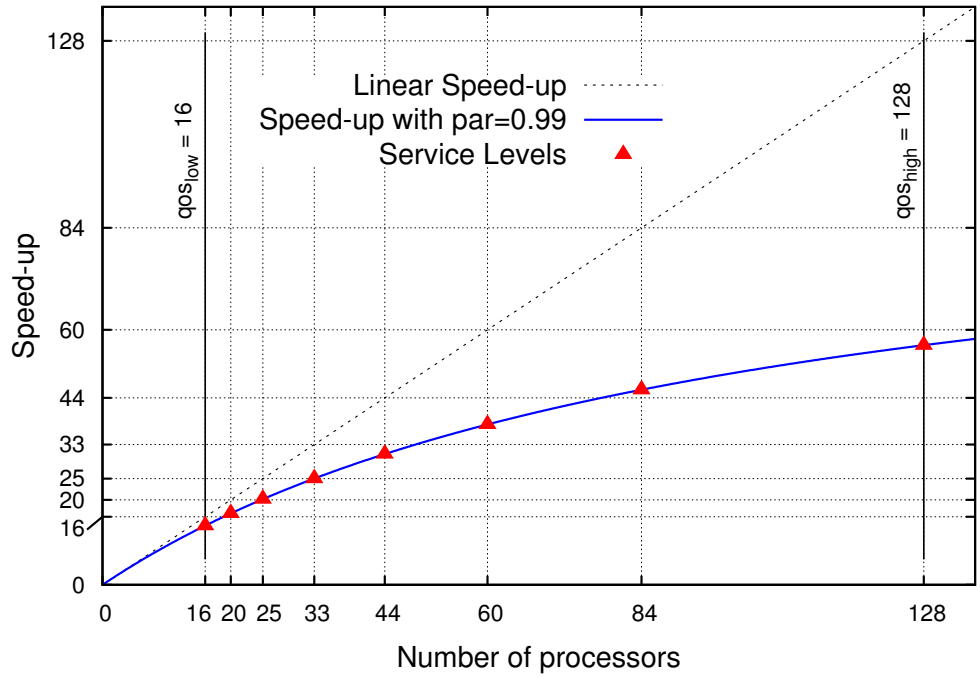

Figure 9.5: Nearly geometric distribution of the service levels for a moldable request of a parallel computation with the parameters: $dur(qos) =$ `model=>amdahl:par=>0.99`, $qos_{low} = 16$, $qos_{high} = 128$ and $M = 8$.

in the processor's allocation must be among the top $10\,\%$ of all changes, (2) the change involves more than $20\,\%$ of the total number of processors, (3) the change is larger than the number of requested processors of the reservation, etc. Then, for each significant event a number of service levels and durations is determined.

Property Based Distribution. The **property based** distribution calculates a minimal number of time-qos-slots such that the characteristics of the property are well represented. For example, if a provider employs a flat rate for the reservation fee, most of the above distributions determine far too many time-qos-slots. Also, cost models with periodic changes (night/day and workday/weekend) are not well covered with the above distribution models. The property based distribution addresses such situations by sampling the property functions and deriving time-qos-slots which allow a good interpolation of intermediate candidates. Obviously, this method is only applicable if the shape of the property's function (e.g., periodic costs) is known by the provider. The computational complexity of this method can be limited by restricting the desired interpolation accuracy and/or sampling frequency.

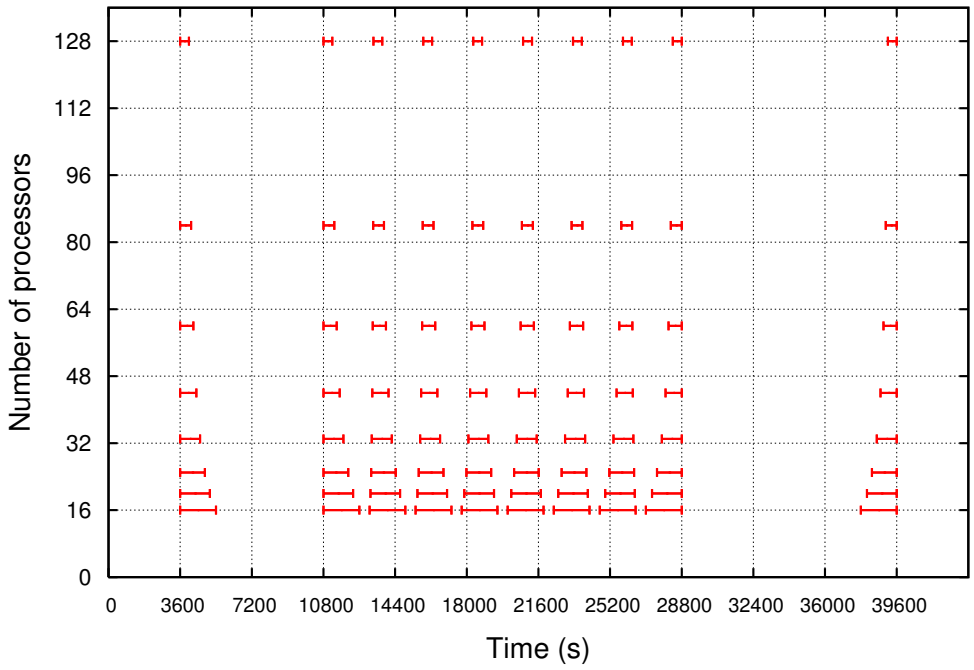

Figure 9.6: Static workload based distribution of time-qos-slots for a moldable request of a parallel computation with the parameters: $dur(qos) =$ `model=>amdahl:par=>0.99`, $qos_{low} = 16$, $qos_{high} = 128$, $qos_{ref} = 16$, $dur_{ref} = 1800$, $est = 3600$, $let = 39600$, $\Phi_{min} = 10800$, $\Phi_{max} = 28800$ and $M = 8$. The horizontal bars illustrate the potential allocation time (begin, duration, end) of a time-qos-slot.

9.4 Properties of Time-QoS-Slots

While other approaches to advance reservation only consider the pure time-qos-slots [WWZ05] or single fixed properties of time-qos-slots [BBES05, ZBN+04], CORES enables richer scenarios by associating sets of properties with a time-qos-slot on-demand. We call an augmented time-qos-slot a *reservation candidate*, i.e., a tuple

$$\langle begin\,,\, duration\,,\, qos\,,\, \{p_i\}\rangle\,.$$

Formally, this section is about determining functions of p_i – the properties – which depend on the elements of a time-qos-slot, the workload and the policies of a resource. In practice, the mechanisms will iterate over the elements of a distribution of time-qos-slots (cf. Section 9.3) and calculate the values of the needed properties.

The factors influencing the value of a property may be classified into three groups: (1) parameters of a time-qos-slot, (2) parameters of the workload and (3) resource management policies.

Parameters of a Time-QoS-Slot. Obviously, the size of a time-qos-slot should have a direct impact. For example, if the number of requested processors is doubled, the

reservation fee should double too. Also, the book-ahead time[5] may have a significant influence. For example, if a reservation requests a time-qos-slot after the currently known workload is finished, the success probability should be very high.

Parameters of the Workload. The policies for handling both the currently known and the presumed future workload may influence the values of a property. For example, a resource provider servicing scientific applications might only accept reservations if they do not delay currently known batch jobs. In contrast, in a commercial scenario, the acceptance of a request may be a function of the balance of income (fee for the request) and loss (penalty for preempting active requests).

Resource Management Policies. A resource owner may enforce its policies by adjusting the properties accordingly. For example, members of the VO, which owns the resource, may receive a higher priority. Additionally, a property's value may reflect pre-determined *service level agreements* (SLA) between a VO and a resource provider. For example, a provider may charge a reduced fee for compute jobs submitted with a certain SLA.

Section Outline. We introduce several properties specific to compute resources and describe the calculation of their values in detail. Albeit we focus on specific properties, it is easy to see that the *probing* mechanism can be adapted to calculate any other property of interest. In Section 9.4.1, we introduce methods for calculating the *reservation success probability* p_{res}. In Section 9.4.2, we present a method for calculating the property *fitness* (short fit). In Section 9.4.3, we demonstrate the generality of our approach by calculating allocation costs of a reservation.

9.4.1 Methods for Deriving the Property p_{res}

Whether or not a time-qos-slot can be successfully reserved depends on many factors like the current utilization, the amount of the requested capacity, the scheduling policies, etc. The reservation success probability $p_{res} \in [0, 1]$ abstracts from all these factors. It expresses the likelihood that a time-qos-slot may successfully be reserved.

It is, however, difficult to determine the exact value of p_{res}, mainly for two reasons. First, the future status of a resource largely depends on the actual run time of the non-reservation jobs, which is difficult to estimate [LSHS04]. Second, the state of the resources may change between calculating the value p_{res} and a subsequent reserve message (cf. Chapter 11). In the following, we describe the methods *static* and *history* for approximating the property p_{res}.

[5] The time span between the current time and the start time of the time-qos-slot.

The Method *static*

The method *static* calculates p_{res} from the book-ahead time bat, i.e., the time span from the current time ct to the start time $begin$ of the time-qos-slot. In reserving network bandwidth, Greenberg et al. [GSW99] observed a certain admission threshold for the book-ahead time. Before this threshold very few requests are granted. After the threshold almost all requests are granted. We define the corresponding function p_{res}^{static} as

$$p_{res}^{static}(bat) = 1 - e^{-\frac{bat}{h}} ,$$

where $h \in \mathbb{N}_+$ is constant configured by the resource provider. Figure 9.7 illustrates the function for different values of h. The admission threshold is set at a p_{res} value of 0.85. The legend shows the values of the constant h and the corresponding book-ahead time threshold (in parentheses).

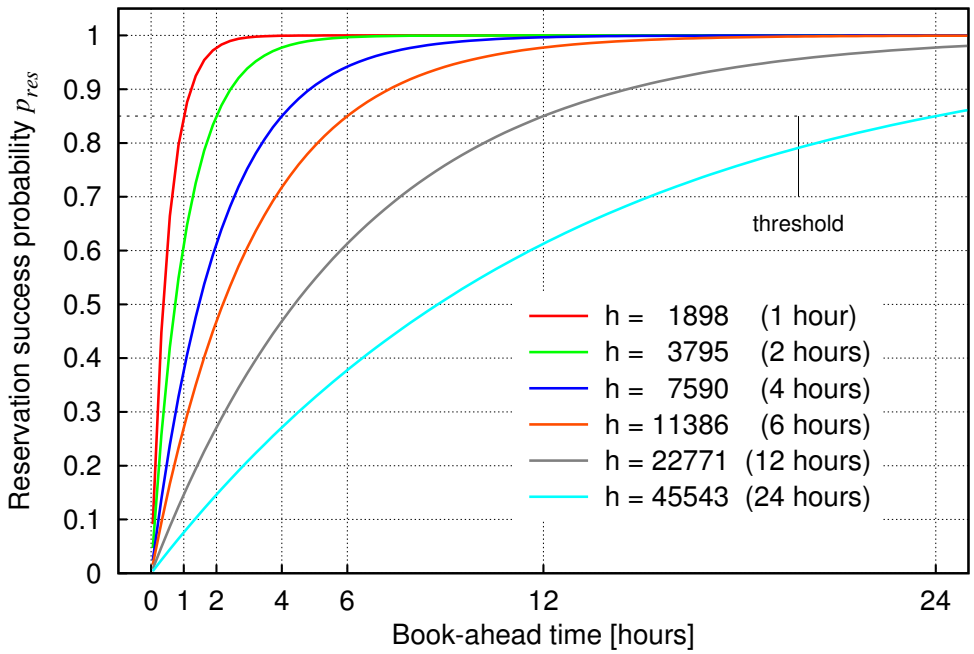

Figure 9.7: Illustration of the reservation success probability p_{res} calculated with the method *static*. The admission threshold is set at 0.85. The legend shows the values of the constant h and the corresponding book-ahead time threshold (in parentheses).

Given a single time-qos-slot the complexity of the method *static* is $O(1)$. Thus, calculating the property of a complete distribution of time-qos-slots TDQ_{dist} has the same complexity as the calculation of the distribution.

The Method *history*

The method *history* is based on recorded utilization data. Every L seconds the number of idle processors is stored in a pair (t, n), where t represents the recording time (e.g,

in seconds since the UNIX epoch) and n is the number of idle processors at that time. The set H contains all recorded data.
Given a time-qos-slot $\langle b, d, np\rangle$ (with $np = qos$ being the number of requested processors), we will calculate the average number of idle processors during the time window $[b, b + d]$ for an averaging period ap, which can be a day, a week, etc. The rationale behind that is to grant requests only if the recorded data indicate that enough capacity should be available at the time of a request.

In the following, we use a *day* as averaging period ($ap = 86400$). Equation (9.1) determines the time of the day $dyt(t)$ of a recording time t.

$$dyt(t) = t - ap \left\lfloor \frac{t}{ap} \right\rfloor . \tag{9.1}$$

Let H_{day} denote the averaged daily history as calculated by

$$H_{day} = \left\{ (t_{day}, n_{day}) \middle| n_{day} = \frac{\sum\limits_{(t,n)\in H \wedge dyt(t)=t_{day}} n}{\sum\limits_{(t,n)\in H \wedge dyt(t)=t_{day}} 1} \right\} . \tag{9.2}$$

Then, Eq. (9.3) determines the average number of idle processors $H^{\varnothing}_{day}(b, d)$ during the time window $[b, b + d]$.

$$H^{\varnothing}_{day}(b, d) = \frac{\sum\limits_{\substack{(t,n)\in H_{day} \wedge \\ [t,n)\cap[dyt(b),dyt(b+d))\neq\emptyset}} n}{\sum\limits_{\substack{(t,n)\in H_{day} \wedge \\ [t,n)\cap[dyt(b),dyt(b+d))\neq\emptyset}} 1} \tag{9.3}$$

Finally, the property function $p^{history}_{res}$ is defined in Eq. (9.4).

$$p^{history}_{res}(b, d, np) = \begin{cases} 1 & \text{if } 2np \leq H^{\varnothing}_{day}(b, d) \\ 2 - \frac{2np}{H^{\varnothing}_{day}(b,d)} & \text{if } np \leq H^{\varnothing}_{day}(b, d) \\ 0 & \text{if } np > H^{\varnothing}_{day}(b, d) \end{cases} \tag{9.4}$$

Because the number of idle processors is only an average value, the p_{res} value increases from 0.0 to 1.0 over an interval from np to $2np$ idle processors. Figure 9.8 illustrates the function $p^{history}_{res}$ for different numbers of requested processors (symbols) and idle processors (horizontal axis).

The complexity of the method *history* is dominated by calculating the average number of idle processors for a set H as defined by Eq. (9.2). Thus, the complexity is $O(n)$ with $n = 2|H|$. If the averaging period ap is known a-priori, the complexity may be reduced by updating the average history – e.g., H_{day} – at each recording event. In that case, the complexity is $O(n)$ still, but with $n = 2|H_{day}|$ or, more precisely, $n = 2\lceil ap/L \rceil$.

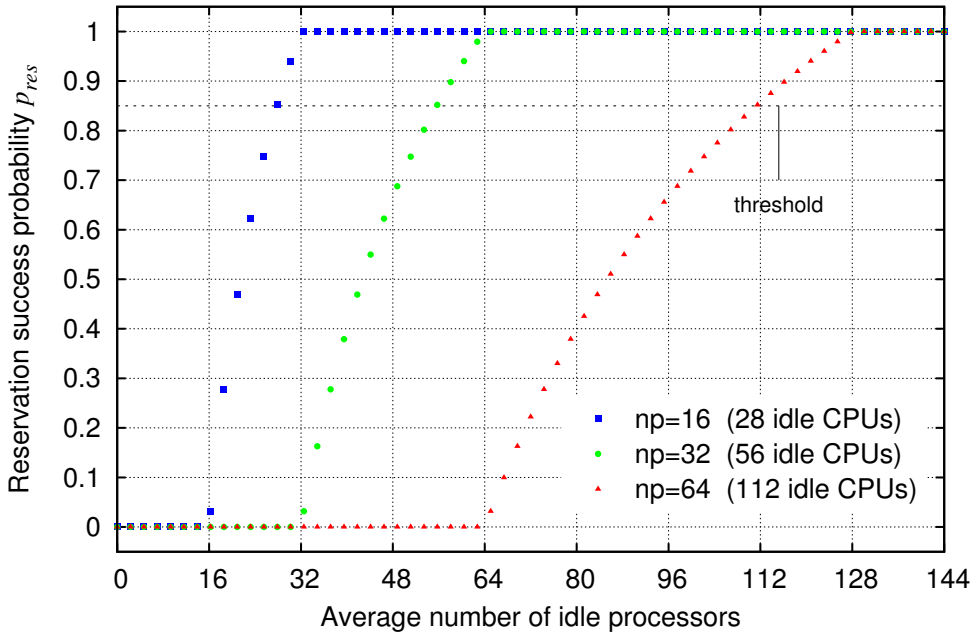

Figure 9.8: Illustration of the reservation success probability p_{res} calculated with the method *history*. The admission threshold is set at a p_{res} value of 0.85. The legend shows the values of the requested number of processors (np) and the corresponding threshold wrt. the number of idle CPUs (in parentheses).

9.4.2 Methods for Deriving the Property fit

The property $fit \in [0, 1]$ provides information about, how well the request fits into the workload of a resource. Although it may be interpreted as *reservation success probability*, we do not strictly define nor interpret it as a probability. Instead, we interpret it as a measure by which a resource specifies its preferences among the time-qos-slots.

The calculation of the property fit takes information about the currently known workload, the request parameters and the resource's local scheduling policies into account. Due to the dynamic nature of a workload, the actual fitness of a time-qos-slot may quickly change. The methods we propose do not cope with this issue directly. Instead, we require that the subsequent reservation steps need to process the information as soon as possible (cf. Chapter 10).

An important issue in deriving the fitness of time-qos-slots is the inaccurate estimates of the runtime of non-reservation jobs [LSHS04]. As a consequence, each job finishing earlier than estimated may partially or completely invalidate the determined fitness values.

In the following, we describe the methods *load* and *what-if* for calculating the property fit. Table 9.2 lists the main parameters of the currently known workload (at time ct) of a compute resource with np processors.

Table 9.2: Parameters of the workload known at the current time ct.

Name	Description
	For each running batch job $rj^k, k \in [1, K]$
rj^k_{stt}	start time
rj^k_{wct}	wall clock time, i.e., the limit on the run time
rj^k_{np}	number of processors
	For each waiting batch job $wj^l, l \in [1, L]$
wj^l_{wct}	wall clock time, i.e., the limit on the duration
wj^l_{np}	number of processors
	For each granted reservation $rsv^m, m \in [1, M]$
rsv^m_{stt}	start time
rsv^m_{edt}	end time
rsv^m_{np}	number of processors

The Method *load*

The method *load* uses information on the current state of the system: running and waiting jobs and active or pending (but already granted) reservations. This information is used to calculate an approximate time T_{WKL} at which the processing of the current workload will be finished. Given a time-qos-slot $\langle b, d, np \rangle$, Eq. (9.5) defines the property fit^{load}.

$$fit^{load}(b, T_{WKL}) = \begin{cases} 1 & \text{if } b \geq T_{WKL} \\ 0 & \text{if } b < T_{WKL} \end{cases} \tag{9.5}$$

The time T_{WKL} is calculated as follows. Let rj^k_{ret} ($k \in [1, K]$) denote the remaining execution time of the k-th running job as defined by

$$rj^k_{ret} = rj^k_{wct} - (ct - rj^k_{stt}) \,.$$

The parameter δ_{rj} specifies the average accuracy of the remaining execution time of the running jobs. The parameter δ_{wj} specifies the average accuracy of the estimated execution time of the waiting jobs.

The approximative time T_{WKL} is calculated in three steps. First, we determine the average end time T_{rj} of the running jobs as follows

$$T_{rj} = \left\lfloor \frac{1}{np} \sum_{k=1}^{K} \left(rj^k_{np} \; rj^k_{ret} \; \delta_{rj} \right) \right\rfloor \,.$$

Second, we calculate the average processing time T_{wj} for the waiting jobs using their estimated execution time as follows

$$T_{wj} = \left\lfloor \frac{1}{np} \sum_{l=1}^{L} \left(wj^l_{np} \; wj^l_{wct} \; \delta_{wj} \right) \right\rfloor .$$

The intermediate T_{WKL} is the sum of T_{rj} and T_{wj}. Third, we include existing reservations (scheduled between the current time ct and T_{WKL}) by iteratively increasing T_{WKL} for such reservations (cf. Alg. 1). In Alg. 1, the set R contains all remaining reservations whose time window $[rsv^m_{stt}, rsv^m_{edt})$ interferes with the time span $[ct, T_{WKL})$. Clearly, the algorithm terminates in at most $|\{rsv^m\}|$ rounds of the **while**–loop.

Algorithm 1: Iterative procedure for calculating the time T_{WKL}.

Require: $T_{WKL} \geq ct, ct \in \mathbb{N}_+, T_{WKL} \in \mathbb{N}_+, RSV = \{rsv^m\}$

$R \Leftarrow \{rsv^m \,|rsv^m \in RSV \;\wedge\; [rsv^m_{stt}, rsv^m_{edt}) \cap [ct, T_{WKL}) \neq \emptyset\}$

while $R \neq \emptyset$ **do**

$$\begin{aligned} T_{WKL} &\Leftarrow T_{WKL} + \left\lfloor \tfrac{1}{np} \sum_{rsv^m \in R} \left(rsv^m_{np} \, (rsv^m_{edt} - rsv^m_{stt}) \right) \right\rfloor \\ RSV &\Leftarrow RSV \setminus R \\ R &\Leftarrow \{rsv^m \,|rsv^m \in RSV \;\wedge\; [rsv^m_{stt}, rsv^m_{edt}) \cap [ct, T_{WKL}) \neq \emptyset\} \end{aligned}$$

Although the value of T_{WKL} is only a rough approximation, the method proved to be reasonable in our experiments (cf. Section 9.6). Due to the definition of the property fitness (cf. Eq. (9.5)), however, holes in the schedule between the current time ct and T_{WKL} are not considered for reservations. Existing reservations starting later than T_{WKL} are also not taken into account. The latter may lead to failing reservation attempts if requests conflict with the existing advance reservations.

The complexity of calculating T_{rj} and T_{wj} is $O(n)$ ($n = K$ running jobs) and $O(n)$ ($n = L$ waiting jobs), respectively. In the worst case, the **while**-loop in Alg. 1 is executed M (granted reservations) times if in each round a single reservation is used to increase T_{WKL}. Hence, determining the set R requires $M(M-1)/2$ operations (i.e., $[rsv^m_{stt}, rsv^m_{edt}) \cap [ct, T_{WKL}) \neq \emptyset$). Thus, the total complexity of calculating T_{WKL} is $O(n^2)$ ($n = M$).

The Method *what-if*

The basic idea of the method *what-if* is to let the fitness reflect the impact of a reservation on non-reservation jobs. For this purpose, the local scheduling system must be able to construct execution plans for jobs without executing them. This requirement is, however, not very restricting, as commonly-used cluster-level schedulers, such as Maui [JSC01], CCS [KR01] or OAR [CCG+05], either provide a simulation mode or can operate in planning mode.

The method *what-if* uses three kinds of execution plans: (1) *original (ORG)*, (2) with *reservation placeholder (RSV)* and (3) with *job placeholder (JOB)*. In the following we describe the generation of the execution plans in detail.

Original. The execution plan P^{ORG} is the schedule of the current workload without the reservation request. We use P^{ORG} as reference plan.

Reservation Placeholder. Given a time-qos-slot $\langle b, d, np \rangle$, this plan places a temporary reservation for np processors from the time b until the time $b + d$ into the original schedule. Then, it determines the execution plan $P^{RSV/b}$ by scheduling the waiting jobs. This procedure is repeated for all time-qos-slots of a distribution.

Job Placeholder. The scheduler determines the time when a job with the same duration and processor requirements of the request would have been started. Therefore a temporary job with an estimated execution time j_{wct} equal to the duration of the reservation request d and the same number of requested processors is submitted to the scheduler. The resulting execution plan P^{JOB} defines a new time-qos-slot by setting its properties as follows: $b = j_{stt}$ (the job's start time), $d = d$ (duration) and $np = np$ (number of processors).

The algorithm calculates the property $fit^{what-if}$ for the execution plans $P^{\{JOB,RSV/b\}}$ by using well-known scheduling metrics such as makespan C_{max} and average completion time C_{avg} of jobs. These metrics are normalized to simplify their weighting in the subsequent calculation. Other well-known scheduling metrics such as slowdown or resource utilization can be easily added if needed. If a time-qos-slot cannot be reserved by a reservation placeholder – because it conflicts with running jobs, the first job at the head of the waiting queue (EASY backfilling) or existing reservations – its fitness value is set to zero.

Let $N = K + L$ be the number of jobs of the current workload and P be one of the above execution plans. The start time of job j^i in the execution plan P is denoted by the term $stt(P, j^i)$ where $1 \leq i \leq N$. The submission time and estimated execution time of a job j^i are denoted by j^i_{sbt} and j^i_{wct}, respectively.

We assess the quality of a simulated execution plan by computing the makespan and the average completion time. Equation (9.6) defines the makespan $C_{max}(P)$ of an execution plan P.

$$C_{max}(P) = \max_{1 \leq i \leq N} \left(stt(P, j^i) + j^i_{wct} \right) \tag{9.6}$$

The makespan C_{max}, stating how long the resource will be occupied, ranks the execution plans from the point of view of the resource owner. Let C^*_{max} denote the minimum makespan for all considered execution plans $P^{\{JOB,RSV/b\}}$.

Equation (9.7) defines the average completion time $C_{avg}(P)$ of the jobs in an execution plan P.

$$C_{avg}(P) = \frac{1}{N} \sum_{1 \leq i \leq N} \left(stt(P, j^i) + j^i_{wct} - j^i_{sbt} \right) \tag{9.7}$$

The average completion time C_{avg} expresses how fast on average the jobs are completed. Thus, it rates the execution plans from the point of view of the users. Let C^*_{avg} denote the minimum average completion time for all considered execution plans $P^{\{JOB,RSV/b\}}$.

Let $\omega_{C_{max}}$ denote the weight for the makespan ($\omega_{C_{max}} \geq 0, \omega_{C_{max}} \in \mathbb{R}$) and $\omega_{C_{avg}}$ denote the weight for the average completion time ($\omega_{C_{avg}} \geq 0, \omega_{C_{avg}} \in \mathbb{R}$). We require that $\omega_{C_{max}} + \omega_{C_{avg}} = 1$.

The property $fit^{what-if}$ of a time-qos-slot $\langle b, d, np \rangle$ is defined by Equation 9.8 using the execution plan "reservation placeholder" $P^{RSV/b}$ generated for that time-qos-slot.

$$fit^{what-if}(b, d, np) = \omega_{C_{max}} \frac{C^*_{max}}{C_{max}(P^{RSV/b})} + \omega_{C_{avg}} \frac{C^*_{avg}}{C_{avg}(P^{RSV/b})} \tag{9.8}$$

By using the minimum values C^*_{max} and C^*_{avg} as numerator of the fractions, components of the makespan and the average completion time are normalized. Thus, the property's value is a number in the real interval $[0, 1]$. The more a time-qos-slot delays the execution of the local jobs, the lower is the value of $fit^{what-if}$.

The complexity of the method *what-if* depends on the costs of placing a single job into the schedule of k running jobs and m granted reservations. In the worst case, it requires $k+m$ scheduling events to be considered. Thus, the complexity of determining a schedule for n waiting jobs into a schedule is $O(n^2)$ with $n \approx k + m$.

9.4.3 Methods for Deriving Reservation Costs

Each resource provider defines the cost c_{BU} of a base unit (unit of time times unit of service level). A base unit has a duration dur_{BU} and is applicable for a single processor $np_{BU} = 1$. Assuming that the allocation costs do not change with the time of the day and the user's affiliation, Eq. (9.9) defines the basic allocation cost of a time-qos-slot $\langle b, d, np \rangle$.

$$cost^{basic}(b, d, np) = \frac{d \cdot np}{dur_{BU} \cdot np_{BU}} c_{BU} \tag{9.9}$$

Assuming that the allocation costs change with the time of the day and the day of the week, Eq. (9.10) defines a more realistic cost function $cost^{time}$. Instead of using a fixed cost of the base unit (as above), the cost of a base unit is defined as a function itself.

$$cost^{time}(b, d, np) = \frac{np}{dur_{BU} \cdot np_{BU}} \sum_{t=b}^{b+d-1} c_{BU}(t)\,, \tag{9.10}$$

where $c_{BU}(t)$ is the time-dependent price of a base unit. The complexity for calculating $cost^{time}$ is $O(n)$, with $n = d$.

9.5 Intermediate Time-QoS-Slots

The number of time-qos-slots calculated by the methods discussed in Section 9.3 is usually much smaller than the number of feasible time-qos-slots, i.e., the size of the space TDQ. Considering single resource requests, the small number of determined slots limits the options for placing a reservation, only. In particular, the placement may not be the optimal one. Considering co-reservation requests, a small number of slots may be insufficient for deriving feasible solutions. Therefore, we need a mechanism for determining intermediate time-qos-slots and calculating their properties. We call a time-qos-slot $\langle b^{imc}, d^{imc}, q^{imc} \rangle$ **intermediate** if the following condition holds.

$$\begin{aligned} \forall b\, \forall q: \quad & \langle b, dur(q), q \rangle \in TDQ_{dist} \\ \Longrightarrow \quad & b \neq b^{imc} \ \vee \ dur(q) \neq d^{imc} \ \vee \ q \neq q_{imc} \end{aligned}$$

For example, all time-qos-slots except the four corners ($cd_{L,L}$, $cd_{L,U}$, $cd_{U,L}$ and $cd_{U,U}$) are intermediate time-qos-slots of the *corner* distribution (cf. Def. 30).

While it is straightforward to determine intermediate time-qos-slots, calculating the values of their properties needs to take various aspects into account. The standard technique is to interpolate these values. Interpolation techniques have been developed for many disciplines such as visualization and engineering. Therefore, we discuss the important aspects of applying interpolation mechanisms to the problem of processing co-reservation requests, only.

Considering a distribution of time-qos-slots as shown in Fig. 9.9 and the values of a single property as shown in Fig. 9.10, we discuss the following seven aspects for using interpolation techniques

A1 – *the dimensionality and type of the definition space*,

A2 – *the used base functions*,

A3 – *the fitting of the interpolant at the given time-qos-slots (data points)*,

A4 – *the partitioning of the space* TDQ,

A5 – *the characteristics of the boundaries of adjacent partitions*,

A6 – *the efficiency of deriving the interpolant*, and

A7 – *the limitations imposed by the "consumers" of the interpolants*.

A1 – Dimensionality of the Definition Space

A property's value may depend on different numbers of variables. In the example of Fig. 9.10, the property depends on two variables – the start time and the service level (number of processors). The interpolation technique may be tailored at specific characteristics of the application scenario (using domain specific information) or be commonly applied to any dimensionality of the definition space.

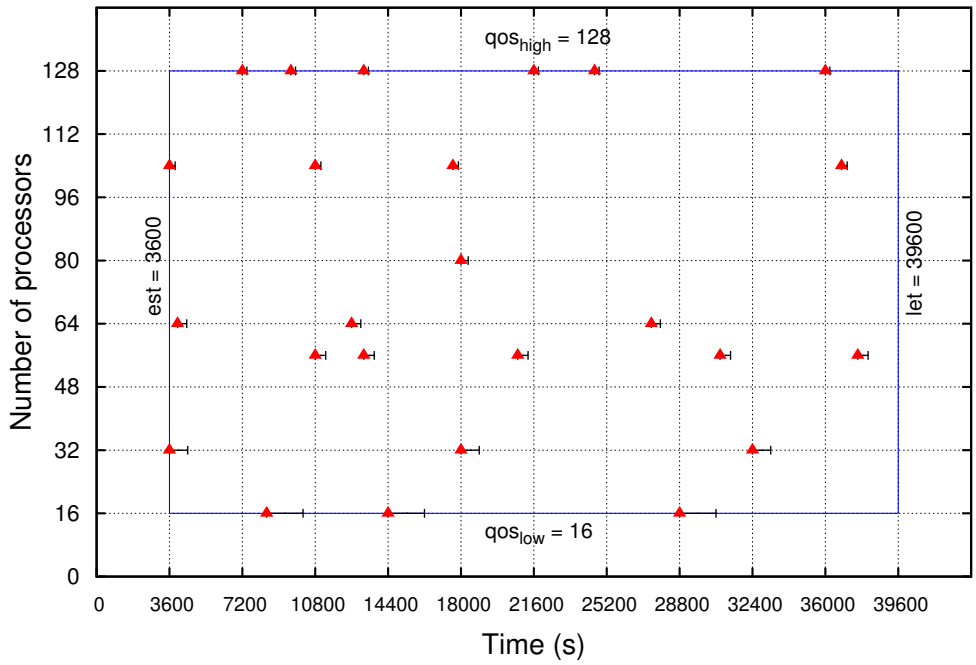

Figure 9.9: Time-qos-slots for a moldable request with the parameters: $dur(qos) =$ `model=>amdahl:par=>0.99`, $qos_{low} = 16$, $qos_{high} = 128$, $qos_{ref} = 16$, $dur_{ref} = 1800$, $est = 3600$, $let = 39600$. The red triangles mark the start time of a time-qos-slot. The horizontal bars stretching to the right from the triangles illustrate the duration of a time-qos-slot. The blue rectangle illustrates the request's bounds.

A2 – Base Functions

Depending on the application area different base functions are commonly used. For example, in visualization mechanisms radix functions are often used as base functions. In scheduling disciplines, however, step functions may be more appropriate, simply because processing capacity is allocated and freed in chunks of integer numbers. Other base functions are polynomial and linear (as special case of polynomial). The choice of the base function depends on many aspects, particularly on the limitations imposed by the use of the interpolants (cf. aspect **A7**).

A3 – Fitting of the Interpolant

In general, the interpolant should exactly fit the property's value at the original time-qos-slots. Obviously, this may not always be possible, e.g., no linear function may interpolate the three data points $(0, 0)$, $(3, 3)$ and $(6, 5)$ in the one-dimensional case. Thus, given the number of time-qos-slots and their property's values, the interpolant may exactly fit the values at the given time-qos-slots or approximates them only. If an interpolant may fit the property's values of the given time-qos-slots exactly, depends on the chosen base function (cf. aspect **A2**) and the partitioning of the space TDQ (cf. aspects **A4** and **A5**).

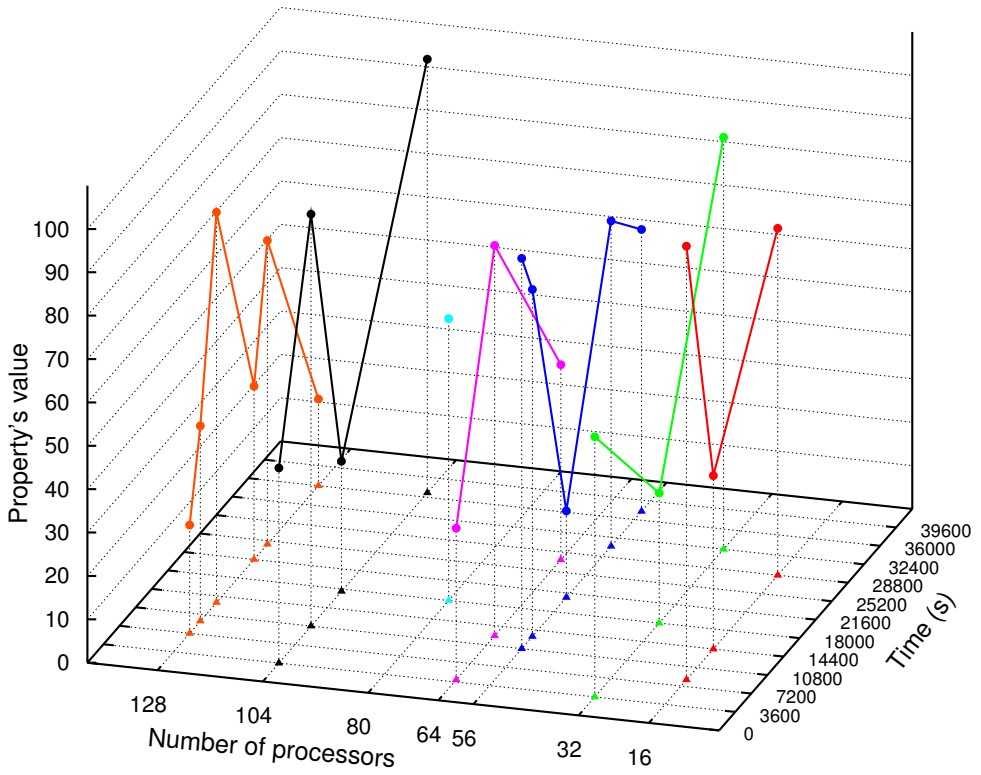

Figure 9.10: Example values of a property for the distribution of time-qos-slots shown in Fig. 9.9. The triangles correspond to the start time of the time-qos-slots in Fig. 9.9.

A4 – Partitioning of the Space TDQ

Using certain base functions, it may be impossible to determine an interpolant which exactly fits the property's values at the original time-qos-slots. If the approximation is not satisfactory, the space TDQ may be split into several partitions and appropriate interpolants may be calculated for each partition. The partitioning may implement some regular pattern or be arbitrarily chosen depending on the distribution of the original time-qos-slots and on the later use in subsequent processing steps.

A5 – Characteristics of the Interpolant

The interpolants of adjacent partitions may be calculated such that their boundaries meet certain conditions. For example, all adjacent facets have the same property's value at the boundary or all adjacent facets possess the same gradient at the boundary.

A6 – Efficiency

Because the status of Grid resources may rapidly change, the calculation of the interpolant must be very efficient.

A7 – Limitations

The interpolant will be used in the subsequent processing step searching for an optimal co-reservation candidate. Thus, the mechanism or the tool used in the optimization step may impact decisions on the aspects discussed so far. For example, many standard solvers only support linear or quadratic functions in constraints and objectives. The consequence of using linear or quadratic functions can be that the candidate space must be partitioned to obtain an acceptable fitting of the interpolant.

9.6 Experimental Evaluation

The stakeholders involved in processing a reservation request may have different and conflicting objectives on the performance for deriving reservation candidates.

Objectives of Different Stakeholders. The *requesters* desire a well fulfillment of their objectives, e.g., to allocate the earliest possible time-qos-slot or the cheapest one. The *providers* need to ensure a fair treatment of the non-reservation requests and desire a high utilization of their resources. The *reservation system* is – for two reasons – mainly interested in an efficient processing of the requests. First, the status of the resources may quickly change, and invalidate the properties' values of the time-qos-slots. Second, requesting a reservation at a resource may induce a significant overhead for the resource to determine if it can admit the request. Thus, the reservation system shall only request reservations which are likely to succeed.

Evaluation Focus. Albeit most of the methods developed in the previous sections are generic and applicable to different types of resources, we exclusively used compute resources in our evaluation. Due to the large number of parameters, the evaluation covers a subset of the described mechanisms only. Particularly, the experiments use the **even** distribution (cf. Section 9.3.2) and test the methods for deriving the properties p_{res} (cf. Section 9.4.1) and fit (cf. Section 9.4.2).

Evaluation Summary. The main results are:

- using more detailed information leads to better system's performance,
- larger request's flexibility results in higher *total reservation success rate*, *and*
- a specific workload pattern causes the majority of reservation-induced job delays.

Evaluation Methodology. Job scheduling algorithms are typically evaluated with off-line simulations based on workload logs of real systems [FF05], i.e., supercomputers or clusters. We evaluated the methods for determining reservation candidates using the workload log of the *SDSC Blue Horizon* supercomputer published in [Fei07].

The system has 144 nodes, each of them consisting of an 8-way SMP with a crossbar connected to a shared memory. While this gives a total of 1152 CPUs, the scheduler always allocates full nodes to requests. Because the workload log contains non-reservation batch jobs only, we converted a fraction of these into reservation requests.

Section Outline. In the following, we briefly introduce the experimental setup in Section 9.6.1, describe the main properties of the used job and reservation workloads in Section 9.6.2, introduce the measured performance metrics in Section 9.6.4, describe the parameters of the simulation runs in Section 9.6.3, and present the results of the simulations in Section 9.6.5.

9.6.1 Experimental Setup

The experiments were carried out in a simplified setup with a single, simulated resource shown in Fig. 9.11. The resource was managed by a local scheduler (LRMS) in simulation mode. On top of the LRMS we used the Local Reservation Service (LRS), which processes the probe messages and forwards reserve messages to the LRMS. The Grid Reservation Service (GRS) receives the reservation requests.

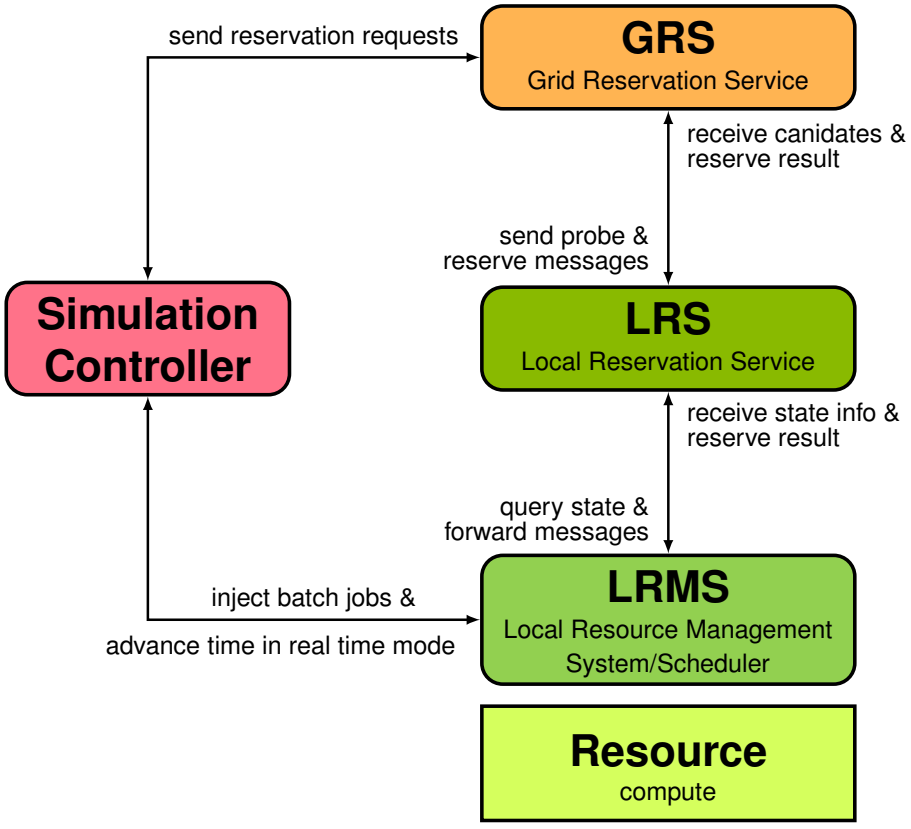

Figure 9.11: Experimental setup of the reservation framework and the interplay of the components during a simulation run.

The discrete event simulation is steered by a single controller which injects jobs into the LRMS and sends reservation requests to the GRS. It also switches between logical and real time mode. The former is applied if no reservation request is being processed. Thus, the duration of a simulation may be much smaller than the time span of the workload logs. The real time mode is used if a reservation request is being processed. In that mode, the LRMS receives every n seconds a signal to advance its state by n seconds. Thus, we can model state changes of the resource over the course of processing a request.

Only a single site (resource) was used in the experiments. In a multi-site scenario, we expect a higher acceptance rate for reservation requests and load balancing among the sites. Besides simulating only a single resource and leaving out the resource discovery (cf. Fig. 9.11), we employed two additional restrictions to simplify the experiments. First, we allowed only one client to request a reservation at any time instance, i.e., we did not simulate concurrent requests. Second, we used the same user objectives for all reservation requests. The prioritized user preferences were set to (1) earliest end time, (2) minimum cost and (3) maximum reservation success probability p_{res}.

9.6.2 Workloads

We describe the creation of the workloads and their quantitative characteristics.

Creation of the Workloads

The original workload log of the *SDSC Blue Horizon* supercomputer contains 250,440 jobs. We cleaned the log by removing interactive jobs and jobs, which allocated zero processors or ran for zero seconds. The cleaned log contains 161,016 jobs.

To reduce the run time of a single simulation, we selected the first 2000 batch jobs from the cleaned log, resulting in a simulated wall clock time of approx. ten days. Since the workload log does not include information on job reservations, we transformed 10% of the 2000 jobs into reservation requests. This was done by splitting all jobs into 200 groups of 10 consecutive jobs and randomly selecting one job from each group to become a reservation request. The selection was done once. All simulations were run with the same sets of remaining batch jobs and converted reservation requests.

The parameters of a converted reservation request rsv^i are derived as follows. The submission time rsv^i_{sbt}, the reference duration $rsv^i_{dur_{ref}}$ and the reference number of processors $rsv^i_{np_{ref}}$ are set to the job's submission time job^j_{sbt}, the job's allocated number of processors job^j_{np} and the job's actual execution time job^j_{aet}, respectively. All requests used Amdahl's law (cf. Eq. (4.1)) as speed-up model with sequential fractions s in the interval $[0, 0.1]$. The properties to be determined were the reservation cost ($cost$), the reservation success probability (p_{res}) and the fitness of a time-qos-slot (fit). We chose varying values for the remaining parameters – the earliest start time (est), the latest end time (let), and the range of the number of processors ($[qos_{low}, qos_{high}]$) – for different simulation runs. The calculation of these parameters is as follows.

Execution Time Window $[est, let]$ The earliest start time est is the sum of the job's submission time job^j_{sbt} and the book-ahead time $bat \in \{0, 2, 4, 6, 12, 24\}$ (in hours). The book-ahead time bat is the same for all requests in a single simulation run. The latest end time let is the sum of the earliest start time est, the reference duration of the reservation request $rsv^i_{dur_{ref}}$ and the start time flexibility $stf \in \{0, 1, 2, 5, 10, 30\}$ (in hours). Note, while the size of the execution time window may differ for different reservation requests in a single simulation run, the start time flexibility is the same for all requests in a single simulation run.

Service Level Flexibility $[qos_{low}, qos_{high}]$ The range $[qos_{low}, qos_{high}]$ defines the flexibility in the number of processors that can be utilized by the program. The bounds qos_{low} and qos_{high} are determined by multiplying the job's number of allocated processors job^j_{np} by the factors f_{low} and f_{high}. In the simulation runs, we used two sets of factors: $\langle 1, 1 \rangle$ for simulating a program without any flexibility and $\langle 0.5, 2 \rangle$ for simulating a more flexible program. For example, when the original job in the workload log ran on 8 processors, we used the ranges $[8, 8]$ and $[4, 16]$ in two different simulation runs.

Characteristics of the Workload of the 1,800 Remaining Jobs

Figure 9.12 shows the distribution of the jobs' actual execution time. The crosses illustrate the number of jobs with a specific execution time (left vertical axis). The violet curve shows the cumulated number of jobs (right vertical axis).

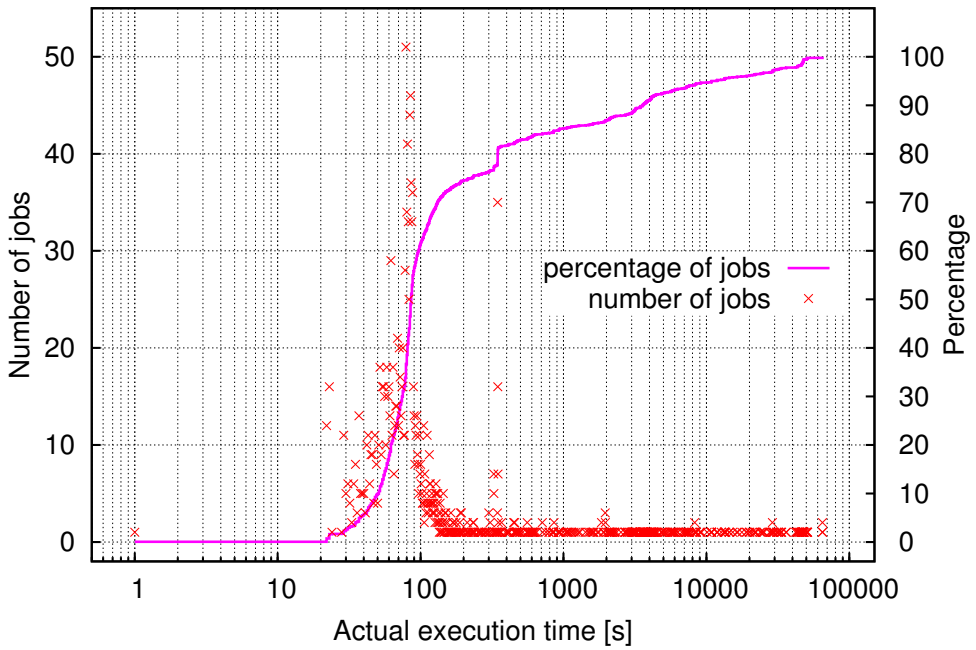

Figure 9.12: Distribution of the actual execution times.

Figure 9.13 shows the distribution of the numbers of processors. Crosses show the number of jobs (left axis). The violet curve shows the percentage of jobs (right axis).

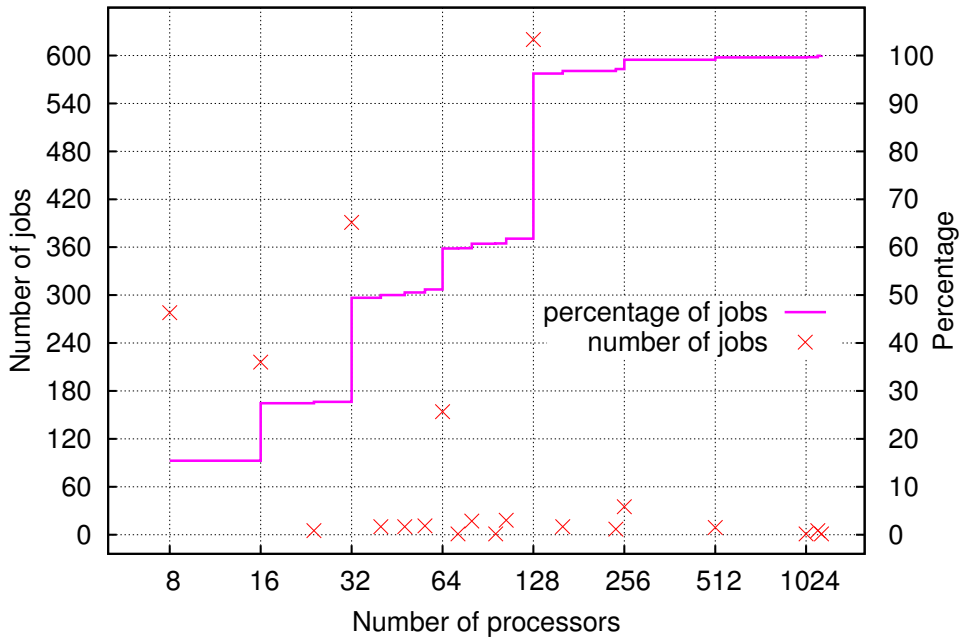

Figure 9.13: Distribution of the number of processors.

Figure 9.14 shows the varying utilization of the processors (vertical axis) along the simulation time (horizontal axis).

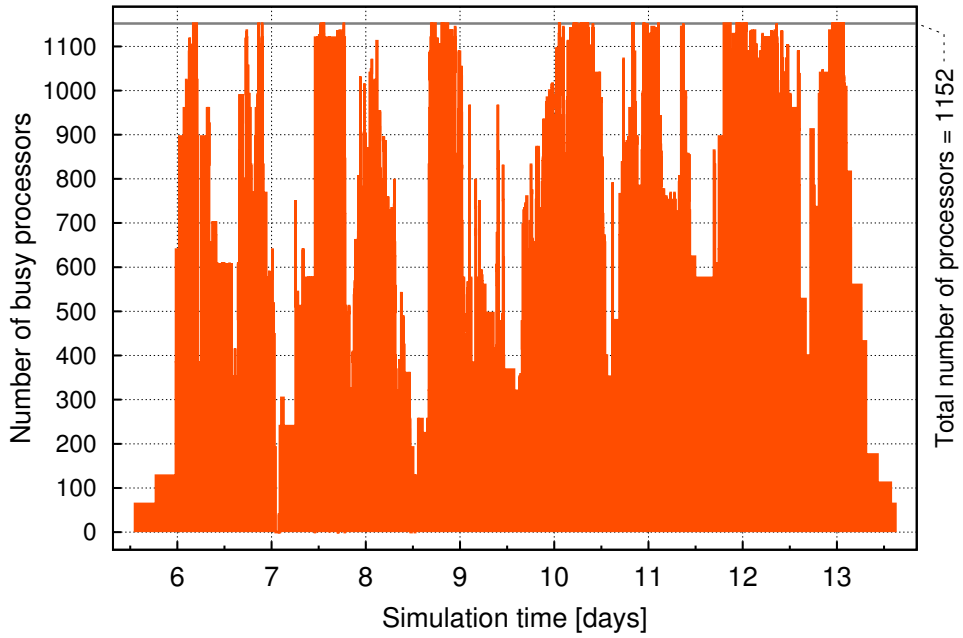

Figure 9.14: Utilization for the 1,800 remaining batch jobs.

Figure 9.15 shows the backlog (vertical axis) vs. the simulation time.

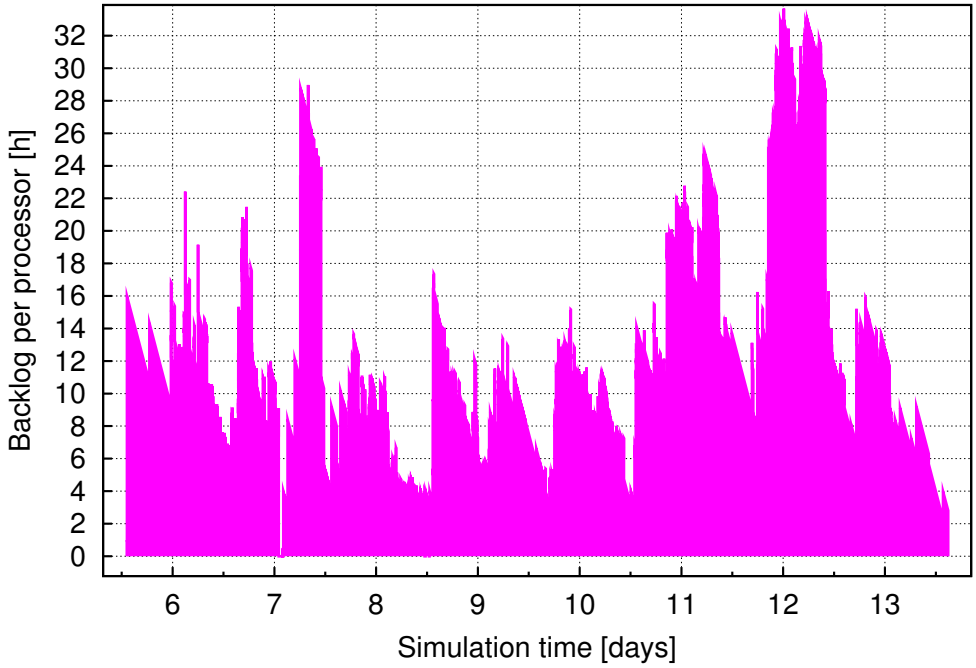

Figure 9.15: Backlog for the 1,800 remaining batch jobs.

Characteristics of the Workload of the 200 Reservation Requests

Because the reservation requests were derived from jobs of the workload log, the distribution of their duration and number of processors shows the same characteristics as the remaining jobs (cf. Figures 9.12 and 9.13).

Figure 9.16 illustrates the relation between the utilization incurred by the 1,800 remaining jobs and the number of processors of the reservation requests. The simulation time is shown at the horizontal axis. The yellow graph shows the utilization (as in Fig. 9.14). The red error bars stretching down from the total number of processors show the reservation requests. Each bar is printed at its earliest start time. The length of each bar represents the requested number of processors. The illustration shows phases with a large potential for conflicts between the jobs and the reservation requests (at peak utilization), but also phases with small potentials for conflicts (at lower utilization).

Figure 9.17 illustrates the relation between the utilization incurred by the 1,800 remaining jobs and the duration of the reservation requests. The utilization part is the same as in the last figure. The red error bars are printed at the vertical position $1152 - rsv_{np}$. Horizontally, each bar begins at its earliest start time (the start time flexibility was set to zero hours) and extends to the right till its latest end time. That is, the length of each bar illustrates the request's duration. Clearly, there are many small requests not interfering with the utilization generated by the non-reservation jobs. Even if they would be requested with different book-ahead times, they would most likely

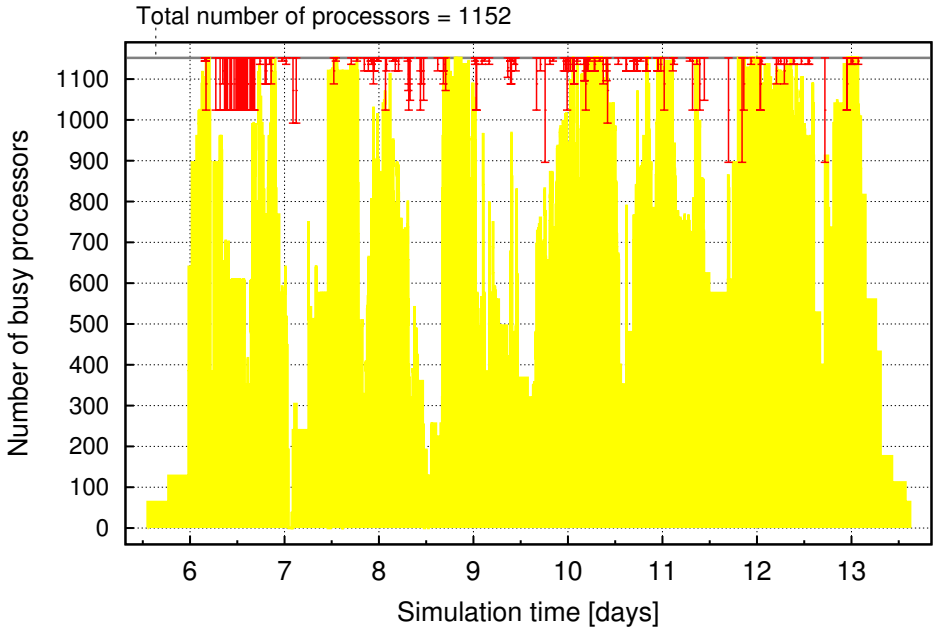

Figure 9.16: Example distribution of the requested number of processors (vertical error bars) drawn at the reservation's earliest start time vs. the background utilization.

not interfere with the background utilization. On the other hand, the larger reservation requests interfere with the background utilization for the shown book-ahead times and would do so at other (larger) book-ahead times, too.

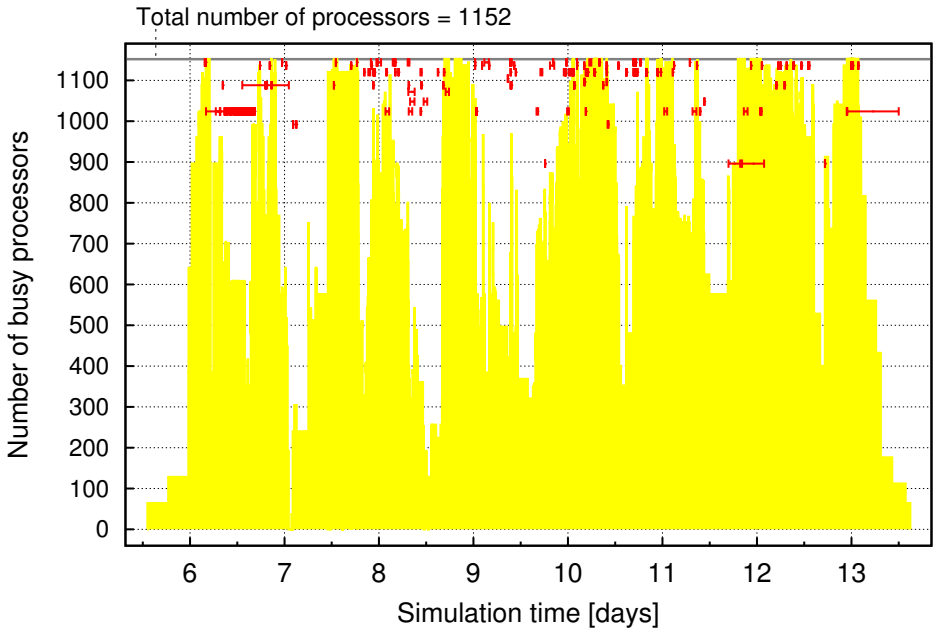

Figure 9.17: Example distribution of the requested duration (horizontal error bars) drawn at the reservation's earliest start time vs. the background utilization.

9.6.3 Parameters of the Simulations

For each method (p_{res}: static/history; fit: load/what-if) we performed 36 experiments varying the book-ahead time and the start time flexibility. The book-ahead times bat were 0, 2, 4, 6, 12 and 24 hours. The values of the start time flexibility stf were 0, 1, 2, 5, 10 and 30 hours. For all experiments, the threshold for filtering reserve messages at the LRS was arbitrarily set to 0.85. Table 9.3 lists the values of the main parameters specific to the methods.

Table 9.3: Values of the parameters used in the experiments.

Method	Param.=Value	Note
static	$h = 11386$	Slots with 6 hours book-ahead time receive a p_{res} value of 0.85.
history	$ap = 86400$	Averaging period of 86400 seconds (1 day)
load	$\delta_{rj} = 0.5$	Accuracy of the wall clock time of the running jobs
load	$\delta_{wj} = 0.5$	Accuracy of the wall clock time of the waiting jobs
what-if	$\omega_{C_{max}} = 0.1$	Weight of the makespan
what-if	$\omega_{C_{avg}} = 0.9$	Weight of the completion time

9.6.4 Performance Metrics

We measured different performance metrics to study the behavior of the probing mechanism. The **efficiency** of the methods was analyzed by counting the number of reserve messages sent between the GRS and the LRS as well as between the LRS and the LRMS. The LRS is not just passing the messages to the LRMS, but performs a filtering of incoming messages to ensure reservation admission policies. In particular, it may use the methods for calculating the properties of a time-qos-slot and let only those messages pass through whose properties' values are within a certain range or above/below a configurable threshold. We measured the number of reserve messages for both successful and failed reservation requests.

We studied the **impact** of the successful reservation requests on the non-reservation jobs by calculating the **delay** of the jobs. The more jobs are delayed by reservations, the less the users will be satisfied. Finally, we measured the **number of successful reservation requests** out of the 200.

9.6.5 Simulation Results and Discussion

We present the main results from two extensive sets of experiments performed for two publications [RSR06, RR06]. We used the same workload log as basis for the two sets of experiments.

Reservation Success Rate

The reservation success rate is defined as the ratio of the granted reservation requests to the submitted reservation requests. We expect the success rate to grow with an earlier booking (larger book-ahead time) and an increased execution time window (larger start time flexibility). This is because the former reduces the competition of concurrent jobs, while the latter increases the flexibility in arranging a favorite starting time.

Table 9.4 compares the average reservation success rate of all workloads in an experiment. We find that the more information a method is using, the higher is the reservation success rate. The methods are ordered (1) *static*, (2) *load*, (3) *history* and (4) *what-if* wrt. the amount of information they use.

Table 9.4: Comparison of the average reservation success rate.

	Method of property p_{res}		Method of property fit	
LRS filter	*static*	*history*	*load*	*what-if*
load	62.26 %	84.51 %	79.74 %	84.92 %
what-if	68.81 %	96.99 %	79.74 %	97.43 %

Figure 9.18 shows the results for the method *what-if* using the admission filter *load*. The book-ahead time is depicted at the horizontal axis. The reservation success rate is shown at the vertical axis. Each curve represents a set of simulation runs with the same start time flexibility.

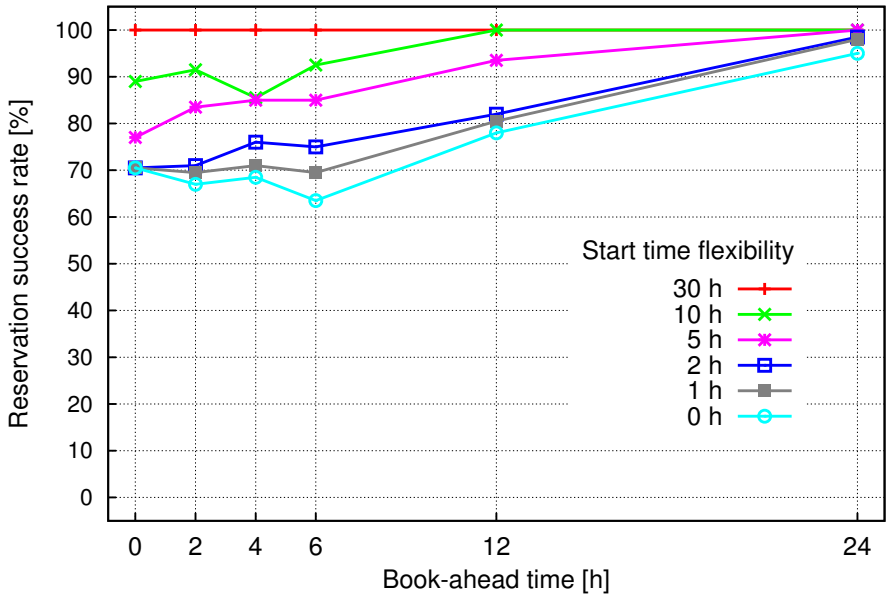

Figure 9.18: Reservation success rate for the 200 reservation requests using the method *what-if* for the property fit and the *load*-based LRS admission filter.

Figure 9.19 shows the results for the method *what-if* using the admission filter *what-if*. The book-ahead time is depicted at the horizontal axis. The reservation success rate is shown at the vertical axis. Each curve represents a set of simulation runs with the same start time flexibility.

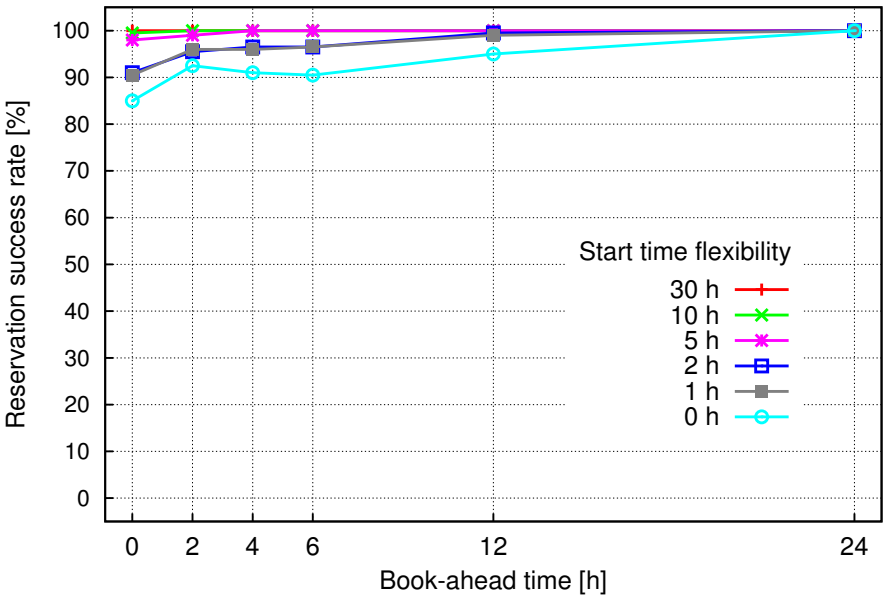

Figure 9.19: Reservation success rate for the 200 reservation requests using the method *what-if* for the property fit and the *what-if*-based LRS admission filter.

The results not only confirm our conjecture, but also illustrate how fast the success rate grows with respect to the added flexibility for choosing the start time (different curves) and the book-ahead time (x-axis). Furthermore, the admission filter based on the method *load* is more restrictive than the one based on the method *what-if*.

Efficiency of the Reservation Algorithm

Good reservation algorithms must efficiently handle both cases, successful and unsuccessful reservation requests. The philosophy of CORES is to avoid reserve messages that might fail. It uses the information gathered via the probe messages, in particular, the reservation success probability p_{res} and the fitness fit. After receiving the reservation candidates as response to a probe message they are processed in three steps.

First, the Grid Reservation Service (GRS) compares the p_{res} or fit value with a configurable threshold. In the experiments, only candidates with a value greater or equal to 0.85 are processed further. For example, using the method *static* all reservation requests with a book-ahead time of more than six hours would pass this check while all others would be rejected (cf. Fig. 9.7). Similarly, using the method *history* a reservation request would be accepted if more than 1.74-times of the requested number of processors are available during the requested period of time (cf. Fig. 9.8).

Second, the Local Reservation Service (LRS) verifies whether the currently known workload of the resource permits the request to proceed. At this point, CORES is able to enforce that reservations may not overtake waiting jobs. Also, the impact of reservations on the normal jobs may be limited. Note, that this and the following check are more expensive than the GRS check (see above), because they involve communication with and processing at the resource site. Hence, a high filtering efficiency at the GRS layer is of **prime** importance.

Third, the candidate is passed to the local resource management system (LRMS). The better the GRS threshold filter and the LRS checks are, the less requests will fail at the local scheduler. Nevertheless, such failures may happen when the values of the properties were inaccurate or when the requested capacity has been allocated to other workload entities in the meantime.

Tables 9.5 and 9.6 show the detailed results for all simulations. Each column contains the average values for a series of 36 experiments using the same method to calculate the property and the same method for the LRS filter. The experiments, however, differed in the book-ahead time and the start time flexibility. Note, because of the additional *Job Placeholder* execution plan, 52 reservation candidates were calculated with the method *what-if*.

Table 9.5: Efficiency of the methods *static*, *history*, *load* and *what-if* to calculate p_{res} and fit wrt. the **successful** requests. The numbers represent the average values for all 36 experiments tested with each combination of a property and an LRS filter.

	LRS filter: *load*				LRS filter: *what-if*			
	static	*history*	*load*	*what-if*	*static*	*history*	*load*	*what-if*
No. of requests	125	169	159	170	138	194	159	195
No. of candidates	51	51	51	52	51	51	51	52
Filtered by GRS (%)	32.3	33.2	38.9	35.2	34.5	36.2	38.9	38.8
Filtered by LRS (%)	4.8	5.4	0.0	4.8	0.6	1.0	0.1	0.0
Tested by LRMS (%)	0.1	0.2	0.1	0.0	0.0	0.0	0.0	0.0
Successful (%)	2.0	2.0	2.0	1.9	2.0	2.0	2.0	1.9
Not tested (%)	60.8	59.2	59.0	58.1	62.9	60.8	59.0	59.3

From the results for the successful requests we derive the following two findings.

1. Both methods used as LRS filter serve the goal of avoiding a large number of reserve messages to the LRMS very well. The method *what-if* performs marginally better than the method *load* (by $\approx 5\,\%$).

2. The *what-if*-based LRS filter increases the average number of successful requests by up to $15\,\%$.

Table 9.6: Efficiency of the methods *static*, *history*, *load* and *what-if* to calculate p_{res} and fit wrt. the **failed** requests. The numbers represent the average values for all 36 experiments tested with each combination of a property and an LRS filter.

	LRS filter: *load*				LRS filter: *what-if*			
	static	*history*	*load*	*what-if*	*static*	*history*	*load*	*what-if*
No. of requests	75	31	41	30	62	6	41	5
No. of candidates	51	51	51	52	51	51	51	52
Filtered by GRS (%)	91.8	61.3	99.9	68.9	99.5	80.9	99.9	100.0
Filtered by LRS (%)	8.2	38.3	0.0	31.1	0.5	19.1	0.1	0.0
Tested by LRMS (%)	0.0	0.4	0.1	0.0	0.0	0.0	0.0	0.0

From the results for the failed requests we derive the following four findings.

1. As for the successful requests, both methods used as LRS filter avoid sending most of the reserve messages to the LRMS.
2. The LRS filter *load* removes up to $30\,\%$ more candidates than the *what-if*-based filter. These candidates could be successfully reserved as the results (successful requests) for the *what-if*-based LRS filter show.
3. The main reason for the higher number of failed requests with the methods *static* and *load* is their inability to consider time-qos-slots before some threshold, i.e., 6 hours for method *static* with $h = 11386$ and the backlog time T_{WKL} for the method *load*.
4. The property's value calculated by the method *history* is too high. Even the optimistic *what-if*-based LRSfilters out $\approx 20\,\%$ of the candidates.

In summary, the method *what-if* performs best as LRS filter and achieves the best results for calculating the property's value for both successful and failed requests. The method *history* performs seconds best with respect to the number of successful requests. For the failed requests, it requires a large number of message exchanges between the GRS and the LRS, especially if the *load*-based LRS filter is used. Despite its high efficiency, the method *load* achieves the third place, because of the low number of successful requests. The property's values calculated by this method are too conservative. The method *static* performs "worst". It is, however, a very simple method, which considers the book-ahead time of a reservation request, but not the current workload of a resource.

Table 9.7: Comparison of the makespan results: Each row shows the averages for a series of 36 experiments using the same method of the property and the LRS filter.

Series (*property / LRS filter*)	Succ. req.	Makespan (s)	Makespan (%)
Batch jobs only	–	697,197	100.0
static / load	125	+13,210	101.9
static / what-if	138	+16,325	102.3
history / load	169	+14,026	102.0
history / what-if	194	+16,724	102.4
load / load	159	+13,491	101.9
load / what-if	159	+13,489	101.9
what-if / load	170	+14,087	102.0
what-if / what-if	195	+17,152	102.5

Impact on Non-Reservation Jobs

Reservations reduce the scheduling flexibility for non-reservation jobs. The proposed reservation algorithm assumes that jobs and reservations access resources exclusively. That means, before a reservation can become active, all user jobs accessing the reserved resources must have finished. Therefore the LRMS's scheduler starts only jobs which are guaranteed to terminate before the reservation begins. Intuitively, this leads to a *lower utilization* of resources and lets user jobs experience a *higher delay* before they can start. The makespan, i.e., the execution time of the complete schedule, and response time, i.e., the time each job spends in the system, are considered to be central metrics for measuring job management performance [FR98]. In the following, we present an overview and a detailed study of the results of 288 experiments for all combinations of the properties and the LRS filters.

Average Makespan Results. Table 9.7 shows the results for the makespan. The workload of the 1,800 jobs without any reservation has a makespan of 697,197 seconds. With reservation requests the makespan was extended by 13,210 seconds (*static / load*-filter) to 17,152 seconds (*what-if / what-if*-filter). The LRS filter *what-if* results in a higher increase in the makespan for all property methods (except the method *load*). The higher increase is a result of admitting reservations with earlier begin times.

Average Response Time Results. While the makespan (mainly) reflects the situation at the end of the batch workload, results for the response time cover all jobs of a workload. Table 9.8 shows the results for the jobs which were delayed by reservations. The workload of the 1,800 jobs without any reservation was used as reference run (column "Original"). Table 9.8 unveils that only 14 to 18 % of all jobs are delayed.

Table 9.8: Comparison of the response time results: Each row shows the averages for a series of 36 experiments using the same method of the property and the LRS filter.

Series (*property/LRS filter*)	Number of jobs		Response time per job		
	Absolute	(%)	Original	Affected	$\frac{\text{Affected}}{\text{Original}}$
static/load	260	14.44	5462	11800	2.16
static/what-if	272	15.11	5853	13197	2.25
history/load	302	16.78	8185	14812	1.81
history/what-if	324	18.00	8648	16129	1.87
load/load	295	16.39	7491	14086	1.88
load/what-if	295	16.39	7483	14079	1.88
what-if/load	302	16.78	8172	14790	1.81
what-if/what-if	327	18.17	8622	16197	1.88

The results for the property methods *history* and *what-if* are very similar. The increase of the number of delayed jobs between the LRS filter *load* and *what-if* is a consequence of the increased number of successful reservation requests (cf. Table 9.7). The results for the property method *load* (for both LRS filters) illustrate, again, that it derives more conservative values than the method *what-if* does. The method *static* yields the lowest number of delayed jobs, but it also leads to the lowest number of successful reservation requests (cf. Table 9.7).

Detailed Analysis of the Job Delays. In the following, we provide a detailed analysis of the job delays wrt. to the following questions:

Q1 – *How does the book-ahead time influence the job delays?*

Q2 – *How are the delays distributed among the jobs?*

Q3 – *How are the delays distributed along the simulation time?*

Due to the vast amount of data, we only present the detailed results for the experiments using the property method *what-if* with the LRS filter *what-if* and a start time flexibility of 30 hours. In the six remaining experiments, only the book-ahead time differed (0, 2, 4, 6, 12 and 24 hours).

Question Q1. Table 9.9 shows that the average response time correlates with the book-ahead time. In general, the higher is the book-ahead time, the more jobs ($9.55\,\%$ - $23.83\,\%$) are delayed. The relation is, however, not monotonic: the largest book-ahead time does not need to incur the largest response time. The average ratio of the affected to the original response time per job ranges between 1.37 and 2.59.

Table 9.9: Details on the response time of delayed jobs for the experiments using the property method *what-if*, the LRS filter *what-if* and a start time flexibility of 30 hours.

	Number of jobs		Response time per job		
Book-ahead time (hours)	Absolute	(%)	Original	Affected	$\frac{\text{Affected}}{\text{Original}}$
0	172	9.55	10676	14633	1.37
2	280	15.55	8922	14072	1.57
4	319	17.72	9249	14951	1.61
6	429	23.83	8228	16882	2.05
12	393	21.83	9239	23962	2.59
24	407	22.61	7444	18808	2.52
∅ (cf. Table 9.8)	327	18.17	8622	16197	1.88

Question Q2. Figures 9.20 and 9.21 illustrate the delay ratios for individual jobs of the experiments with 0 and 24 hours book-ahead time, respectively. The jobs' original response time is denoted on the horizontal axis. The vertical axis denotes the ratio of the affected (delayed) to the original response time. The jobs are categorized into five subjectively defined classes (colored crosses separated by lines, cf. Table 9.10).

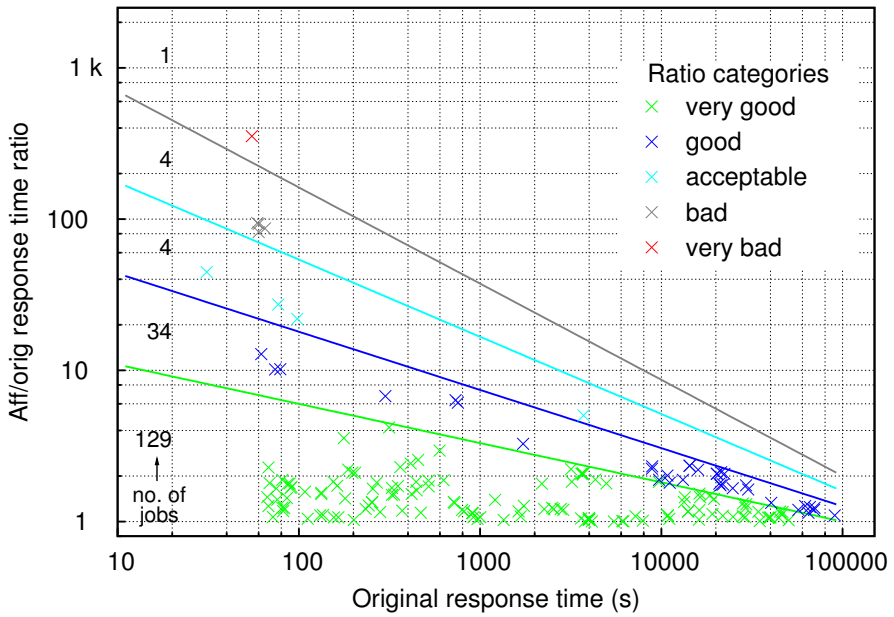

Figure 9.20: Job delay ratios for a book-ahead time of 0 hours.

Figures 9.20 and 9.21 show that most jobs are found in the classes *very good*, *good* and *acceptable*. While the total number of delayed jobs increases with larger book-ahead time, these jobs are not evenly distributed over all classes. The class *very good* gets

Table 9.10: Job delay classes with j_o (original) and j_d (delayed) response times.

Class	Definition
very good	$\frac{j_d}{j_o} < e^{2.986-0.259 \log j_o}$
good	$e^{2.986-0.259 \log j_o} \leq \frac{j_d}{j_o} < e^{4.663-0.384 \log j_o}$
acceptable	$e^{4.663-0.384 \log j_o} \leq \frac{j_d}{j_o} < e^{6.340-0.510 \log j_o}$
bad	$e^{6.340-0.510 \log j_o} \leq \frac{j_d}{j_o} < e^{8.017-0.636 \log j_o}$
very bad	$e^{8.017-0.636 \log j_o} \leq \frac{j_d}{j_o}$

saturated around 180 jobs (approximately $10\,\%$ of all jobs). The other classes receive increasing numbers of jobs with larger book-ahead times.

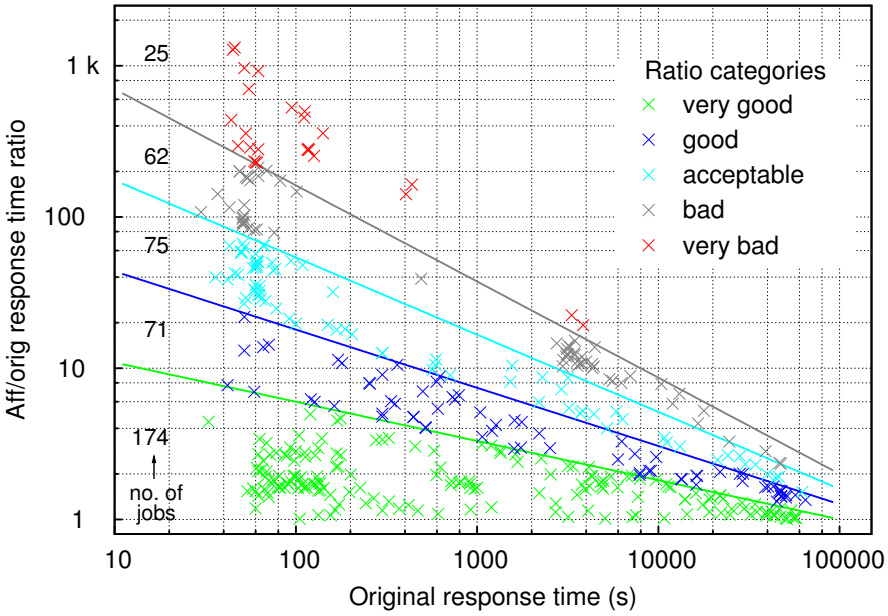

Figure 9.21: Job delay ratios for a book-ahead time of 24 hours.

Question Q3. Figure 9.22 shows the cumulated additional waiting time of the delayed jobs for the six experiments with the property method *what-if*, the LRS filter *what-if* and a start time flexibility of 30 hours. For each job, its delay was cumulated at the job's start time. The curves show regions with no or small increases and "events" (or short time frames) with significant increases of the cumulative delay.

The sharp increases of the curves show situations in which blocking jobs or reservations finished and many waiting jobs could start in parallel. A detailed analysis of the logs revealed a specific pattern illustrated in Fig. 9.23. At the current time 'now'

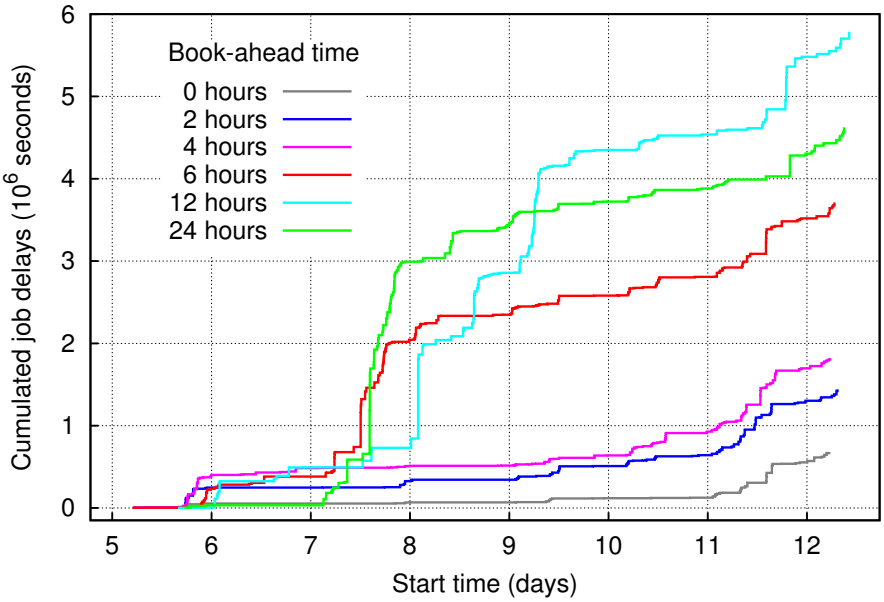

Figure 9.22: Cumulated additional waiting time of delayed jobs for the experiments with the property method *what-if*, LRS filter *what-if* and start time flexibility of 30 hours.

the jobs $RJ_{1/2/3/4}$ are being executed. Granted advance reservations are shown along the horizontal axis (marked with RSV). These reservations were small in their size (duration and number of processors). The waiting jobs $WJ_{1/2/3}$ are planned to start after the last reservation.

The observed behavior (sharp increases) is caused by the very large job WJ_1 which is planned to start at the end of the last reservation. The number of processors of this job is close to the total available number of processors of the resource. Therefore the job is either blocked by already running jobs or by the sequence of reservations. With EASY backfilling, the job is planned into the schedule to prevent its starvation, after it has advanced to the head of the waiting queue. No other job is executed before the large job, because they either request too many processors or too much execution time.

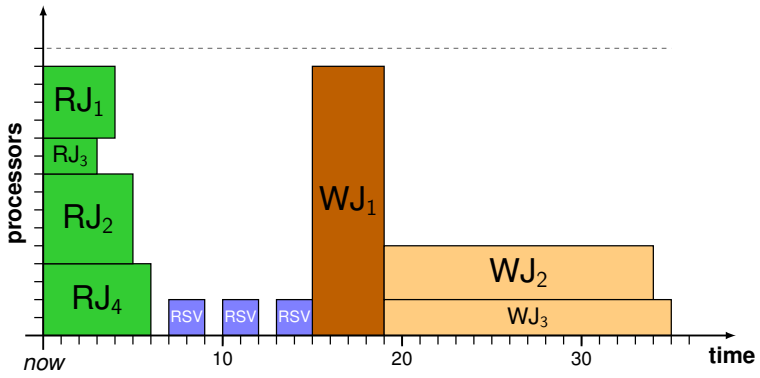

Figure 9.23: Pattern causing significant delays of jobs.

The number and the sizes of the sharp increases may be reduced by

Job size limitation: We repeated the experiments with modified batch workloads by limiting the number of requested processors to 88 (affecting 10 of 1,800 jobs). The delays were significantly reduced.

Improving execution time estimates: Jobs with more accurate – *smaller* – execution time estimates could fit in the hole before a large job. The situation we observed will not disappear completely, but the number of delayed jobs and their delays could be reduced.

9.7 Summary

We presented a generic mechanism for deriving the future status of a resource. Given a flexible reservation request, the mechanism derives

1. a distribution of time-qos-slots,
2. the values for a list of properties at each time-qos-slot, *and*
3. interpolates values at intermediate time-qos-slots.

We demonstrated the applicability of the mechanism by several methods for deriving the *reservation success probability* p_{res} and the *fitness* fit.

These methods were experimentally evaluated for compute resources wrt. the **reservation success rate**, the **efficiency** of the reservation algorithm and the **impact** on non-reservation jobs. We found that the methods performed in the following order (best to worst): (1) *what-if*, (2) *history*, (3) *load*, and (4) *static*. That order is reciprocally proportional to the amount of information used by the methods and their complexity.

We also found that the impact on non-reservation jobs is sensitive on the book-ahead time. A detailed analysis of the job delays showed that a high fraction of the delays is incurred by a few events. These events show a common pattern, whose amplitude may be reduced by limiting the sizes of parallel jobs and by enforcing more accurate job runtime estimates.

Chapter 10

Mapping Requests to Co-Reservation Candidates

We present two models for mapping requests to co-reservation candidates. A co-reservation candidate is composed of multiple reservation candidates, which were derived through the probing step. Particularly, the mapping assigns each atomic request to a resource at a begin time, for a duration and a service level. The models implement the mathematical formalization of the co-reservation problem (cf. Chapter 5) as integer program (IP) and as binary program (BP).

Chapter Outline. We list requirements on mapping mechanisms in Section 10.1. In Section 10.2, we discuss related work. Thereafter, we incrementally develop the IP model in Section 10.3. Next, the BP model is developed in Section 10.4. We experimentally evaluate the scalability of the IP and BP models in Section 10.5. In Section 10.6, we study several strategies for refining the models in case of failures and change requests. We close the chapter with a summary in Section 10.7.

10.1 Requirements

Mechanisms for mapping co-reservation requests to candidates must fulfill several requirements which are listed below.

R1 – Fairness Among Users and Resources

The mapping shall acknowledge the constraints and objectives of both the users and the resource providers. This requirement is similar to *symmetric matching* provided by Condor ClassAds [RLS98], RedLine [LF03b] and GRDL [SR06] (cf. Section 7.2.3).

R2 – Flexible Construction of Constraints and Objectives

The mechanisms shall support multiple kinds of constraints and objectives. First, this requirement addresses the type of functions used for aggregating properties to constraints and objectives. Second, it concerns the "reach" of single constraints and objectives, i.e., the means to specify relationships among parts of a co-reservation. Third, it deals with the ability to specify constraints and objectives on any property.

R3 – Efficient Processing of Problem Instances

Because the state of the Grid environment may change quickly, the mechanisms need to find a solution as soon as possible. If the search space is too large, it shall be easy to reduce it.

R4 – Effective Means for Determining Alternative Mappings

Even if a solution is found quickly, a subsequent reserve message may fail because of stiff competition or inaccurately calculated properties. In that case, the problem shall easily be refined to find alternative mappings.

10.2 State of the Art

Previous work on scheduling multiple advance reservations focused on very specific applications and only partially supported the generic model presented in Chapter 5. In contrast, the approach of CORES supports a wide variety of scenarios in terms of the structure of the applications, the types of the reservable resources and the means to specify constraints and objectives.

The VIOLA meta-scheduler [WWZ05] schedules rigid compute jobs to multiple pre-selected resources by incrementally increasing the advance booking period until all jobs may be allocated. In contrast to our work, it only supports one criteria – the earliest completion time. It also lacks support for arbitrary constraints.

In [WPH07], Wieczorek et al. present a taxonomy of the multi-criteria Grid workflow scheduling problem. In contrast to our generic model, none of the analyzed approaches supports all capabilities, i.e., multiple criteria, multiple job chains, advance reservations, moldable requests and different types of requests (compute, network, storage, etc.). In the following, we discuss the most advanced approaches studied in [WPH07]. Brandic et al. [BBES05] propose a workflow engine which supports quality-of-service. First, for each workflow activity, it contacts candidate services (cf. eligible resources in our model) and negotiates a single service level acknowledging the desired characteristics such as maximum execution time and maximum price. Second, it uses *integer programming* to assign each activity to a single candidate service such that the utility function is maximized. The utility function is the weighted sum of the objectives of each activity and the overall workflow. Each activity as well as

the overall workflow may specify multiple objectives. The major differences to our approach are the single offer, the missing support for constraints and objectives of the resources and the lack of temporal and spatial relationships among the activities. In our model, the number of offers is only limited by the domains of the start time, the duration and the service level.

The management of workflows has also been extensively studied in the context of web services. In [ZBN+04], Zeng et al. present a middleware for selecting web service instances to compose complex workflows. In particular, they propose a QoS model for atomic web services and for composite services. Based on this model, the middleware implements a QoS-aware selection of web services such that the user's satisfaction is maximized. The user's satisfaction is defined as the weighted sum of multiple criteria chosen by the user itself. The actual selection of the web services is implemented using *integer programming*. Besides constraints on each individual workflow activity, their implementation supports global constraints on the aggregated values of individual activities. For example, the total budget for executing a workflow or its total execution time may be limited. A side effect of constraining the execution time is the creation of a schedule, that is, a solution assigns a start time to each activity. In contrast, our model explicitly considers the start time as a variable. Additionally, the models of CORES support moldable service levels and durations to optimize the user's satisfaction. Also, the resources (or web services) may specify constraints and objectives for each involved entity. Moreover, our approach provides rich capabilities for specifying temporal and spatial relationships and allows to define global constraints and objectives explicitly.

In [CPEV05], Canfora et al. apply *genetic algorithms* to optimize the assignment of workflow activities to candidate services. Their approach supports user defined constraints and objectives for each activity. The objective of the whole workflow is constructed by aggregating the objectives of the individual activities. Canfora et al. describe different aggregation functions depending on the type of the objective (e.g., cost, time, availability, etc.) and the structural relationships among the activities (e.g., sequence, switch, loop, etc.). In our model, we only support the weighted sum as aggregation function. Moreover, we do not need to distinguish between different structural elements of a workflow, since we assume that all activities must be executed. Because we aim at using standard solvers, all constraints and objectives must be linear functions. In contrast, the use of genetic algorithms allows to use arbitrary functions. The main difference to our work, however, is the single variable s (representing the selected resource) per activity. Additionally, our model supports variables for the start time, the duration and the service level. Of course, that flexibility comes at a high cost, namely, the size of the search space.

Table 10.1 provides an overview of the discussed related work and compares them against the approach of CORES.

Table 10.1: Existing approaches to schedule multiple advance reservations and workflows. Symbols: **side** ($\mathcal{R}$ - request, $\mathcal{S}$ - resource), **types of entities** (any - compute, network, data, license, ...; ws - web service), **variables per assignment** (s - resource, t - start time, q - service level, d - duration), **CONstraints** (upper left {no.}: S-single, M-multiple; upper right {properties}: variables t, q, d, A-any; lower left {specification}: I-implicitly, E-explicitly; lower right {functions / comparison}: C-constant, L-linear, A-arbitrary / $=$-equality, $\geq$-inequality) **OBJectives** (upper left: cf. CON; upper right: cf. CON + optimization goal; lower left: cf. CON; lower right {functions / aggregation}: cf. CON / ω-weighted sum, p-Pareto set), **TEMPoral relationships** (ST - same time, SEQ - sequence, OL - partially overlapping), **SPATial relationships** (net - network, nnt - non-network), **techniques** (GA - genetic algorithms, IP/BP - integer/binary programming, T&E - trial and error).

Approach	No. of requests	Types of entities	Assignm. variables	Modeling of an atomic entity: $CON_{\mathcal{R}}$	$OBJ_{\mathcal{R}}$	$CON_{\mathcal{S}}$	$OBJ_{\mathcal{S}}$	Modeling of a co-reservation: TEMP	SPAT	OBJ	CON	Applied technique
Reservation based Meta-Scheduling												
VIOLA [WWZ05]	1	any	t	M t,q,d I $C_{=}$	- - - -	M A I $C^{\geq}$	- - - -	ST	no	- - - -	S $\min t$ E NL	T&E
Workflow Management												
VGE [BBES05]	1	ws	s, t	M A I $C^{\geq}$	M max E L_{ω}	M A I $C_{=}$	- - - -	no	no	M A I $C^{\geq}$	M max E L_{ω}	IP
Miscellaneous Approaches												
Zeng et al. [ZBN+04]	1	ws	s, t	M A I $C^{\geq}$	M A E C_{ω}	M A I $C_{=}$	- - - -	SEQ	no	M A I $A^{\geq}$	M A I C_{ω}	IP
SeCSE [CPEV05]	1	ws	s	M A I $C^{\geq}$	M max E C_{any}	M A I $C_{=}$	- - - -	no	no	M A I $A^{\geq}$	M max I A_{any}	GA
Our Approach												
CORES (this work)	≥ 1	any	$s, t,$ q, d	M A E $L_{=}^{\geq}$	M A E L_{ω}	M A E $L_{=}^{\geq}$	M A E L_{ω}	SEQ, ST, OL	net nnt	M A E $L_{=}^{\geq}$	M A E L_{ω}	IP, BP

10.3 Modeling as Integer Problem

We incrementally develop the integer model of the co-reservation problem beginning with the variables, continuing with basic integrity constraints, generic constraints, temporal and spatial relationships and concluding with the objective function.

Requests and Resources

The requests R and the resources S are defined as in Def. 1 (cf. page 23). The set of eligible resources $S(r)$ is defined as in Chapter 5 (cf. page 24).

Variables and their Domains

We associate four variables with each possible assignment $r_l \triangleright s_k$:

- a binary variable $x_{l,k} \in \{0, 1\}$,
- an integer variable $t_{l,k} \in T \subset \mathbb{N}$ representing the start time,
- an integer variable $d_{l,k} \in D \subset \mathbb{N}$ representing the duration, *and*
- an integer variable $q_{l,k} \in Q \subset \mathbb{N}$ representing the service level.

The sets T, D, Q are defined as $T = \{0\} \cup [T_{LO}, T_{UP}]$, $D = \{0\} \cup [D_{LO}, D_{UP}]$ and $Q = \{0\} \cup [Q_{LO}, Q_{UP}]$, respectively. The terms T_{LO}, D_{LO}, Q_{LO} and T_{UP}, D_{UP}, Q_{UP} denote the lower (subscript LO) and the upper (subscript UP) bounds of the variables' domain.

We assume that all properties, constraints and objectives (see below) are linear combinations of the variables $x_{l,k}$, $t_{l,k}$, $d_{l,k}$ and $q_{l,k}$.

A binary variable $x_{l,k}$ is set to 1 **iff** the request r_l is mapped to the resource s_k. The actual solution variables corresponding to the atomic request r_l (cf. Def. 2), comprising of the resource $V_s(r_l)$, the start time $V_t(r_l)$, the duration $V_d(r_l)$ and the service level $V_q(r_l)$ are derived as follows

$$V_s(r_l) = \sum_{k=1}^{K} k\, x_{l,k}\,, \quad V_t(r_l) = \sum_{k=1}^{K} t_{l,k}\, x_{l,k}\,, \quad V_d(r_l) = \sum_{k=1}^{K} d_{l,k}\, x_{l,k}\,, \quad V_q(r_l) = \sum_{k=1}^{K} q_{l,k}\, x_{l,k}\,.$$

Integrity of a Solution

We ensure the integrity of a solution by two restrictions constituting the set SIC (cf. Chapter 5). Because a request r_l shall be mapped *at most once* to a resource, we constrain the values of the binary variables as follows

$$\forall r_l \in R: \quad \sum_{s_k \in S(r_l)} x_{l,k} \;\leq\; 1\,.$$

The values of the variables $t_{l,k}$, $d_{l,k}$ and $q_{l,k}$ are further constrained by

$$\forall r_l \in R \quad \forall s_k \in S(r_l): \qquad \begin{aligned} t_{l,k} \geq x_{l,k}\, T_{LO}\,, \quad t_{l,k} \leq x_{l,k}\, T_{UP}\,, \\ d_{l,k} \geq x_{l,k}\, D_{LO}\,, \quad d_{l,k} \leq x_{l,k}\, D_{UP}\,, \\ q_{l,k} \geq x_{l,k}\, Q_{LO}\,, \quad q_{l,k} \leq x_{l,k}\, Q_{UP}\,. \end{aligned}$$

All these constraints ensure that the variables $t/d/q_{l,k}$ may be set to values in their domain if the request r_l is assigned to the resource s_k. Otherwise these variables are set to zero.

Properties

Any property $p^{id}_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))$ can be written as **linear** combination

$$p^{id}_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l)) \;=\; \alpha^{id}_{l,k}\, V_t(r_l) \;+\; \beta^{id}_{l,k}\, V_d(r_l) \;+\; \gamma^{id}_{l,k}\, V_q(r_l) \;+\; \delta^{id}_{l,k}\,,$$

with id being an identifier and $\alpha^{id}_{l,k}$, $\beta^{id}_{l,k}$, $\gamma^{id}_{l,k}$, $\delta^{id}_{l,k} \in \mathbb{R}$. We assume that the properties of all assignments are uniformly enumerated from 1 to M. That is, a property is identified by the same number id for all assignments.

Constraints on Single Assignments

We implement constraints on single assignments $r_l \triangleright s_k$ by the following expression

$$\forall r_l \in R,\ \forall s_k \in S(r),\ \forall sac^{cop} \in SAC_{r_l \triangleright s_k}:\ \sum_{m=1}^{M} sac_m\, p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l)) \quad cop \quad 0\,.$$

The set $SAC_{r_l \triangleright s_k}$ contains all single assignment constraints sac^{cop} of the assignment $r_l \triangleright s_k$ with cop being the comparison operator (either $=$ or $\geq$). The term sac_m represents the aggregation coefficient of the property $p^m_{r_l \triangleright s_k}$.

> **Example 10.1 (Deadline of a sequential co-reservation)**
> *We illustrate the implementation of the constraint of Example 5.1 (cf. Chapter 5). The properties $p^{stt}_{r3\triangleright}$, $p^{dur}_{r3\triangleright}$ and $p^{dl}_{r3\triangleright}$ are identified by the numbers 1, 2 and 3, respectively. The comparison operator cop is $\geq$ (greater than or equal). The $M = 3$ aggregation coefficients are $sac_1 = -1$, $sac_2 = -1$ and $sac_3 = 1$.*

Temporal Relationships

We implement temporal relationships by the following expression

$$\forall tr^{cop} \in TR:\ \sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{m=1}^{M} tr_m(r_l \triangleright s_k)\, p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))\ cop\ \ 0\,.$$

The set TR contains all temporal relationships tr^{cop} with the comparison operator cop. The term $tr_m(r_l \triangleright s_k)$ represents the aggregation coefficient of the property $p^m_{r_l \triangleright s_k}$. That is, a temporal relationship is defined by appropriate values of the $tr_m(r_l \triangleright s_k)$.

Example 10.2 (Sequential Job Chain)
We illustrate the implementation of the constraint of Example 5.2 with $L = 3$ steps (cf. Chapter 5). The $M = 3$ properties are enumerated as in Example 10.1. We use the same resources ($K = 7$) and assignments $A_{1/2}$, $B_{1/2/3}$ and $C_{1/2}$ as in Example 5.3. The $M \cdot K$ aggregation coefficients are

$$\begin{array}{lllllllll} tr_1(A_1) = -1, & tr_2(A_1) = -1, & tr_3(A_1) = 0\,; & tr_1(A_2) = -1, & tr_2(A_2) = -1, & tr_3(A_2) = 0\,; \\ tr_1(B_1) = 1, & tr_2(B_1) = 0, & tr_3(B_1) = 0\,; & tr_1(B_2) = 1, & tr_2(B_2) = 0, & tr_3(B_2) = 0\,; \\ tr_1(B_3) = 1, & tr_2(B_3) = 0, & tr_3(B_3) = 0\,; & & & \\ tr_1(C_1) = 0, & tr_2(C_1) = 0, & tr_3(C_1) = 0\,; & tr_1(C_2) = 0, & tr_2(C_2) = 0, & tr_3(C_2) = 0\,. \end{array}$$

Spatial Relationships

We implement non-network and network spatial relationships by the generic expression

$$\forall sr \in SR: \quad \sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{m=1}^{2} sr_m(r_l \triangleright s_k)\, p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l)) \;=\; 0\,.$$

The set SR contains all spatial relationships sr. The spatial properties p^1 and p^2 represent the "left" and the "right" end-point of a resource, respectively. These differ for network resources, but are the same for non-network resources. The term $sr_m(r_l \triangleright s_k)$ represents the aggregation coefficient of the spatial property $p^m_{r_l \triangleright s_k}$. Thus, a spatial relationship is defined by appropriate values of the $sr_m(r_l \triangleright s_k)$.

Example 10.3 (Transfer of Data)
We illustrate the implementation of the spatial relationship of Example 5.3 by two constraints sr^a and sr^b. The example involves $L = 3$ steps (cf. Chapter 5). The spatial properties of the seven resources are given by the table

	Resource						
Property	s_1	s_2	s_3	s_4	s_5	s_6	s_7
p^1	1	1	9	1	3	2	2
p^2	1	9	9	3	3	3	2

We assume the same assignments as in Example 10.2. Most of the $2MK + 2MK$ ($2MK$ for each constraint) aggregation coefficients $sr^{a/b}_m(A_{1/2}/B_{1/2/3}/C_{1/2})$ are set to zero. The non-zero coefficients are

$$sr^a_1(A_1) = 1, sr^a_1(A_2) = 1, sr^a_1(B_1) = -1, sr^a_1(B_2) = -1, sr^a_1(B_3) = -1,$$
$$sr^b_2(B_1) = -1, sr^b_2(B_2) = -1, sr^b_2(B_3) = -1, sr^b_2(C_1) = 1, sr^b_2(C_2) = 1\,.$$

Note, the constraints sr^a and sr^b represent the left and the right element of the result space $\mathbb{R}^2$ of Def. 14 (cf. Chapter 5).

Constraints on Multi Assignments

We implement constraints on multi assignments as generalization of temporal relationships. Let M denote the total number of properties. Then, the following expression implements constraints on multi assignments.

$$\forall mac^{cop} \in MAC\colon \sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{m=1}^{M} mac_m(r_l \triangleright s_k)\, p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l)) \quad cop \quad 0$$

The set MAC contains all multi assignments constraints mac^{cop} with the comparison operator cop. The term $mac_m(r_l \triangleright s_k)$ represents the aggregation coefficient of the property $p^m_{r_l \triangleright s_k}$. Thus, a multi assignments constraint is implemented by appropriate values of the $mac_m(r_l \triangleright s_k)$.

Example 10.4 (Limiting the Total Reservation Cost)
We illustrate the implementation of the constraint of Example 5.4 with $L = 3$ steps (cf. Chapter 5). The reservation cost and the maximum budget are given by the properties p^1 and p^2, respectively ($M = 2$). We assume the same assignments as in Example 10.2. The $M \cdot K$ aggregation coefficients are

$$\begin{array}{llll} mac_1(A_1) = -1, & mac_2(A_1) = 1\,; & mac_1(A_2) = -1, & mac_2(A_2) = 1\,; \\ mac_1(B_1) = -1, & mac_2(B_1) = 0\,; & mac_1(B_2) = -1, & mac_2(B_2) = 0\,; \\ mac_1(B_3) = -1, & mac_2(B_3) = 0\,; & & \\ mac_1(C_1) = -1, & mac_2(C_1) = 0\,; & mac_1(C_2) = -1, & mac_2(C_2) = 0\,. \end{array}$$

Note, we arbitrarily associated the budget property with request r_1.

Objective Function

The objective function is composed of normalized properties $\|p^m_{r_l \triangleright s_k}\|_{p^m}$, aggregation coefficients $o_m(r_l \triangleright s_k)$ and the objectives' weight o^ω. We normalize a real-valued property $p^m_{r_l \triangleright s_k}$ as follows

$$\|p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))\|_{p^m} = \frac{p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))}{\max(|\min(p^m)|, |\max(p^m)|)} \tag{10.1}$$

$$\text{with} \qquad \min(p^m) = \min_{r_l \in R,\, s_k \in S(r_l)} p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))$$

$$\text{and} \qquad \max(p^m) = \max_{r_l \in R,\, s_k \in S(r_l)} p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))\,.$$

The term $o_m(r_l \triangleright s_k)$ denotes the m-th aggregation coefficient of the assignment $r_l \triangleright s_k$. Using the weighted sum as criteria B (cf. Def. 21), the objective function is defined by the following expression.

$$\begin{aligned} &\text{minimize} \\ &\sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{o \in O_R} o^\omega \sum_{m=1}^{M} o_m(r_l \triangleright s_k)\, \|p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))\|_{p^m} \\ &+ \sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{o \in O_{s_k}} o^\omega \sum_{m=1}^{M} o_m(r_l \triangleright s_k)\, \|p^m_{r_l \triangleright s_k}(V_t(r_l), V_d(r_l), V_q(r_l))\|_{p^m} \end{aligned}$$

The terms O_R and O_{s_k} denote the sets of the objectives of the co-reservation request R and the resource s_k ($k = 1, \ldots, K$), respectively.

Example 10.5 (Min End Time & Max Fitness)
We illustrate the implementation of Example 5.5 with $L = 3$ steps (cf. Chapter 5). The end time is calculated as the sum of the start time (property p^1) and the duration (property p^2) of request r_3. The fitness (cf. Section 9.4.2) is given by property p^3.

Because all entities – the co-reservation request and the resources – possess only a single objective all weights are set to 1. The properties are normalized as in Eq. (10.1). We assume the same assignments as in Example 10.2. Most of the $M \cdot K$ aggregation coefficients $o_{1/2/3}(A_{1/2}/B_{1/2/3}/C_{1/2})$ of the requests are zero. Also, most of the $M \cdot K$ aggregation coefficients of the resources are zero. The non-zero coefficients are

coefficients of the co-reservation request

$$o_1(C_1) = 1,\ o_2(C_1) = 1,\ o_3(C_1) = 0\,; \quad o_1(C_2) = 1,\ o_2(C_2) = 1,\ o_3(C_2) = 0\,;$$

coefficients of resources

$$o_3(A_1) = -1,\ o_3(A_2) = -1\,; \quad o_3(B_1) = -1,\ o_3(B_2) = -1,\ o_3(B_3) = -1\,;$$
$$o_3(C_1) = -1,\ o_3(C_2) = -1\,.$$

Note, the coefficients of the resources are negative to invert the optimization sense (minimize → maximize).

10.4 Modeling as Binary Problem

We incrementally develop the binary model of the co-reservation problem beginning with the variables, continuing with basic integrity constraints, generic constraints, temporal and spatial relationships and concluding with the objective function.

Requests and Resources

The requests R and the resources S are defined as in Def. 1 (cf. page 23). The set of eligible resources $S(r)$ is defined as in Chapter 5 (cf. page 24).

Variables and their Domains

In the binary model, the properties are only defined at specific tuples constructed from a matching request-resource-pair (r_l, s_k) and a time-qos-slot $\langle t, d, q\rangle$. Each tuple

$$\langle r_l, s_k, t, d, q\rangle$$

is represented by a **binary** variable

$$x_{\langle l,k,t,d,q\rangle} \in \{0, 1\}\,.$$

A binary variable $x_{\langle l,k,t,d,q\rangle}$ is set to 1 **iff** the request r_l is mapped to the resource s_k with the start time t, the duration d and the service level q.

Let $TDQ_{r_l \triangleright s_k}$ denote the set of all time-qos-slots derived for the assignment $r_l \triangleright s_k$. The actual solution variables corresponding to the atomic request r_l (cf. Def. 2), comprising of the resource $V_s(r_l)$, the start time $V_t(r_l)$, the duration $V_d(r_l)$ and the service level $V_q(r_l)$ are derived as follows

$$V_s(r_l) = \sum_{k=1}^{K} \sum_{\substack{\langle t,d,q\rangle \in \\ TDQ_{r_l \triangleright s_k}}} k\, x_{\langle l,k,t,d,q\rangle}\,, \quad V_t(r_l) = \sum_{k=1}^{K} \sum_{\substack{\langle t,d,q\rangle \in \\ TDQ_{r_l \triangleright s_k}}} t\, x_{\langle l,k,t,d,q\rangle}\,,$$

$$V_d(r_l) = \sum_{k=1}^{K} \sum_{\substack{\langle t,d,q\rangle \in \\ TDQ_{r_l \triangleright s_k}}} d\, x_{\langle l,k,t,d,q\rangle}\,, \quad V_q(r_l) = \sum_{k=1}^{K} \sum_{\substack{\langle t,d,q\rangle \in \\ TDQ_{r_l \triangleright s_k}}} q\, x_{\langle l,k,t,d,q\rangle}\,.$$

Integrity of a Solution

Let $TDQ_{r_l \triangleright}$ denote the set of all time-qos-slots derived for the set of assignments $r_l \triangleright$. Because a request r_l shall be mapped *at most once* to a resource and a time-qos-slot, we constrain the values of the binary variables by

$$\forall r_l \in R: \quad \sum_{s_k \in S(r_l)} \sum_{\langle t,d,q\rangle \in TDQ_{r_l \triangleright s_k}} x_{\langle l,k,t,d,q\rangle} \quad \leq \quad 1\,.$$

Properties

A property $p^{id}_{r_l \triangleright s_k}(t,d,q)$ with identifier id assigns a real-valued number to the tuple $\langle r_l, s_k, t, d, q\rangle$. We assume that all properties are enumerated from 1 to M.

Constraints on Single Assignments

We implement constraints on single assignments $r_l \triangleright s_k$ by

$$\forall r_l \in R,\ \forall s_k \in S(r),\ \forall sac^{cop} \in SAC_{r_l \triangleright s_k} : \sum_{\substack{\langle t,d,q\rangle \in \\ TDQ_{r_l \triangleright s_k}}} x_{\langle r_l, s_k, t, d, q\rangle} \left(\sum_{m=1}^{M} sac_m\, p^m_{r_l \triangleright s_k}(t,d,q) \right) \ cop\ 0\,.$$

The set $SAC_{r_l \triangleright s_k}$ contains all single assignment constraints sac^{cop} of the assignment $r_l \triangleright s_k$ with cop being the comparison operator (either $=$ or $\geq$). The term sac_m represents the aggregation coefficient of the property p^m.

Example 10.6 (Deadline of a sequential co-reservation)
We illustrate the implementation of the constraint of Example 5.1 (cf. Chapter 5). The properties $p^{stt}_{r3\triangleright}$, $p^{dur}_{r3\triangleright}$ and $p^{dl}_{r3\triangleright}$ are identified by the numbers 1, 2 and 3, respectively. The comparison operator cop is $\geq$ (greater than or equal). The $M = 3$ aggregation coefficients are $sac_1 = -1$, $sac_2 = -1$ and $sac_3 = 1$.

Temporal Relationships

We implement temporal relationships by the following expression

$$\forall tr^{cop} \in TR : \sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{\langle t,d,q\rangle \in TDQ_{r_l \triangleright s_k}} x_{\langle r_l,s_k,t,d,q\rangle} \left(\sum_{m=1}^{M} tr_m(r_l) p^m_{r_l \triangleright s_k}(t,d,q) \right) \; cop \;\; 0\,.$$

The set TR contains all temporal relationships tr^{cop} with the comparison operator cop. The term $tr_m(r_l)$ represents the aggregation coefficient of the property p^m and the request r_l. That is, a temporal relationship is defined by appropriate values of the $tr_m(r_l)$.

Example 10.7 (Sequential Job Chain)
We illustrate the implementation of the constraint of Example 5.2 with $L = 3$ steps (cf. Chapter 5). The $M = 3$ properties are enumerated as in Example 10.6. The $M \cdot L$ aggregation coefficients are

$$\begin{array}{lll} tr_1(r_1) = -1, & tr_1(r_2) = 1, & tr_1(r_3) = 0\,; \\ tr_2(r_1) = -1, & tr_2(r_2) = 0, & tr_2(r_3) = 0\,; \\ tr_3(r_1) = 0, & tr_3(r_2) = 0, & tr_3(r_3) = 0\,. \end{array}$$

Spatial Relationships

We implement the two types – non-network and network spatial relationships – by the generic expression

$$\forall sr \in SR : \sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{\substack{\langle t,d,q\rangle \in \\ TDQ_{r_l \triangleright s_k}}} x_{\langle r_l,s_k,t,d,q\rangle} \left(\sum_{m=1}^{2} sr_m(r_l) p^m_{r_l \triangleright s_k}(t,d,q) \right) \;=\; 0\,.$$

The set SR contains all spatial relationships sr. The spatial properties p^1 and p^2 represent the "left" and the "right" end-point of a resource, respectively. These differ for network resources, but are the same for non-network resources. The term $sr_m(r_l)$ represents the aggregation coefficient of the spatial property p^m for the request r_l. Thus, a spatial relationship is defined by appropriate values of the $sr_m(r_l)$.

Example 10.8 (Transfer of Data)
We illustrate the implementation of the spatial relationship of Example 5.3 by two constraints sr^a and sr^b. The example involves $L = 3$ steps (cf. Chapter 5). The spatial properties of the seven resources are given by the table

	Resource						
Property	s_1	s_2	s_3	s_4	s_5	s_6	s_7
p^1	1	1	9	1	3	2	2
p^2	1	9	9	3	3	3	2

The $2L + 2L$ ($2L$ for each constraint) aggregation coefficients are

$$\begin{array}{lll} sr_1^a(r_1) = 1, & sr_2^a(r_1) = 0 & sr_1^b(r_1) = 0, \quad sr_2^b(r_1) = 0 \\ sr_1^a(r_2) = -1, & sr_2^a(r_2) = 0 & sr_1^b(r_2) = 0, \quad sr_2^b(r_2) = -1 \\ sr_1^a(r_3) = 0, & sr_2^a(r_3) = 0 & sr_1^b(r_3) = 0, \quad sr_2^b(r_3) = 1\,. \end{array}$$

Note, the constraints sr^a and sr^b represent the left and the right element of the result space $\mathbb{R}^2$ of Def. 14 (cf. Chapter 5).

Constraints on Multi Assignments

We implement constraints on multi assignments as generalization of temporal relationships. Let M denote the total number of properties. Then, the following expression implements constraints on multi assignments.

$$\forall mac^{cop} \in MAC : \sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{\substack{\langle t,d,q \rangle \in \\ TDQ_{r_l \triangleright s_k}}} x_{\langle r_l, s_k, t, d, q \rangle} \left(\sum_{m=1}^{M} mac_m(r_l) p^m_{r_l \triangleright s_k}(t,d,q) \right) \quad cop \quad 0$$

The set MAC contains all multi assignments constraints mac^{cop} with the comparison operator cop. The term $mac_m(r_l)$ represents the aggregation coefficient of the property p^m and the request r_l. Thus, a multi assignments constraint is implemented by appropriate values of the $mac_m(r_l)$.

Example 10.9 (Limiting the Total Reservation Cost)
We illustrate the implementation of the constraint of Example 5.4 with $L = 3$ steps (cf. Chapter 5). The reservation cost and the maximum budget are given by the properties p^1 and p^2, respectively ($M = 2$). The $M \cdot L$ aggregation coefficients are

$$\begin{array}{lll} mac_1(r_1) = -1, & mac_1(r_2) = -1, & mac_1(r_3) = -1\,; \\ mac_2(r_1) = 1, & mac_2(r_2) = 0, & mac_2(r_3) = 0\,. \end{array}$$

Note, we arbitrarily associated the budget property with request r_1.

Objective Function

The objective function is composed of normalized properties $\|p^m\|_{p^m}$, aggregation coefficients $o_m(r_l)$ and $o_m(s_k)$, and the objectives' weight o^ω. We normalize a real-valued property p^m as follows

$$\|p^m_{r_l \triangleright s_k}(t,d,q)\|_{p^m} = \frac{p^m_{r_l \triangleright s_k}(t,d,q)}{\max(|\min(p^m)|, |\max(p^m)|)} \tag{10.2}$$

$$\text{with} \qquad \min(p^m) = \min_{\substack{r_l \in R,\, s_k \in S(r_l) \\ \langle t,d,q \rangle \in TDQ_{r_l \triangleright s_k}}} p^m_{r_l \triangleright s_k}(t,d,q)$$

$$\text{and} \qquad \max(p^m) = \max_{\substack{r_l \in R,\, s_k \in S(r_l) \\ \langle t,d,q \rangle \in TDQ_{r_l \triangleright s_k}}} p^m_{r_l \triangleright s_k}(t,d,q)\,.$$

The terms $o_m(r_l)$ and $o_m(s_k)$ denote the m-th aggregation coefficient of the request r_l and the resource s_k, respectively. Using the weighted sum as criteria B (cf. Def. 21), the objective function is defined by the following expression.

$$\begin{aligned}&\text{minimize}\\&\sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{o \in O_R} o^{\omega} \sum_{\substack{\langle t,d,q \rangle \in \\ TDQ_{r_l \triangleright s_k}}} x_{\langle r_l, s_k, t, d, q \rangle} \left(\sum_{m=1}^{M} o_m(r_l) \| p^m_{r_l \triangleright s_k}(t,d,q) \|_{p^m} \right)\\ +&\sum_{r_l \in R} \sum_{s_k \in S(r_l)} \sum_{o \in O_{s_k}} o^{\omega} \sum_{\substack{\langle t,d,q \rangle \in \\ TDQ_{r_l \triangleright s_k}}} x_{\langle r_l, s_k, t, d, q \rangle} \left(\sum_{m=1}^{M} o_m(s_k) \| p^m_{r_l \triangleright s_k}(t,d,q) \|_{p^m} \right)\end{aligned}$$

The terms O_R and O_{s_k} denote the sets of the objectives of the co-reservation request R and the resource s_k ($k = 1, \ldots, K$), respectively.

Example 10.10 (Min End Time & Max Fitness)
We illustrate the implementation of Example 5.5 with $L = 3$ steps (cf. Chapter 5). The end time is calculated as the sum of the start time (property p^1) and the duration (property p^2) of request r_3. The fitness (cf. Section 9.4.2) is given by property p^3. Because all entities – the co-reservation request and the resources – possess only a single objective all weights are set to 1. The properties are normalized as in Eq. (10.2). The eligible resources are the same as in Example 10.9 ($K = 7$). The $M \cdot (L + K)$ aggregation coefficients are

$M \cdot L$ coefficients of the co-reservation request

$o_1(r_1) = 0, \quad o_2(r_1) = 0, \quad o_3(r_1) = 0\,;$
$o_1(r_2) = 0, \quad o_2(r_2) = 0, \quad o_3(r_2) = 0\,;$
$o_1(r_3) = 1, \quad o_2(r_3) = 1, \quad o_3(r_3) = 0\,;$

$M \cdot K$ coefficients of resources

$o_1(s_1) = 0, \quad o_2(s_1) = 0, \quad o_3(s_1) = -1\,;$
$o_1(s_2) = 0, \quad o_2(s_2) = 0, \quad o_3(s_2) = -1\,;$
$o_1(s_3) = 0, \quad o_2(s_3) = 0, \quad o_3(s_3) = -1\,;$
$o_1(s_4) = 0, \quad o_2(s_4) = 0, \quad o_3(s_4) = -1\,;$
$o_1(s_5) = 0, \quad o_2(s_5) = 0, \quad o_3(s_5) = -1\,;$
$o_1(s_6) = 0, \quad o_2(s_6) = 0, \quad o_3(s_6) = -1\,;$
$o_1(s_7) = 0, \quad o_2(s_7) = 0, \quad o_3(s_7) = -1\,.$

Note, the coefficients of the resources are negative to invert the optimization sense (minimize → maximize).

10.5 Experimental Evaluation

Many optimization problems suffer from a large search space, which is common for resource management in Grid environments. We will assess the scalability of the models developed in Section 10.3 and 10.4. In particular, we run experiments with different combinations of several scenario parameters. The properties of time-qos-slots, the aggregation coefficients of the constraints as well as the objectives are determined by random number generators. Accordingly, each experiment is repeated several times.

Table 10.2: Parameters of the experimental evaluation of the integer model.

Parameter	Values
Application scenario	S1, S2, S3, S4, S5, S6, S7, S8
Number of requests	$1, 2, 3$
Number of resources	$1, 2, 3$
Number of constraints	$6, 9$
Number of runs	3

10.5.1 Evaluation of the Integer Model

We performed 432 experiment runs to study which instances are solvable in a reasonable time and which parameters influence the solving time most. In the experiments, we varied the number of requests, the number of resources, the number of the constraints per entity (requests and resources) and the structure of the co-reservation requests. Because we generated random numbers for properties, constraints and objectives, we repeated each experiment three times. The experiments' parameters are shown in Table 10.2. The application scenarios S1-S8 are depicted in Fig. 10.1.

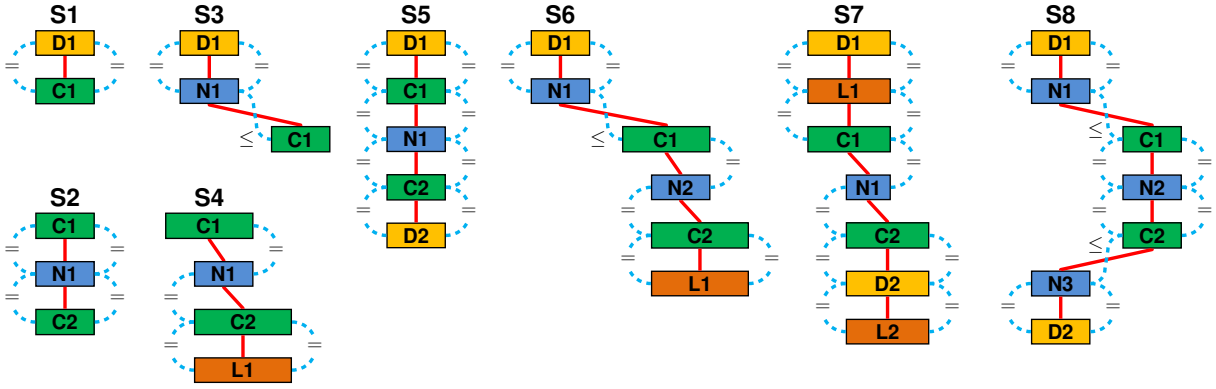

Figure 10.1: Application scenarios S1 to S8 used in the evaluation of the integer model. Solid red lines indicate spatial relationships. Temporal relationships are shown by dashed blue lines plus a comparison operator. The type of a requested resource is given by the first letter of the tag in a box – C for compute, D for data, N for network and L for license.

Several problem instances were solved in parallel by running CPLEX [ILO] processes – one per single experiment instance – on a SUN Galaxy 4600 16-core system with 64 GB of RAM. Each process used a single processor core only.

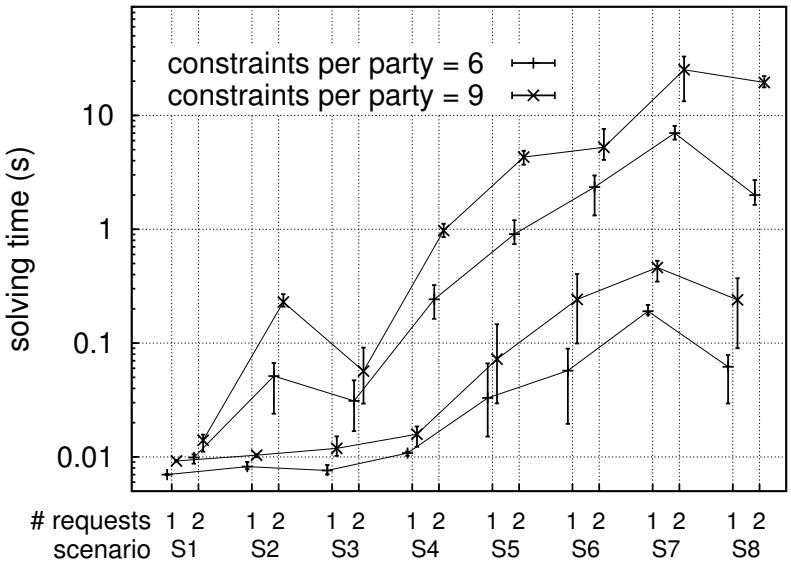

Figure 10.2: Solving time vs. the scenario and the number of requests for **one eligible resource** per request. The error bars show the average, min and max times.

Results

Figures 10.2, 10.3 and 10.4 show the times needed for scheduling requests of the application scenarios requiring two to six non-network resources and none to three network resources. The number of eligible non-network resources per request is one, two and three in Fig. 10.2, Fig. 10.3 and Fig. 10.4, respectively. The scenarios are ordered such that their complexity increases along the horizontal axis. For each scenario, four error bars are plotted, each one corresponding to a combination of the number of requests and the number of constraints.
Although, the experiments involved only a few number of requests and resources, we observed large solving times. The solving time increases exponentially, except for co-reservation requests containing only two parts, i.e., for scenario S1 (cf. Fig. 10.1). Studying the graphs in more detail, we find that the problem instances with a single eligible resource per atomic request (all instances in Fig. 10.2) are solvable in reasonable time.

The problem instances with two and three eligible resources per atomic request (cf. Fig. 10.3 and 10.4), require more time for finding an optimal solution. Most of the smaller instances (S1 to S4) and instances with a single request are solved in reasonable time, i.e., in less than 30 seconds. However, the slightly more complex scenarios require significantly more time. We also considered even more complex scenarios involving multiple steps of a job chain. These were solved efficiently in case of a single eligible resource per atomic request, but often required several hours solving time when the number of resources was increased.

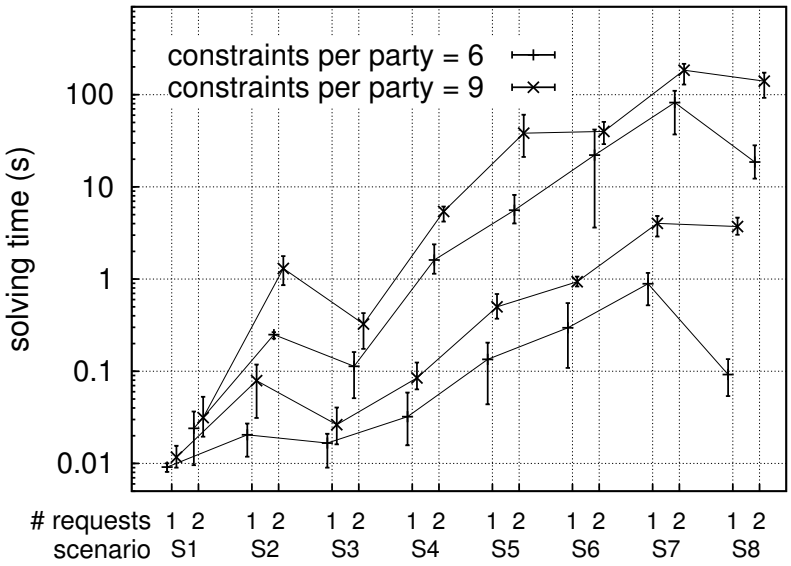

Figure 10.3: Solving time vs. the scenario and the number of requests for **two eligible resources** per request. The error bars show the average, min and max times.

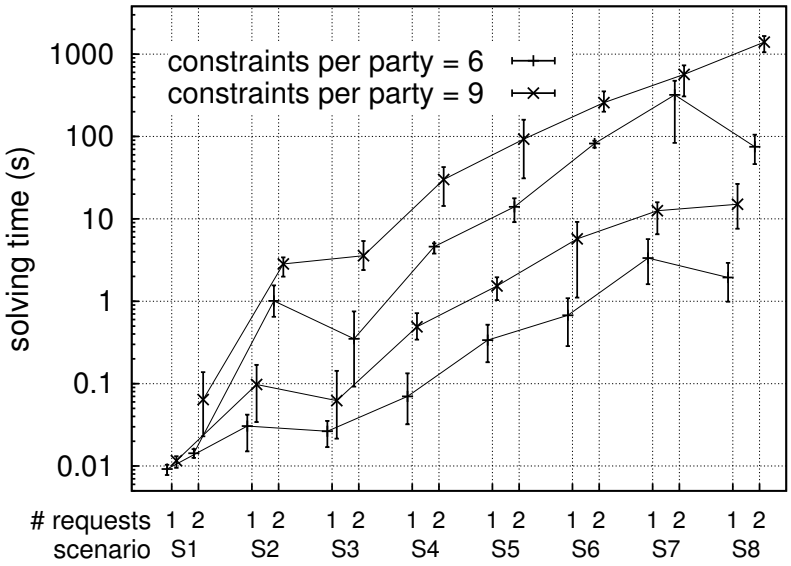

Figure 10.4: Solving time vs. the scenario and the number of requests for **three eligible resources** per request. The error bars show the average, min and max times.

Thus, we can devise the following recommendations for using an integer model to schedule co-reservations. First, for each atomic request only a single eligible resource should be considered. This requires a very efficient filtering in the matchmaking (cf. step ② in Fig. 6.2 on page 38). Second, the number of constraints should be kept small. Last, requests for complex applications requiring multiple resources should be scheduled individually.

10.5.2 Evaluation of the Binary Model

We used the following scenario for evaluating the binary model.

> *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 engine for two hours starting four hours after the reservation on the IBM p690 begins and a 100 Mbit/s-network link between the p690 and the visualization engine for the same time.*

The structure of the scenario is comparable to scenario S6 of Section 10.5.1 albeit S6 requires an additional non-network resource.

Table 10.3: Parameters of the experimental evaluation of the binary model.

Parameter	Values
Number of start times (corresponding time gap)	7 (11h), 12 (6h), 23 (3h), 34 (2h), 67 (1h), 133 (30m), 265 (15m), 397 (10m)
Number of resources	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20

We studied the impact of the number of eligible resources and the number of time-qos-slots on the time needed to find the optimal co-reservation candidate for a single co-reservation request. Table 10.3 shows the used parameters. The resulting number of binary variables is calculated as follows

$$nbv(S,T) = \left(3\,|S| + 2\,|S|^2\right) \cdot |T| \ ,$$

where S is the set of resources and T is the set of considered start times. The number of constraints is always 14 for the given scenario. In the absence of workload traces for co-reservations we randomly generated the time-qos-slots and their properties (cf. step ③ in Fig. 6.2 on page 38). For each parameter pair (number of resources, number of time-qos-slots), we executed 10 experiments and calculated the average time for finding the optimal co-reservation candidate. Several problem instances were solved in parallel by running CPLEX [ILO] processes – one per single experiment instance – on a SUN Galaxy 4600 16-core system with 64 GB of RAM. Each process used a single processor core only.

Figure 10.5 shows the solving time vs. the number of reservation candidates. Each curve represents the experiments with a specific number of resources.
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 in Fig. 6.2 on page 38) may only be valid for a certain time. Thereafter, the "best" co-reservation candidate is sub-optimal or reservation attempts

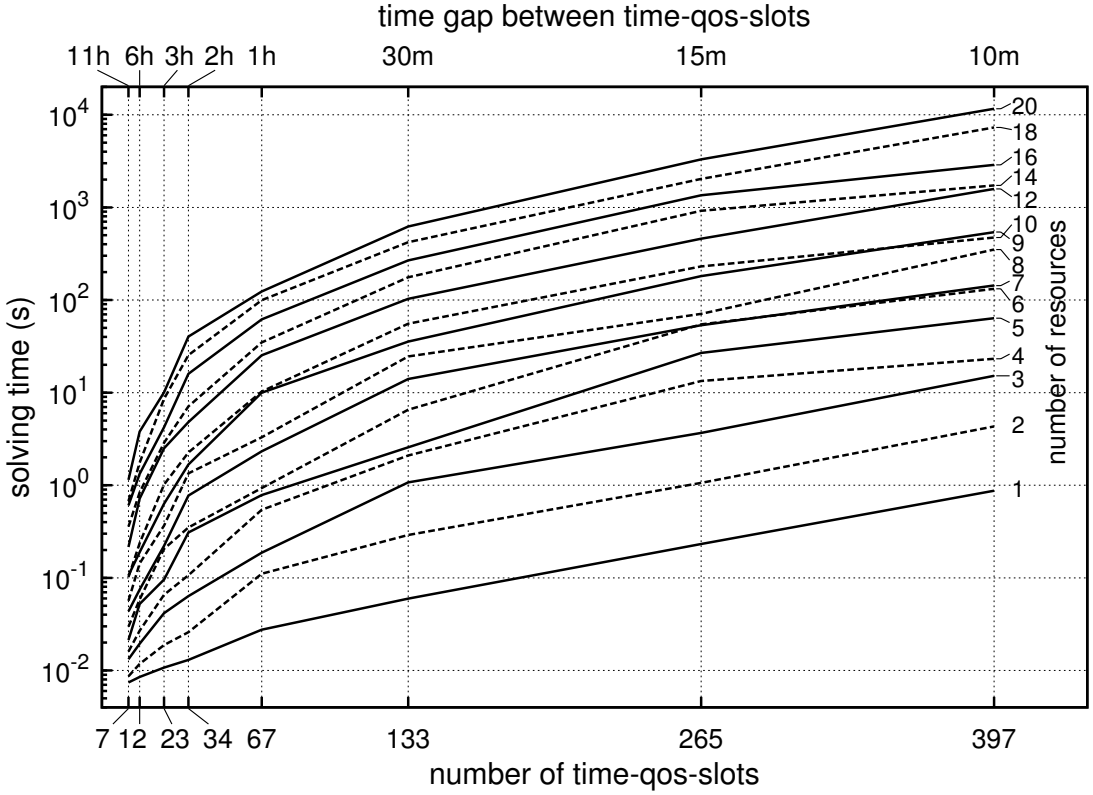

Figure 10.5: Solving time for several numbers of resources and time-qos-slots.

(cf. step ⑤ in Fig. 6.2 on page 38) simply fail. Third, the longer the book-ahead time (earliest start time) of the co-reservation, the longer solving times may be acceptable.

The experimental results provide two means for limiting the solving time – (1) the reservation system may ask the resource providers for a limited number of time-qos-slots and (2) use less eligible resources than found through the resource information service query (cf. step ② in Fig. 6.2 on page 38).

10.6 Refining IP and BP Models

We introduce several refinement strategies of the models developed in Sections 10.3 and 10.4. Refining a problem instance becomes necessary if

1. a failure occurs while trying to acquire reservations,
2. a failure occurs after the co-reservation was granted, *or*
3. a change request was issued after the co-reservation was granted.

Refining a problem instance means to adapt the optimization problem such that the new solution acknowledges the above cases. Handling the second and third case most

likely requires to perform the matchmaking (cf. step ② in Fig. 6.2 on page 38) and the probing (cf. step ③ in Fig. 6.2 on page 38) again. Thereafter, the actual refinement strategies are applied. In the first case, the matchmaking and probing steps are not required. In this section, we only introduce refinement strategies, because matchmaking and probing can be done by simply executing the mechanisms proposed in Chapter 8 and 9, respectively.

While a change request (third case) defines the desired refinement explicitly, the failure cases only provide hints on how the mapping should be changed. Therefore, we develop the following schemes for adapting a problem instance in case a failure occurred:

- removing the faulty resource completely,
- removing the faulty time-qos-slot only, *and*
- removing regions around faulty time-qos-slots.

All these schemes may be applied on all request parts requiring a refinement. Because we consider the refinements of any two request parts as independent, we only illustrate the refinement of a single request part in the subsequent sections.

Refinement Methodology. Let $CoRP = \{\langle R_i, S_i, T_i, D_i, Q_i, SIC_i, P_i, SAC_i, TR_i, SR_i, MAC_i, O_i\rangle\}$ denote the decomposed co-reservation problem. Initially, the set $CoRP$ contains a single element – a problem instance – only. Applying certain adaptation operations may lead to additional elements, i.e., single problem instances are split into multiple instances. All adaptations need to be carried out on all elements of $CoRP$. If parts of a co-reservation candidate should be kept, appropriate **equality** constraints are added to all problem instances. The new solution is found by solving all instances (elements of $CoRP$) individually and selecting the solution of the instance with the smallest objective value as global solution.

Excluding Faulty Resources

The simplest strategy is to exclude the resource incurring the reservation failure from the co-reservation problem. That is, the resource is treated as if it does not exist. Clearly, if the resource is the only *candidate* of a request, the problem becomes infeasible by excluding the resource. This can be easily avoided by letting the co-reservation allocator (cf. Chapter 11) verify the number of candidate resources and select the appropriate refinement strategy. In the following, we demonstrate how the integer model (cf. Section 10.3) and the binary model (cf. Section 10.4) are adapted.

Let $s_k \in S(r_l)$ denote the k-th resource to be excluded from the set of eligible resources $S(r_l)$ of request r_l.

Adapting the Integer Model. We exclude the resource s_k of request r_l by simply adding the constraint

$$x_{l,k} = 0 \tag{10.3}$$

to the solution integrity constraints SIC_i of each $CoRP_i \in CoRP$.

Adapting the Binary Model. We exclude the resource s_k of request r_l by adding the constraint

$$\sum_{\langle t,d,q\rangle \in TDQ_{r_l \triangleright s_k}} x_{\langle r_l,s_k,t,d,q\rangle} = 0$$

to the solution integrity constraints SIC_i of each $CoRP_i \in CoRP$.

Excluding Faulty Time-QoS-Slots

If the failure is limited to a single time-qos-slot $\langle t_f, d_f, q_f\rangle$ of the assignment $r_l \triangleright s_k$, the above strategy may be too restrictive. Thus, we propose a strategy which excludes the faulty time-qos-slot only.

Adapting the Integer Model. Depending on the relation of t_f, d_f and q_f to the bounds of their corresponding domains T, D, Q, we adapt a problem instance (1) by simply adjusting the boundary constraints or (2) by splitting the instances for each domain. We illustrate the adaptation for a single variable and domain, say t and T. Let T_{LO} and T_{UP} denote the lower and the upper bound of the domain T. If t_f equals the lower bound T_{LO}, we augment the solution integrity constraints SIC_i of each $CoRP_i \in CoRP$ with the constraint

$$t \geq (T_{LO} + 1)x_{l,k}\,.$$

Similarly, if t_f equals the upper bound T_{UP}, we augment the solution integrity constraints SIC_i of each $CoRP_i \in CoRP$ with the constraint

$$t \leq (T_{UP} - 1)x_{l,k}\,.$$

If t_f neither equals T_{LO} nor T_{UP}, we replace each $CoRP_i$ with two instances, $CoRP_i^{LO}$ and $CoRP_i^{UP}$. We create the instance $CoRP_i^{LO}$ by adding the constraint

$$t \leq (t_f - 1)x_{l,k}$$

to the solution integrity constraints SIC_i. Similarly, we create the instance $CoRP_i^{UP}$ by adding the constraint

$$t \geq (t_f + 1)x_{l,k}$$

to the solution integrity constraints SIC_i.

Adapting the Binary Model. We adapt each instance $CoRP_i$ by simply adding the constraint

$$x_{\langle r_l,s_k,t_f,q_f,d_f\rangle} = 0 \tag{10.4}$$

to the solution integrity constraints SIC_i.

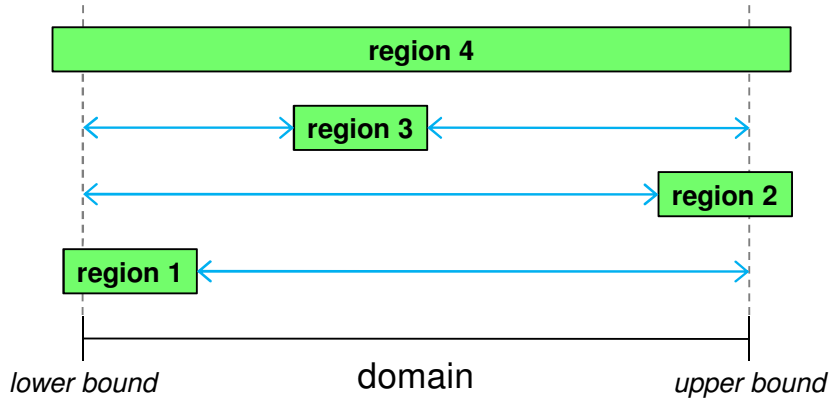

Figure 10.6: Relations of regions to be excluded from a domain.

Excluding Regions around a Faulty Time-QoS-Slot

Excluding just a single faulty time-qos-slot may lead to a solution which is very close to the failed one. Therefore, it may be more appropriate to exclude regions around a faulty time-qos-slot from the problem instance.

Given a faulty time-qos-slot $\langle t_f, q_f, d_f \rangle$ of the assignment $r_l \triangleright s_k$ and region diameters $t_a \in \mathbb{N}_+$, $d_a \in \mathbb{N}_+$ and $q_a \in \mathbb{N}_+$ we propose a strategy which excludes the region $[t_f - t_a, t_f + t_a] \times [d_f - d_a, d_f + d_a] \times [q_f - q_a, q_f + q_a]$ from the search space. Figure 10.6 shows the four cases of the relation of a single-dimensional region (green boxes) to the corresponding domain.

Adapting the Integer Model. Depending on the relation of the regions $[t_f - t_a, t_f + t_a]$, $[d_f - d_a, d_f + d_a]$ and $[q_f - q_a, q_f + q_a]$ to the bounds of their corresponding domains T, D, Q (cf. Fig. 10.6), we adapt a problem instance (1) by simply adjusting the boundary constraints (regions 1 and 2), (2) by splitting the instances for each domain (region 3) or (3) by excluding the resource s_k from the problem instance (region 4). We illustrate the adaptation for a single variable and domain, say t and T. Let T_{LO} and T_{UP} denote the lower and the upper bounds of the domain T. If the region $[t_f - t_a, t_f + t_a]$ overlaps with the lower bound (cf. region 1 in Fig. 10.6), we augment the solution integrity constraints SIC_i of each $CoRP_i$ with the constraint

$$t \geq (t_f + t_a + 1)x_{l,k}\,.$$

Similarly, if the region overlaps with the domain's upper bound (region 2 in Fig. 10.6), we augment the solution integrity constraints SIC_i of each $CoRP_i$ with the constraint

$$t \leq (t_f - t_a - 1)x_{l,k}\,.$$

If the region is fully contained in the domain (cf. region 3 in Fig. 10.6), we split each $CoRP_i$ into two instances, say $CoRP_i^{LO}$ and $CoRP_i^{UP}$. We create $CoRP_i^{LO}$ by adding the constraint

$$t \leq (t_f - t_a - 1)x_{l,k}$$

to the solution integrity constraints SIC_i. Similarly, we create $CoRP_i^{UP}$ by adding the constraint

$$t \geq (t_f + t_a + 1)x_{l,k}$$

to the solution integrity constraints SIC_i.

If the region is a superset of the domain (cf. region 4 in Fig. 10.6), all time-qos-slots of the assignment $r_l \triangleright s_k$ must be excluded. This is implemented by removing the resource s_k from any instance $CoRP_i$ (cf. Section *Excluding Faulty Resources*) as in Eq. (10.3).

Adapting the Binary Model. Let $IS_{l,k}(t_f, t_a, d_f, d_a, q_f, q_a)$ denote the intersection of the region $[t_f \pm t_a] \times [d_f \pm d_a] \times [q_f \pm q_a]$ and the set $TDQ_{l,k}$, i.e.,

$$IS_{l,k}(t_f, t_a, d_f, d_a, q_f, q_a) := \left\{ \langle t, d, q \rangle \middle| \langle t, d, q \rangle \in TDQ_{l,k} \ \wedge \ \langle t, d, q \rangle \notin [t_f \pm t_a] \times [d_f \pm d_a] \times [q_f \pm q_a] \right\}.$$

We exclude the region of time-qos-slots by simply adding the constraint

$$\sum_{\substack{\langle t,d,q \rangle \in \\ IS_{l,k}(t_f,t_a,d_f,d_a,q_f,q_a)}} x_{\langle r_l, s_k, t, d, q \rangle} \quad = \quad 0 \tag{10.5}$$

to the solution integrity constraints SIC_i of each $CoRP_i \in CoRP$.

10.7 Summary

We presented two approaches for modeling the mapping of requests to co-reservation candidates, studied their scalability and discussed refinement strategies for determining alternative mappings.

Both approaches satisfy the requirements **R1** (Fairness) and **R4** (Refinements) very well. Concerning requirement **R2** (Constraints and Objectives), the integer model limits the base functions to linear combinations. While the binary model is not limited to linear functions of the properties itself, it is if aggregations of properties are considered. The less restricted modeling of properties comes at the cost of a lower solution accuracy, i.e., each time-qos-slot is modeled by a single binary variable. Both approaches are only suitable to small problem instances (requirement **R4** Efficiency). This issue can be leveraged by reducing the size of the search space. In the integer model, the only means is to limit the number of resources. In the binary model, however, it is possible to trade-off solution accuracy versus solution time by reducing the number of resources, the considered start times, durations and service levels.

Chapter 11

Allocating Resources to a Co-Reservation Candidate

The result of the mechanisms proposed in Chapter 10 is a *co-reservation candidate* CA^S (cf. Def. 7), i.e., a mapping of request parts to resources, start times, durations and service levels. We call such a mapping a candidate, because it is not reserved (or allocated) yet. A mechanism for allocating resources to a co-reservation candidate sends reserve messages – one for each request part – to the resources specified in the candidate. The design of such a mechanism is mainly concerned with the following issues:

I1 – *What information is passed with the reserve messages?*,

I2 – *How are the messages sent – sequentially or in parallel?*,

I3 – *Do successful reserve messages need a confirmation?*, *and*

I4 – *How are unsuccessful reserve message handled?*

These issues may not be solved individually. For example, consider a scenario where reservations do not need a confirmation, but may only be canceled by paying a (small) fee. Then, it may be worthwhile to send the reserve messages one after another and only send the next if the last one succeeded. Thus, the potential penalty fee for canceling already granted reservations may be minimized.

The strategy for handling failures may depend on the overall optimization goal. That is, if reserve messages are denied, a mechanism may follow two main schemes: (1) cancel already granted parts, refine the co-reservation problem (cf. Section 10.6) and try to find a new co-reservation candidate and (2) retain already granted parts, but search for alternatives of the failed parts. While the first method will always derive the global optimum, it may also require more cancel operations. A consumer wants to minimize the number of cancelations, because they may incur additional costs. Also, a provider must deny or postpone requests requiring the capacity already granted to another reservation. Hence, it could be better for both the requester and the resource providers if the second method is used. The second method, however, may not find the global optimum.

We present several mechanisms for allocating resources to a co-reservation candidate. Each mechanism works under different assumptions and exposes different characteristics to the requester of a co-reservation and/or to the resources being reserved. All mechanisms, however, use the same method to send reserve messages.

Information of a reserve Message. Let CA^S be a combination of assignments defining a co-reservation candidate. For any tuple $\langle r_l, s_k, t, d, q\rangle \in CA^S$ and the properties $P_{r_l \triangleright s_k}$, a reserve message contains the following information:

- the start time t of the reservation,
- the duration d of the reservation,
- the service level q of the reservation, *and*
- a list of pairs (id, val) for all properties $p^{id} \in P_{r_l \triangleright s_k}$ and $val = p^{id}_{r_l \triangleright s_k}(t, d, q)$.

Outline. We discuss requirements on the reservation mechanisms in Section 11.1. Thereafter, we describe the state of the art in Section 11.2. The mechanisms are presented in Sections 11.3 and 11.4. We close the chapter with a summary in Section 11.5.

11.1 Requirements

All-or-nothing semantics. The requester is interested in a co-reservation of all parts or none. That is, a mechanism must provide some relaxed form of the well-known transaction property *atomicity* [WV02].[1]

Non-reservable request parts. As described in Section 7.1.1, it may be desirable to specify auxiliary parts in a co-reservation request. These parts are used to define the utility function of the mapping, but are not to be reserved. A reservation mechanism must detect these parts and omit them from acquiring reservations.

Balancing the objectives of the consumers and the providers. Naturally, the parties involved in acquiring a co-reservation have got different and often contradicting objectives. For example, consumers want a high total reservation success rate, while providers want to minimize the impact of an allocation mechanism on the utilization of their resources. A mechanism for acquiring resources to candidates must be able to balance the objectives of all parties.

[1] The other transaction properties are of minor interest (*isolation*), inherently implemented by the resources' local management (*durability*) and achieved through the serialization of reserve messages at the local reservation services (*consistency*).

Handling of concurrent requests. Because multiple reservation services may compete for the same resources or even the same reservation candidates, an allocation mechanism must resolve deadlocks and livelocks.

11.2 State of the Art

Despite the large amount of work proposing Grid resource management systems, the requirements listed above are mostly ignored (the second and the third requirement). We begin our survey on methods for ensuring *all-or-nothing* semantics in distributed database systems and Web services management. Thereafter, we present approaches to co-allocation in Grid resource management.

Transactions in Distributed DBMSs. In [TGGL82], Traiger et al. propose four properties any transaction mechanism should provide in order to ease the use of a distributed database system. These properties are *location* transparency, *replication* transparency, *concurrency* transparency and *failure* transparency. Location and replication transparency are of no concern to CORES since reservation candidates always explicitly name the resource, time frame and service level. Concurrency transparency hides the effects of concurrent transactions, particularly it provides a consistent view of the data. Albeit multiple co-reservation requests may compete for the same reservation candidates, concurrency control is provided through the serialization of reserve messages at each resource. Traditional transactions in DBMSs and allocations of co-reservations have most in common with respect to *failure* transparency. In [TGGL82], four classes of failures – (1) application detected, (2) local node crashes, (3) communication network failures, and (4) failures at remote nodes – are discussed. Assuming that all CORES components store their state persistently, a co-reservation mechanism must pay special attention to application detected failures (1) and communication network failures (3). The former corresponds to denied reserve messages and may eventually lead to aborting the whole co-reservation. Communication network failures may cause a situation, where a resource grants an allocation request but the response is never received by the Grid Reservation Service. Such failures may be efficiently detected by means of timeouts, that is, to let resources grant reservations *preliminarily* only and require a confirmation within a certain period of time. The confirmation of multiple preliminary reservations requires an appropriate commit protocol. The often used two-phase-commit (2PC) protocol [Gra78] is not appropriate because a participant may not abort a transaction once it has sent a prepared messages (corresponding to granting a reservation). More advanced transaction management schemes were introduced to cope with long-lived or long-running transactions. While reserving multiple resources does not necessarily require a long time, some lessons may be learned from *sagas* [GMS87] and flexible transactions [ELLR90]. In [GMS87], Garcia-Molina and Salem proposed to split a transaction into multiple sub-transactions. Sub-transactions may be committed before their parent transaction commits. If a *saga* fails, two forms of

recovery schemes – *backward* or *forward* – may be executed. Backward recovery requires compensation transactions which "semantically" undo the effects of already committed sub-transactions. Forward recovery requires checkpoints from which all missing sub-transactions may be performed. At first sight, *sagas* seem to match the requirements of CORES very well – sub-transactions are similar to reservations of individual resources and compensating is implemented by simply canceling a reservation. However, the concept of *sagas*: (1) is too strict with respect to failure recovery, (2) does not allow "semantically" equivalent sub-transactions substitute for failed ones and (3) requires all sub-transactions to be given beforehand. In contrast, an allocator of a co-reservation may consider alternative candidates if one or multiple original candidates could not be allocated. In [ELLR90], Elmagarmid et al. propose *flexible* and *mixed* transactions both possibly spanning multiple *autonomous* data bases. Flexible transactions support the concept of alternative sub-transactions, while mixed allow compensable and un-compensable sub-transactions in a global transaction. They also introduce the concept of time, that is, support for restrictions when sub-transactions may be executed and when the processing must be finished (deadline). The global transaction is composed statically. In particular, the alternatives and their order must be given before the actual execution begins. In CORES, reserve messages may always be issued in a dynamically determined order, but their success probability may change with time.

Transactions in Web Service Environments. The most advanced protocols/mechanisms to manage transactions in Web services are the Business Transaction Protocol (BTP) [FHC04], the Web Services Business Process Execution Language (WS-BPEL) [AAA+07] and the WS-Coordination protocol family (WS-CO) [FJ07, LW07, FL07]. They propose mechanisms for efficiently processing long-running transactions, resembling flexible transactions as proposed by Elmagarmid et al. [ELLR90]. There are, however, small differences between them. For example, Sauter and Melzer [SM05] found that BPEL4WS LRT (the predecessor of WS-BPEL) essentially provides the same functionality as WS-BusinessActivity (WS-BA, part of WS-CO), but lacks support for remote activities. More importantly, BPEL4WS LRT only supports static sets of participants while WS-BA allows to adapt the set of participants dynamically. Little and Feingold [LF03a] compared WS-Tx (the predecessor to the WS-Coordination family) and BTP. They found that WS-CO provides a better separation of business and transaction logic. Since, WS-CO builds upon a large family of Web Service standards it only needs to define the transaction management protocol. In contrast, BTP needs to define many aspects which are not related to transaction management. Furthermore, they argue that WS-CO may better leverage transaction management in existing back-ends than BTP does.

The OASIS' *Web Services (WS) Transactions* protocol suite defines the framework WS-Coordination [FJ07] for coordinating the use of multiple web services as well as the standards WS-AtomicTransaction [LW07] for short-lived activities and WS-BusinessActivity [FL07] for long-running activities. WS-AtomicTransaction implements the well-known two-phase commit protocol. WS-BusinessActivity (WS-BA) supports nes-

ted scopes and relaxed failure handling. That is, the parent scope may continue the transaction if a child fails. WS-BA introduces the concept of "tentative" operations, which require a confirmation, and supports the compensation of completed activities. Ouyang et al. [OSM01] propose an optimistic commit protocol for conversational transactions. Conversational transactions provide *all-or-nothing* semantics to a set of independent component transactions. In their model, component transactions – more precisely, the attached business logic – may only be undone until some deadline. They argue that neither two-phase commit [Gra78], the Transaction Internet Protocol [EKL98] nor the optimistic commit protocol proposed by Levy et al. [LKS91] are well suited for such conversational transactions. Therefore, Ouyang et al. propose the concept of *transactions in update*. Before a component transaction passes its deadline, the corresponding e-service repeats the business logic and sends an updated response to its parent e-service.

Zhao et al. [ZMMS05] propose to first gather reservations for business tasks and show how reservations may be integrated in protocols such as WS-Tx and WS-Coordination [FJ07]. The business logic is only executed after all needed reservation are acquired. Thus, other transactions do not see intermediate results as in *sagas* [GMS87], InterBase [ELLR90], the Web Services transaction protocols (BTP [FHC04], WS-BusinessActivity [FL07] and WS-BPEL 2.0 [AAA+07]). Additionally, compensation handlers are only needed to undo reservations.

In [CBC06], Choudry et al. argue that protection from potentially unsuccessful transactions through timeouts may increase the failure rate the longer a transaction lives. They propose that a client pays some fee to held up expired bookings to let the transaction finish successfully. Bookings may be weighted to acknowledge the different importance of different services. As in flexible transactions [ELLR90], the transaction coordinator may consider alternative providers to tolerate denied bookings.

Co-Allocation in Grid Environments. While several systems were developed for coallocating multiple resources (cf. Sec. 10.2), most of them ignore the autonomous nature of Grid resources. That is, these systems do not handle reservation failures appropriately if at all. Therefore, we concentrate on research which emphasizes that particular characteristic of Grid resources.

Snell et al. [SCJG00] consider several issues arising in meta-scheduling multiple advance reservations spanning multiple administrative domains. Depending on the number of meta-schedulers, different behaviors are observed. The information about the available time slots may be outdated. Thus, allocations may fail. If multiple metaschedulers compete for scarce resources, deadlock and livelock situations must be handled. They propose to alleviate deadlocks by eliminating the hold and wait condition [SGG04]. For livelock situations, they propose two different approaches. In the first (non-deterministic) approach, all acquired allocations are released and the metascheduler waits some random time before starting a new trial. The second method uses preliminary reservations. The preliminary reservations need to be confirmed before a timeout expires.

GridARS [TNK+08] is an advance reservation-based framework for co-allocating compute and network resources. Its co-allocation mechanism employs a basic two-phase commit protocol, allowing the providers to abort after a timeout for confirming preliminary reservations has expired.
In [KM05], Kuo and Mckeown present a protocol for advance reservation and co-allocation of resources in a Grid environment. Their protocol covers the whole life of an application run – from the reservation of the resources until the execution of the application completes. The basic two-phase commit protocol is made non-blocking by allowing any party to abort the transaction at any time. Kuo and Mckeown, however, do not propose schemes for the efficient co-allocation.

HARC [Mac07] proposes a commit protocol based on Paxos Consensus [Lam01] to improve the fault tolerance of the transaction coordinator. The transaction coordinator is replaced by a set of $2F + 1$ acceptors. If no more than F acceptors fail, the protocol guarantees that both the users and the resources see the same outcome of the co-allocation process. Like all the other co-allocation protocols, HARC focuses on the transactional behavior of a co-allocation but on its efficiency.

Summary. We discussed several models addressing the problem of ensuring all-or-nothing semantics of transactions in data bases [ELLR90, LKS91] and web services management [AAA+07, FJ07, FL07]. In Grid environments, mechanisms for advance reservation and co-allocation typically employ a two-phase commit protocol with timeouts [TNK+08], allowing cancelations any time [KM05] or focus on improving the fault tolerance of the coordinator [Mac07]. All of the presented prior art, however, does not propose mechanisms for the efficient allocation of reservations. Therefore, we develop schemes which build upon transaction models, but specifically address the issues **I1**–**I4** raised at the beginning of this chapter.

11.3 Sequentially Allocating Resources

The general procedure for sequentially allocating resources is shown in Alg. 2. First, the allocation order is determined (line 1). In each iteration of the main loop (lines 2 to 15), the procedure sends a reserve message for the first remaining request part (line 3). If the reservation is granted, the reservation counter is increased (line 5) and the algorithm continues with the next iteration. In case of a failure, the procedure tries to find alternative reservation candidates for a subset of all request parts (line 7). If an alternative co-reservation candidate was found, the changed reservations are canceled (line 9) and the reservation counter is decreased accordingly (line 10). If no alternative was found, all previously obtained reservations are canceled (line 12) and the algorithm returns with a failure (line 13). Before starting a new iteration, the algorithm verifies if any previously granted preliminary reservation has expired, decreases the reservation counter accordingly (line 14) and recalculates the allocation order (line 15). If all parts have been granted, preliminary reservations are confirmed (line 16). Finally,

Algorithm 2: General procedure for sequentially allocating resources.

```
1  calculate allocation order for candidate CA^S
2  while ¬ all parts have been successfully reserved do
3      send reserve message for 1^st remaining request part
4      if reservation successful then
5          increase reservation counter
6      else
7          search for alternative co-reservation candidate
8          if search successful then
9              cancel changed reservations
10             decrease reservation counter accordingly
11         else
12             cancel previously obtained reservations
13             return FAILURE
14         decrease counter for expired preliminary reservations
15         recalculate allocation order for the remaining parts of CA^S
16 confirm preliminary reservations
17 return SUCCESS
```

the algorithm returns success (line 17). The algorithm contains two main building blocks:

1. the calculation of the allocation order (lines 1 and 15), *and*
2. the search for alternative co-reservation candidates (line 7).

Next, we describe the calculation of the allocation order in detail (cf. Section 11.3.1). Thereafter, we present schemes for searching alternative co-reservation candidates (cf. Section 11.3.2).

11.3.1 Calculating the Allocation Order

The participants in a co-reservation – the users and the resource providers – are concerned with different performance metrics of an allocation order. Users are interested in a high total reservation success rate and a low cancelation fee. Providers want to minimize the impact on the utilization of the resources. Note, these goals may be the same or similar to the objectives used for finding a mapping of requests to co-reservation candidates (cf. Chapter 10).

Let I_{CA^S} denote the set of the numbers of the request parts in CA^S, i.e.,

$$I_{CA^S} = \{i \mid \exists \langle r_l, s_k, t, d, q \rangle \in CA^S \quad \text{with} \quad i = l\} \,. \tag{11.1}$$

We define an allocation order as the function

$$\psi : \ I_{CA^S} \longrightarrow [1, |I_{CA^S}|] \, .$$

That is, the allocation step of part i is denoted by the expression $\psi(i)$. Accordingly, the expression $\psi^{-1}(j)$ denotes the request part, which is processed at the j-th allocation step. Let n denote the number of request parts to be reserved, i.e., $n = |CA^S| = |I_{CA^S}|$. The set of all allocation orders defined over I_{CA^S} is denoted by AO_n.
Next, we formally define the main performance metrics. Thereafter, we present several ordering schemes and discuss their ability to satisfy the goals of the stakeholders.

Performance Metrics

The terms p_i^{cost}, p_i^{pen}, p_i^{esr} and p_i^{fit} denote the i-th request part's reservation cost, reservation cancelation penalty, estimated reservation success rate and reservation fitness, respectively. We denote the success probability of allocating resources to the i-th request part ($i \in I_{CA^S}$) during the j-th allocation step by the term $\mathsf{p}(i, j)$. The expression $\overline{\mathsf{p}}(i, j)$ denotes the probability that the allocation of resources to the i-th request part fails during the j-th step, i.e., $\overline{\mathsf{p}}(i, j) = 1 - \mathsf{p}(i, j)$. Note, we assume that allocations of resources to different request parts are independent of each other. Thus, their success probability is independent as well.

Total Reservation Success Rate. For a given allocation order ψ, we define the success probability $\mathsf{P}_{\mathsf{sq}}^{\psi}(j)$ of the sequence of the first j allocation steps as

$$\mathsf{P}_{\mathsf{sq}}^{\psi}(j) = \prod_{i=1}^{j} \mathsf{p}\left(\psi^{-1}\left(i\right), i\right) \, . \tag{11.2}$$

The probability that the sequence from allocation step u to step v for a given order ψ fails is defined as

$$\overline{\mathsf{P}}_{\mathsf{sq}}^{\psi}(\mathsf{u}, \mathsf{v}) = \prod_{i=u}^{v} \overline{\mathsf{p}}\left(\psi^{-1}(i), i\right) \, .$$

The **average** total reservation success rate $\mathsf{P}_{\varnothing}(n)$ is defined by Eq. (11.3).

$$\mathsf{P}_{\varnothing}(n) = \frac{1}{|AO_n|} \sum_{\psi \in AO_n} \mathsf{P}_{\mathsf{sq}}^{\psi}(n) \tag{11.3}$$

Utilization. Because reservations block resources requested by other jobs, providers want to minimize both the number of cancelations and the time between the approval of a reserve message and the reception of a cancel message. For the sake of simplicity, we assume that the failure of a single allocation step j results in canceling the previously granted reservations of the steps 1 to $j - 1$. Furthermore, we assume that the loss function $\mathsf{ls}(i, j, k)$ specifies the costs incurred by a reservation of part i granted at

step j and canceled at step k. We define the **partial** impact $\mathsf{U}_{\mathsf{p}}^{\psi}(i)$ on the utilization of the resource provider addressed by part i as

$$\mathsf{U}_{\mathsf{p}}^{\psi}(i) = \begin{cases} \sum_{j=\psi(i)}^{n-1} \mathsf{P}_{\mathsf{sq}}^{\psi}(j)\ \overline{\mathsf{p}}\left(\psi^{-1}(j+1), j+1\right)\ \mathsf{ls}\left(i, \psi(i), j+1\right) & \text{if } \psi(i) < n\,, \\ 0 & \text{otherwise}\,. \end{cases} \tag{11.4}$$

If the request part i is processed during the first $n-1$ steps ($\psi(i) < n$), the order impacts the utilization. Each summand is composed of three factors: (1) the probability $\mathsf{P}_{\mathsf{sq}}^{\psi}(j)$ that the first j allocation steps succeed, (2) the probability $\overline{\mathsf{p}}(\psi^{-1}(j+1), j+1)$ that the next allocation step $j+1$ fails, and (3) the cost $\mathsf{ls}(i, \psi(i), j+1)$ incurred by that failure.

Finally, the **average** total impact on the utilization of the resource provider addressed by request part i is defined as

$$\mathsf{U}_{\varnothing}(i) = \frac{1}{|AO_n|} \sum_{\psi \in AO_n} \mathsf{U}_{\mathsf{p}}^{\psi}(i)\,. \tag{11.5}$$

Cancelation Fees. Cancelations of already granted reservations may induce some penalty, i.e., a cancelation fee. Naturally, a user is interested in minimizing the total cancelation fees.

Let p_i^{pen} denote the fee for canceling the reservation of request part i. For a given allocation order ψ and an allocation step k, we define the **partial** cancelation fee $\mathsf{F}_{\mathsf{P}}^{\psi}(k)$ of step k as

$$\mathsf{F}_{\mathsf{p}}^{\psi}(k) = \left(\prod_{j=1}^{k-1} \mathsf{p}\left(\psi^{-1}(j), j\right)\right) \cdot \left(\sum_{j=1}^{k-1} p_{\psi^{-1}(j)}^{pen}\right) \cdot \overline{\mathsf{p}}\left(\psi^{-1}(k), k\right)\,.$$

The three factors are (1) the probability that the k-th allocation step is reached, (2) the aggregated cancelation fee of the steps $1, \ldots, k-1$, and (3) the probability that the k-th allocation step fails. The **expected** cancelation fee of a single allocation order ψ is defined as

$$\mathsf{F}_{\mathsf{E}}^{\psi} = \sum_{k=2}^{n} \mathsf{F}_{\mathsf{P}}^{\psi}(k)\,. \tag{11.6}$$

Finally, the **average** total cancelation fee considering all possible allocation orders is defined as

$$\mathsf{F}_{\varnothing} = \frac{1}{|AO_n|} \sum_{\psi \in AO_n} \mathsf{F}_{\mathsf{E}}^{\psi}\,. \tag{11.7}$$

Ordering Schemes

The overall goal of the ordering schemes is to achieve a high satisfaction wrt. the performance metrics presented above. While it is possible to calculate a global optimum, this may simply take too much time. Hence, the efficient calculation of the allocation orders requires heuristics which satisfy the goals to some degree. We present the following ordering schemes

- random order,
- smallest success probability first order,
- earliest start time first order,
- smallest cancelation fee first order, *and*
- longest confirmation time first order.

Random Order. The allocation order ψ is randomly generated. Thus, the average total reservation success rate, the average total impact on the utilization of a resource and the average total cancelation fee are calculated as in Equations (11.3), (11.5) and (11.7), respectively.

Smallest Success Probability First Order. Clearly, the earlier failing candidates are identified,

- the earlier alternatives may be looked up,
- the fewer granted reservations need to be canceled, *and*
- the smaller will the cancelation fee be.

Ordering the n request parts by their increasing estimated success probability is a means to achieve these properties. The metrics total reservation success rate, partial impact on the utilization of the resources and the expected cancelation fee of a given order ψ are calculated with the formulas given in Equations (11.2), (11.4) and (11.6), respectively.

Earliest Start Time First Order. A co-reservation candidate specifies a start time p_i^{stt} for each request part i. The start times may differ if the application requesting the co-reservation is a job chain. In that case, it may be needed to acquire the reservations in the order of increasing start times. This is particularly important for parts with small advance booking times, i.e., the difference of the start time and the current time. An allocation order with increasing start times can easily be determined.

The properties total reservation success rate, partial impact on the utilization of the resources and the expected cancelation fee of a given order ψ are calculated with the formulate given in Equations (11.2), (11.4) and (11.6), respectively.

Smallest Cancelation Fee First Order. In order to minimize the total cancelation fee, the request parts may be sorted from small to high cancelation fees. Because the estimated success rate p_i^{esr} of a request part may not correlate with the part's cancelation fee, the proposed ordering could contradict the expectation of minimizing the total cancelation fee.

Example 11.1 (Smallest Cancelation Fee First Order)
Consider two cases:

(A) for each request part i the cancelation fee p_i^{pen} equals the estimated success rate at step one, i.e., $\mathsf{p}(i,1)$, and

(B) for each request part i the cancelation fee p_i^{pen} equals the estimated success rate of part $n+1-i$ at step one, i.e., $\mathsf{p}(n+1-i,1)$.

In both cases, we define the estimated success rate as

$$\mathsf{p}(i,j) = (1-(j-1)\alpha_i)\, p_i^{esr}\,.$$

Let the co-reservation contain three parts with the initial estimated success rates $p_1^{esr} = 0.85$, $p_2^{esr} = 0.90$, $p_3^{esr} = 0.95$ and $\alpha_{1/2/3} = 0.01$.

Applying the random order*, the average total cancelation fees $\mathsf{F}_\varnothing$ (cf. Eq. (11.7)) are 0.264 and 0.259 for the cases (A) and (B), respectively. With the scheme* smallest cancelation fee first*, the expected cancelation fees F_E (cf. Eq. (11.6)) are 0.170 and 0.335 for the cases (A) and (B), respectively. In fact, the scheme* smallest success probability first *achieves the lowest cancelation fees with 0.170 and 0.185 for the cases (A) and (B), respectively.*

Longest Confirmation Time First Order. The approach of charging a penalty for canceling a reservation may be enhanced by the following processing steps

1. a provider acknowledges a reserve message with a **preliminary** reservation,
2. the broker must confirm this reservation until some timeout (set by the provider) to secure the reservation.

A client may cancel an unconfirmed preliminary reservation without any charge. If the provider does not receive a confirmation until the timeout expires, the preliminary reservation is automatically canceled. In contrast, canceling a confirmed reservation incurs a penalty.

The value of the timeout may correspond to the competition for resources. Since resources blocked by a preliminary reservation must not be allocated to other requests, a provider may adapt the timeout to its current load. For example, if a provider receives many requests, it may ask for a short timeout to minimize the number of denied requests.

Assuming that the time t_{acq} needed to acquire a preliminary reservation is the same for all request parts and all steps, the order ψ for acquiring the reservations must satisfy the following condition

$$\forall i: \quad i < n \implies p^{tmo}_{\psi^{-1}(i)} \geq t_{acq}(n-i)\,, \tag{11.8}$$

where $p^{tmo}_{\psi^{-1}(i)}$ denotes the timeout associated with the request part allocated at step i.

The scheme *longest confirmation time first* generates an allocation order which *may* satisfy Eq. (11.8). In certain cases, however, no allocation order may satisfy the condition. This is the case, if two (or more) preliminary reservations require a confirmation before the timeout value t_{acq}.

Given an allocation order ψ, which satisfies Eq. (11.8), the total reservation success rate $\mathsf{P}_{\mathsf{sq}}^{\psi}$ and the partial impact on the utilization of the resources $\mathsf{U}_{\mathsf{P}}^{\psi}$ are calculated by Equations (11.2) and (11.4), respectively. The expected cancelation fee $\mathsf{F}_{\mathsf{E}}^{\psi}$ is zero.

Hybrid Schemes. While the above schemes demonstrate that it is easily possible to optimize a single performance metric, a real scenario may require to optimize multiple metrics. In that case, an appropriate means such as a utility function aggregating multiple metrics or Pareto-optimality may be used. Hybrid schemes may also integrate the sequential and the concurrent processing of request parts.

11.3.2 Alternative Co-Reservation Candidates

If the allocation of resources to a co-reservation candidate fails, the reservation procedure derives alternative candidates of a **subset** of all request parts (line 7 of Alg. 2). The benefit of searching for alternatives is, that some characteristics of the original co-reservation candidate may be kept. In particular, already granted reservations may be kept, which can be important if the competition is high for those request parts. Furthermore, retaining as many granted reservations as possible supports limiting the cancelation fees.

In this section, we present schemes for deriving subsets of all request parts for which alternative candidates are determined. The actual mechanisms for determining such alternatives were discussed in Chapter 10.6. We present the following four schemes:

ARP – The scheme *all request parts* always determines alternatives for all request parts.

FRRP – The scheme *failed and reserved request parts* derives alternatives for the failed part and some of the already reserved ones.

FNRP – The scheme *failed and not reserved request parts* determines alternatives for the failed part and some of the not yet reserved parts.

FTSD – The scheme *failed and temporally/spatially dependent request parts* acknowledges the temporal and spatial dependencies of the request.

Scheme – All Request Parts. The scheme *all request parts* (ARP) determines a complete new co-reservation candidate. That is, any new candidate of a request part must not be the same as the old one for that request part. Thus, all already reserved candidates need to be canceled. Since determining a complete new co-reservation effectively requires to execute the optimization procedure (cf. Chapter 10), this scheme may only

be used if the optimization does not consume too much time. While the scheme incurs a (possibly) large overhead and leads to less optimal co-reservation candidates, it may be easily implemented.

Scheme – Failed and Already Reserved Request Parts. The scheme ARP is far too restrictive, because it always cancels all already gathered reservations. Particularly, if the request parts are ordered with the scheme *smallest success probability first*, it would be better to retain the reservations with the smallest success probability.

The scheme *failed and already reserved request parts* (FRRP) tries to retain granted reservations by searching for alternatives of the failed part first. If no alternatives are found, the scheme gradually widens the search horizon. That is, it incrementally adds parts to the subset for which alternatives are looked for. The parts added are taken from those for which reservations were already allocated. Essentially, the scheme implements backtracking as illustrated in Fig. 11.1. The example of the illustration comprises seven reserve messages, three searches for alternative candidates, one backtracking and one cancel operation.

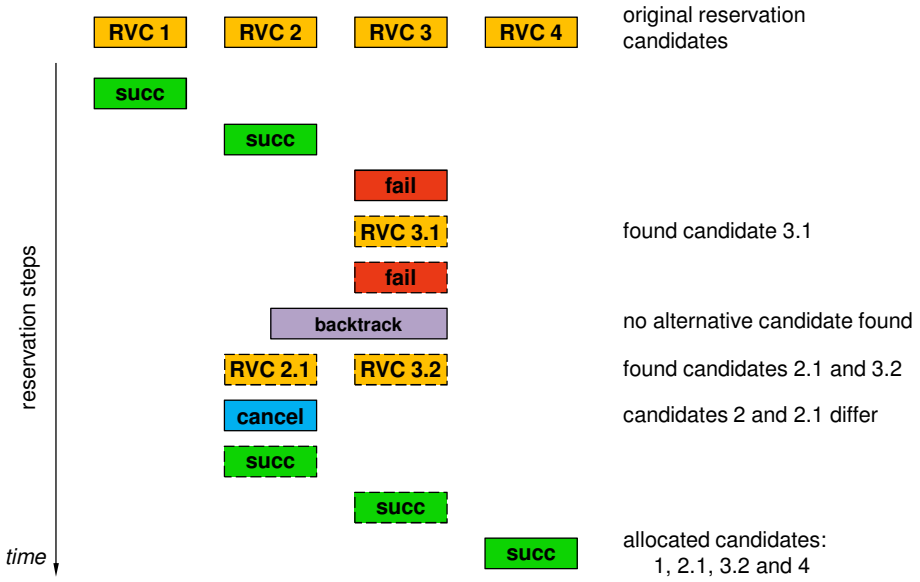

Figure 11.1: Illustration of the scheme *failed and already reserved request parts*.

Because the scheme FRRP retains as many granted reservations as possible, it should yield higher total reservation success rates, smaller cancelation fees and finish faster if the number of backtracking steps is small.

Scheme – Failed and Not Yet Reserved Request Parts. The scheme FRRP may require to cancel already granted reservations. In contrast, the scheme *failed and not yet reserved request parts* (FNRP) increases the search horizon by looking for alternative candidates of request parts which were not yet reserved.

First, similarly to the scheme FRRP, alternative candidates for the failed request part are searched for. Thereafter, the subset of request parts is gradually widened. The scheme FNRP considers the remaining request parts to relax some constraints.

Figure 11.2 shows the processing of an example request requiring six reserve messages, three searches for alternative candidates and one widening operation. The figure also shows that the reservation order may be adapted for the new reservation candidates.

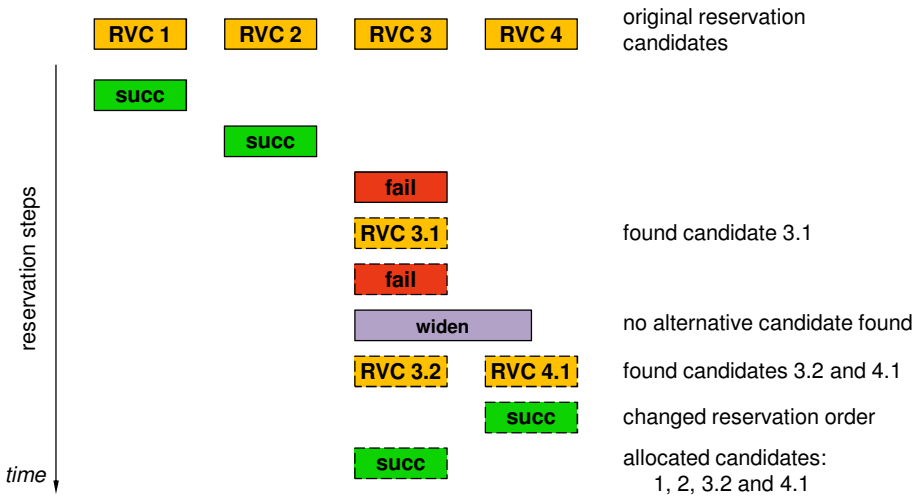

Figure 11.2: Illustration of the scheme *failed and not yet reserved request parts*.

The scheme FNRP shares the main benefits of the scheme FRRP – it should yield higher total reservation success rates, smaller cancelation fees and finish faster the later the first failure occurs. It outperforms the scheme FRRP because it does not need to cancel already granted reservations.

Scheme – Failed & Temporally/Spatially Dependent Request Parts. In general, the schemes FRRP and FNRP will outperform the simple scheme ARP, except if the request parts are in temporal or spatial relationships. In that case, it is more appropriate to widen the search horizon along the dependency graph with the failed part as starting node.

For example, Fig. 11.3 shows the spatial relationships on its top. The data part is connected via a network part to the second compute part (labeled CPU). The first compute part is not dependent on any other request part. If the reservation of the second compute part fails, the scheme *failed and temporally/spatially dependent request parts* (FTSD) – like the previous ones – determines an alternative reservation candidate of the failed part first. Thereafter, it widens the search horizon acknowledging the spatial relationships. Thus, the search for alternative candidates may reveal other resources for the data and network parts.

Since the schemes FRRP and FNRP only looked for new candidates of either the data (FRRP) or the network part (FNRP), it could not find candidates located at different re-

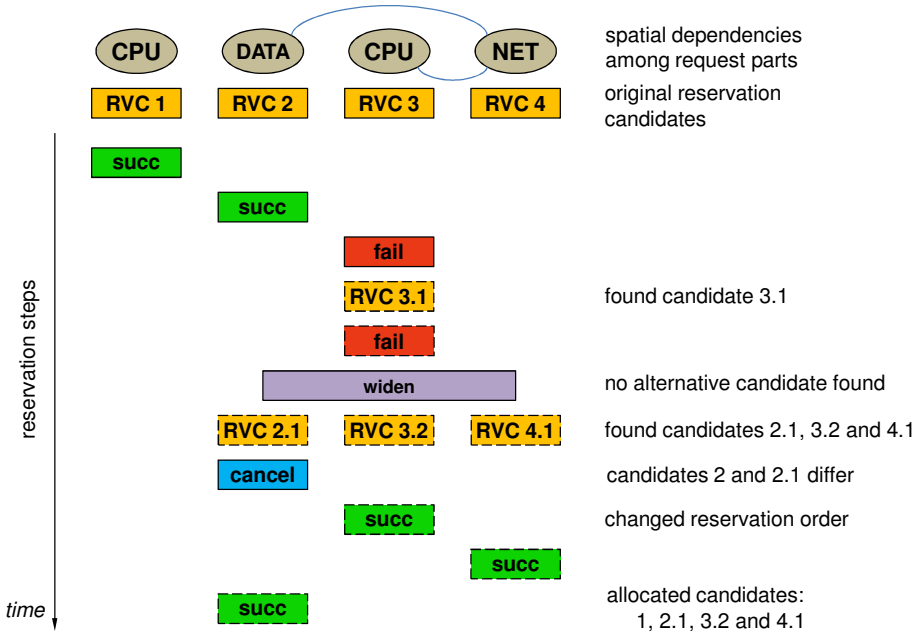

Figure 11.3: Illustration of the scheme *failed and temporally/spatially dependent parts*.

sources. For example, if the candidate of the network part is kept (scheme FRRP), only data and compute resources at the end-points of the network link satisfy the spatial constraints.

Clearly, the scheme FTSD increases the flexibility for finding alternative reservation candidates. Thus, it should outperform the other schemes in terms of the total reservation success rates, the cancelation fees and number of needed reservation steps.

11.4 Concurrently Allocating Resources

The concurrent allocation mechanism, shown in Alg. 3, sends reserve messages in parallel for all unrequested parts (line 2), collects responses (line 5) and acts upon them (lines 7 to 14). Besides sending reserve messages in parallel, a major difference to the sequential procedure is the necessity to keep track of the current state of a request part. In addition to the obvious states unrequested, requested, denied, granted, canceled the states expired and confirmed are needed if granted reservations are preliminary only. Figure 11.4 illustrates the transitions between different states.
Besides the states, the algorithm requires knowledge of the timeout p_i^{tmo} of each preliminarily granted request part i. A timeout value of zero encodes that the granted reservation is not preliminary and no confirmation is needed. Values greater than zero simply specify the UNIX epoch at which it expires. We assume that no messages are lost and that messages are received in the same order as they are sent.

While the concurrent algorithm is similarly structured as the sequential procedure (cf. Alg. 2), it also differs in certain aspects. First, it does not calculate an allocation or-

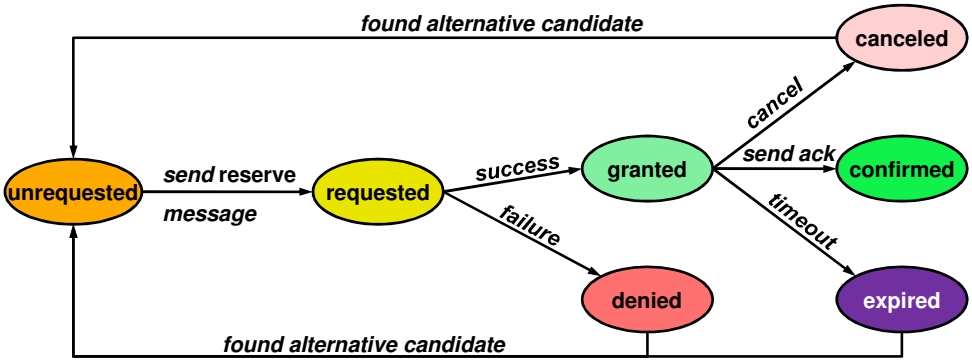

Figure 11.4: State changes of a request part with the concurrent allocation procedure.

der. Second, it keeps track of the yet to be reserved request parts by means of the state unrequested. Third, it avoids infinite waiting for responses by an additional response time limit τ. Forth, if a failure – a denied reserve message or response timeout – happens, it searches for alternative candidates. Fifth, it always sends a cancel message to parts in the state requested, too. Thus, we avoid ambiguous states of parts with messages in transit. Last, preliminary reservations with too short confirmation timeouts are canceled.
Next, we adapt the performance metrics presented for the sequential algorithm (cf. Section 11.3). Thereafter, we present different schemes for determining alternative co-reservation candidates.

Performance Metrics

The terms p_i^{cost}, p_i^{pen}, p_i^{esr} and p_i^{fit} denote the i-th request part's reservation cost, reservation cancelation penalty, estimated reservation success rate and reservation fitness, respectively. Note, we assume that allocations of resources to different request parts are independent of each other. The set I_{CAS} is defined as in Eq. (11.1).

We assume that determining alternative candidates consumes one round and alternatives' properties p_i^{cost}, p_i^{pen}, p_i^{esr} and p_i^{fit} will have the same values. The linearly decreasing success probability $\mathsf{p}(i,j)$ (cf. Example 11.1) is kept as if the candidate was not changed. The term $\overline{\mathsf{p}}(i,j)$ denotes the probability that the reservation of request part i fails in round j.

Reservation Success Rate. The reservation success rate may be recursively calculated by constructing a tree of state changes. Figure 11.5 shows an example tree for two request parts. For the sake of brevity, we assume the only possible states are unrequested, granted and denied. Accordingly, the permitted state changes are unrequested $\longmapsto$ granted, unrequested $\longmapsto$ denied and denied $\longmapsto$ unrequested. The first two state changes are the result of sending the message reserve. The third state change happens when alternative reservation candidates are looked for. Each node is marked with the states of the requests (upper row) and a probability (lower row). The states of the requests parts

Algorithm 3: General procedure for concurrently allocating resources.

```
1  while some parts are in state unrequested do
2      send reserve messages for all unrequested request parts
3      initialize timeout τ for response to reserve messages
4      while ¬ (received all responses ∥ response time τ expired) do
5          collect response and set state (granted or denied)
6          set timeout p_i^tmo for preliminarily granted reservations
7      if (some parts were denied ∥ are still requested) then
8          search for alternative co-reservation candidate
9          if search successful then
10             send cancel message to parts with changed candidate
11             set state of denied, requested and changed parts to unrequested
12         else
13             send cancel message to parts in state granted and requested
14             return FAILURE
15     cancel preliminary reservations with insufficient timeouts p_i^tmo
16 confirm preliminary reservations
17 return SUCCESS
```

are abbreviated, i.e., U for unrequested, G for granted and D for denied. For example, the mark G : D denotes the state granted for the first part and the state denied for the second part. The probability ϕ of a node denotes the likeliness that this node is reached in a sequence from the root node. The likeliness to reach a node at depth[2] k for a state change of request part i is calculated by

$$\phi(k,i) = \begin{cases} 1 & \text{if } k = 0\,, \\ \phi(k-1,i) & \text{if } \textbf{denied} \longmapsto \textbf{unrequested}\,, \\ \phi(k-1,i) \cdot \mathsf{p}(i,k) & \text{if } \textbf{unrequested} \longmapsto \textbf{granted}\,, \\ \phi(k-1,i) \cdot \overline{\mathsf{p}}(i,k) & \text{if } \textbf{unrequested} \longmapsto \textbf{denied}\,. \end{cases}$$

Usually, the tree has an infinite depth. We calculate the reservation success rate by adding the likeliness values ϕ of all nodes in the state G : ⋯ : G (nodes marked red in Fig. 11.5) up to a given depth k. The depth is either given explicitly or implicitly through a threshold on the likeliness value. That is, only nodes with a likeliness value greater than some value are considered.

Impact on the Utilization of a Resource. Because reservations block capacity requested by other jobs, providers want to minimize both the number of cancelations

[2]The number of the processing round or step corresponds to the depth in the tree.

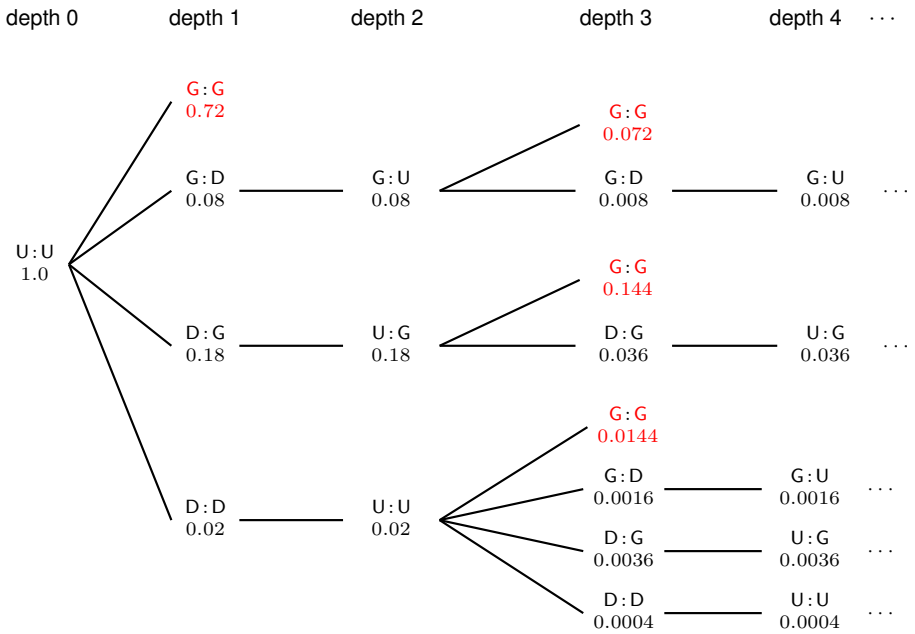

Figure 11.5: Tree of state changes for calculating the reservation success rate.

and the time between the approval of a reservation request and the reception of a cancelation request. Particularly, if granted reservations are preliminary, providers wish to receive potential cancelations as soon as possible, because they are not compensated for the loss incurred by denying other requests. In general, it is difficult to determine a-priori whether the sequential or the concurrent procedure has a larger impact on the utilization of a resource.

The impact on the utilization of a resource can be calculated by a similar approach as the reservation success rate (see above). Particularly, we adapt the tree of state changes by adding the loss function $\mathsf{ls}(i, j, k)$ for each request part (separated by a colon, middle row). The term $\mathsf{ls}(i, j, k)$ denotes the costs incurred by a reservation of part i granted at step j and canceled at step k. The notation of the states (upper row) and the calculation of the likeliness of nodes (lower row) is kept unchanged. Figure 11.6 shows the adapted tree of state changes.

The impact of a reservation of part i canceled at depth k is calculated by considering all nodes marked with the state G for that part. For all these nodes, the product of the likeliness value and the loss value is accumulated. For example, canceling the reservation of request part 2 in round 4, all red elements are used in the calculation of the cancelation fee. Note, if the resource assigned to a request part differs at different nodes, the nodes must be partitioned accordingly. Such situations may appear if a request part was mapped to a different resource in the search for alternative candidates (cf. Section 11.4.1).

Cancelation Fees. Cancelations of already granted reservations may induce some penalty, i.e., a cancelation fee. We adapt the tree implemented for the calculation of the

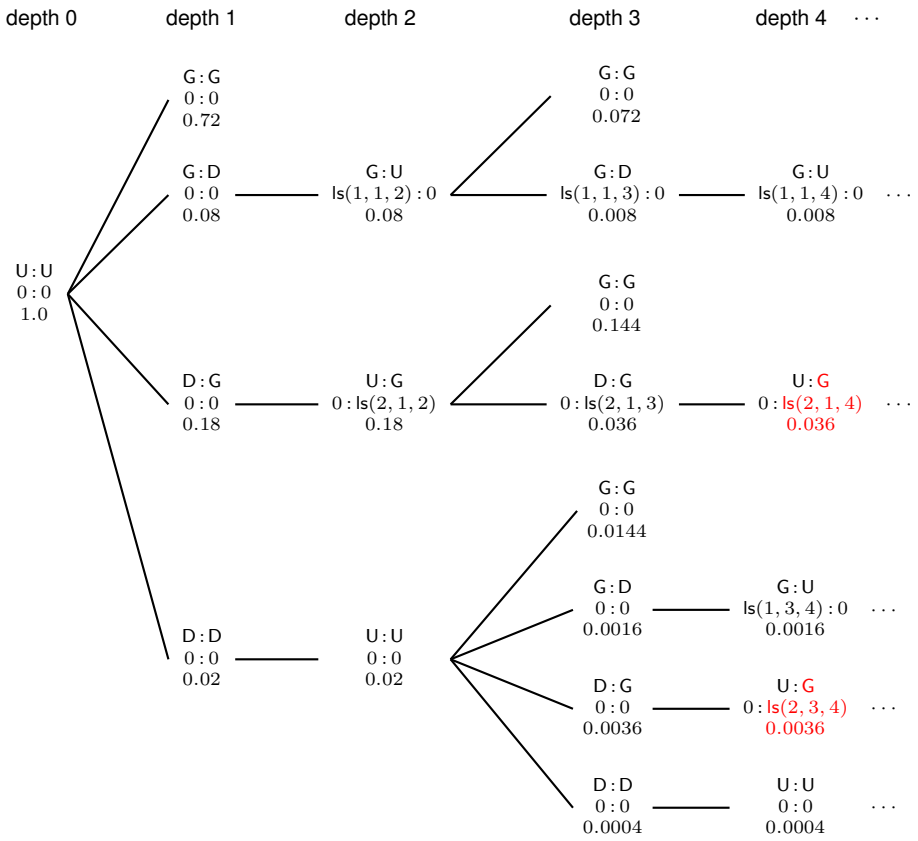

Figure 11.6: Tree of state changes for calculating the impact on the utilization.

impact of a reservation (cf. Fig. 11.6) by replacing the loss values $\mathsf{ls}(i, j, k)$ with the cancelation penalties p_i^{pen}. The adapted tree is shown in Fig. 11.7. The cancelation fee of a single request part is calculated in the same way as the impact on the utilization of a resource (see above). The cancelation fee of a co-reservation is the sum of the cancelation fees of all request parts.

11.4.1 Alternative Co-Reservation Candidates

If the allocation of resources to a co-reservation candidate fails, the reservation procedure (cf. Alg. 3) derives alternative candidates of a subset of all request parts. The benefit of searching for alternatives is, that some characteristics of the original co-reservation candidate may be kept. In particular, already granted reservations may be kept, which can be important if competition is high for those request parts. Furthermore, retaining as many granted reservations as possible supports limiting the cancelation fees.

In this section, we present schemes for deriving subsets of all request parts for which alternative candidates are determined thereafter. The actual mechanisms for determining such alternatives are discussed in Section 10.6. We present the following three schemes:

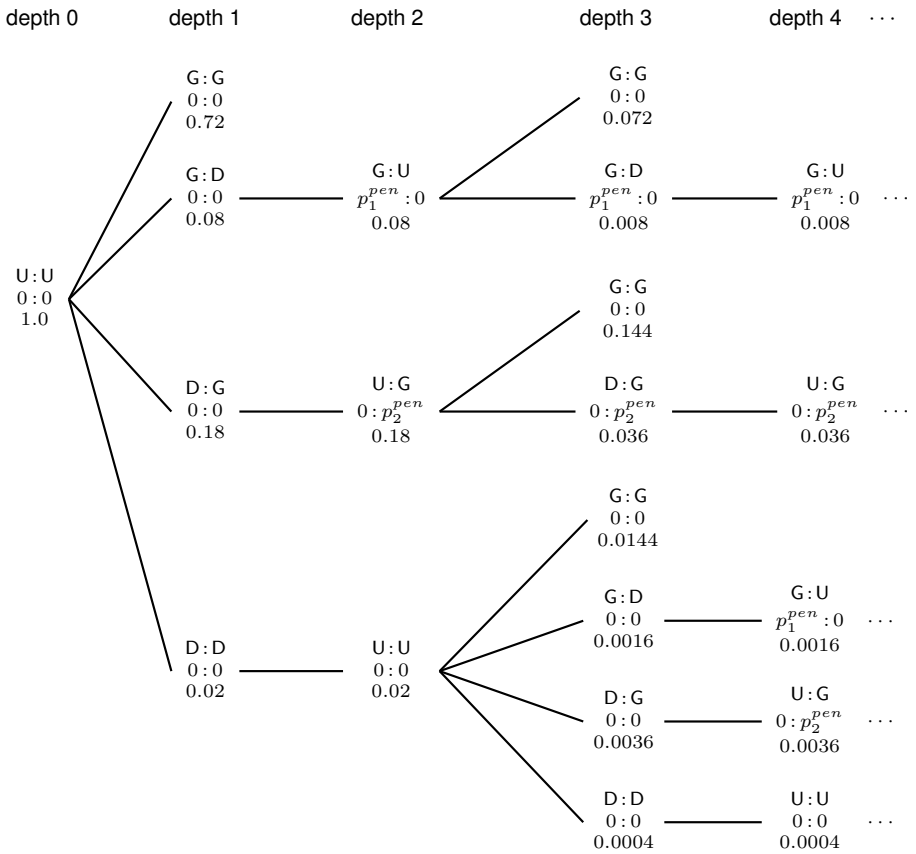

Figure 11.7: Tree of state changes for calculating the cancelation fee.

ARP – The scheme *all request parts* always determines alternatives for all request parts regardless of whether resources to some candidates were already reserved.

FEGP – The scheme *failed, expired and granted request parts* derives alternatives for the failed and expired parts as well as some of the already reserved ones.

FTSD – The scheme *failed and temporally/spatially dependent request parts* acknowledges the temporal and spatial dependencies among the request parts.

Scheme – All Request Parts. The scheme *all request parts* (ARP) determines a complete new co-reservation candidate. That is, any new candidate of a request part must not be the same as the old one for that request part. Thus, all already reserved candidates need to be canceled. Since determining a complete new co-reservation effectively requires to execute the optimization procedure (cf. Chapter 10), this scheme may only be used if the optimization does not consume too much time. While the scheme incurs a (possibly) large overhead and leads to less optimal co-reservation candidates, it may be easily implemented.

Scheme – Failed, Expired and Granted Request Parts. The scheme ARP always cancels all already gathered reservations. While this mechanism is easy to implement, it is far too restrictive.

The scheme *failed, expired and granted request parts* (FEGP) searches for alternatives of the failed, expired and granted request parts. If no alternatives are found for the initial set of failed and expired parts, the scheme gradually widens the search horizon. That is, it incrementally adds parts to the subset for which alternatives are looked for. The parts added are taken from those for which reservations were already allocated. Therefore, the scheme implements backtracking as illustrated in Fig. 11.8. The example of the illustration comprises eight reserve messages, three searches for alternative candidates and one cancel operation.

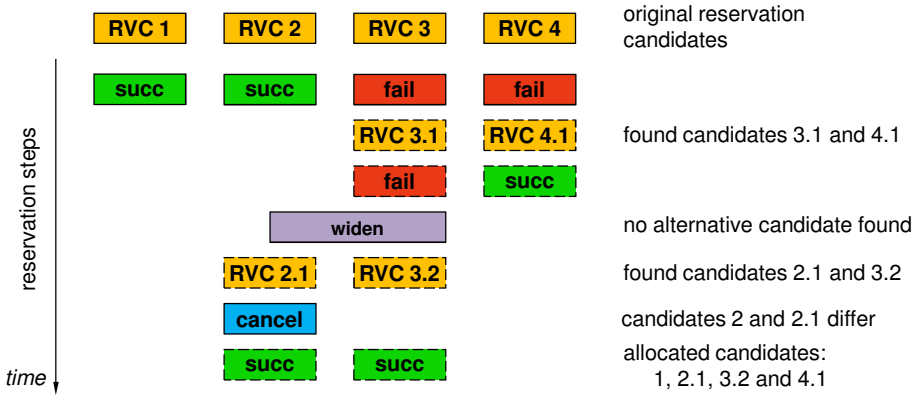

Figure 11.8: Illustration of the scheme *failed, expired and granted request parts*.

Because the scheme FEGP retains as many granted reservations as possible, it should yield higher total reservation success rates, smaller cancelation fees and finish faster if the number of widening operations is small.

Scheme – Failed & Temporally/Spatially Dependent Request Parts. In general, the scheme FEGP will outperform the simple scheme ARP, except if the request parts are in temporal or spatial relationships. In that case, it is more appropriate to widen the search horizon along the dependency graph with the failed and expired parts as starting nodes.

For example, Fig. 11.9 shows the spatial dependencies on its top. The data part is connected via a network part to the second compute part (labeled CPU). The first compute part is not dependent on any other request part. If the reservation of the second compute part fails, the scheme *failed and temporally/spatially dependent request parts* (FTSD) – like the previous one – determines an alternative reservation candidate of the failed and expired part first. Thereafter, it widens the search horizon acknowledging the spatial relationships. Thus, the search for alternative candidates may reveal other resources for the data and network parts.

Since the scheme FEGP does not automatically acknowledge the request's internal structure, it might not find candidates located at different resources. For example, if

the candidate of the network part is kept (widening step in Fig. 11.8), only data and compute resources at the end-points of the network link satisfy the spatial constraints.

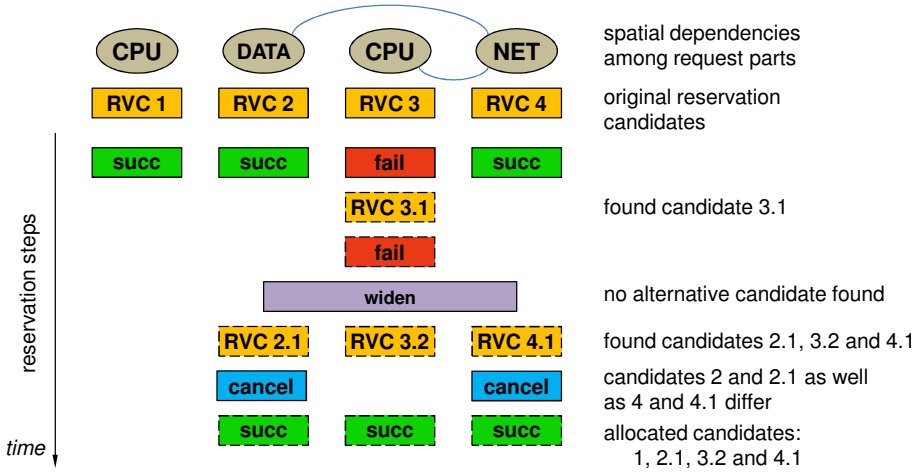

Figure 11.9: Illustration of the scheme *failed and temporally/spatially dependent parts*.

Clearly, the scheme FTSD increases the flexibility for finding alternative reservation candidates. Thus, it should outperform the other schemes in terms of the total reservation success rates, the cancelation fees and number of needed reservation steps.

11.5 Summary

Both the sequential and the concurrent mechanism may trade the optimality of the allocated co-reservation with the efficiency of obtaining it. That is, if one or more candidates may not be reserved, the optimal alternative co-reservation candidate requires that a complete new solution is derived by the GRS. Determining new candidates only for the failed parts may result in non-optimal candidates, but likely yields a higher efficiency due to fewer canceling operations. The number of search operations may be slightly smaller for the concurrent scheme, because failures might be detected earlier as in the sequential scheme. Also, the number of reservation steps will generally be smaller for the concurrent scheme, simply because it exploits parallelism. If the estimated success rate of the candidates decreases rapidly with the number of reservation steps, the total reservation success rate of the concurrent scheme will be higher. Because the sequential mechanism is well suited to control the cancelation cost, it should be used if granted reservations require a cancelation fee. If not all parts require a cancelation fee, the schemes can be combined.

The proposed mechanisms only take the allocation efficiency and the impact of temporary reservations into account. Component failures such as crashing or non-responding services were not in the focus of this chapter, mainly because these were extensively studied in research on distributed systems, specifically on failure models, consensus and transaction commit.

Chapter 12

Using Confirmed Co-Reservations

We introduce the concept of *Virtual Resources* and describe basic and advanced functions for using confirmed co-reservations.

12.1 The Concept of Virtual Resources

We generalize the resource reservation model by introducing ***Virtual Resources*** (*VR*), which correspond to confirmed co-reservations. Virtualization is a common concept found in Grid middleware [CFK+98, RLS03] and many other application scenarios like the Java 2 Runtime Environment [Jav04] or VMware [Vmw08]. In all these cases, an abstraction of resources – computers, operating systems, and services – is provided. The abstraction makes it easier to support the integration of new resources with new properties. Hence, mechanisms building atop *Virtual Resources* can be deployed on any entity which fits into this concept of abstraction.

By viewing a confirmed co-reservation as a *Virtual Resource*, a co-reservation may be subject to subsequent reservation requests, thereby allowing nested reservations. Besides abstracting from resources, virtualization may be used to extend the functionality of resource management systems. This enables advanced usage scenarios such as customized workload scheduling within co-reservations and transparent fault recovery for atomic reservations.

Virtual Resources are built atop physical or virtual resources. The concept of *Virtual Resources* is realized by the following **basic functions** any *VR* must provide:

- reserving a time-qos-slot of the *VR* (*reserve*),
- canceling a confirmed reservation (*cancel*),
- submitting workload to a confirmed reservation (*submit*),
- releasing workload from a confirmed reservation (*release*), *and*
- querying for the current and the future status (*query* and *probe*).

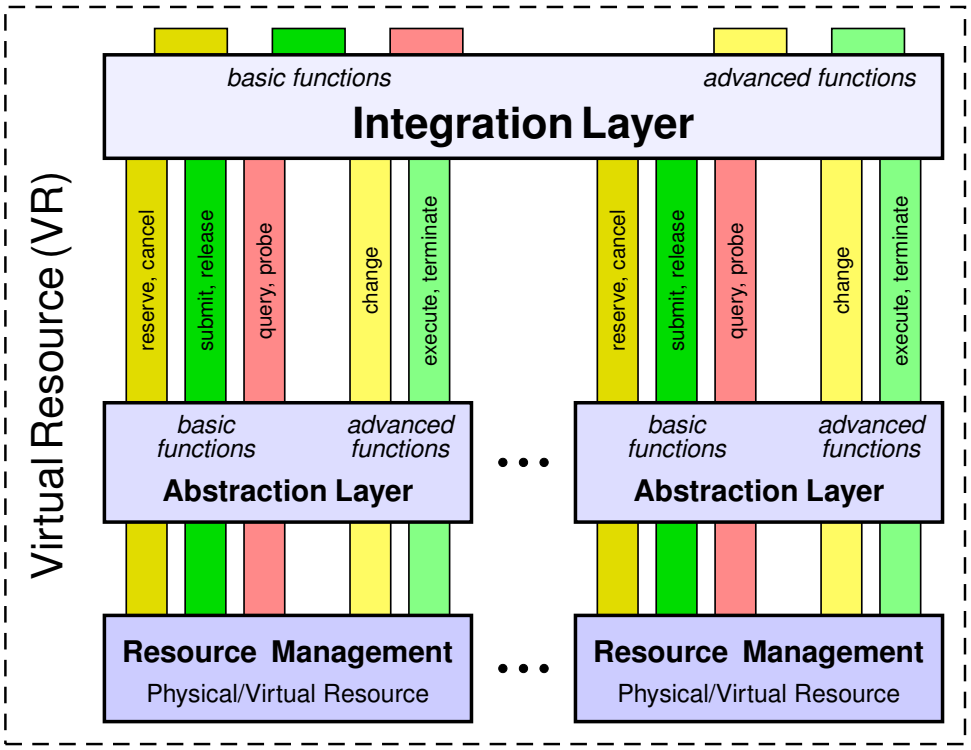

Figure 12.1: Composition of a *Virtual Resource* (*VR*).

Enhanced usage scenarios are possible if a *VR* supports **advanced functions**[1]:

- allowing customized workload scheduling within a reservation (*execute* and *terminate*), *and*
- managing resource allocation faults (*change*).

Figure 12.1 shows the layered model of a *Virtual Resource*. The lower two layers build the core of a *VR*. The upper layer provides means for the integration of multiple resources into a single *virtual* object and for facilitating transparent fault recovery. Bottom-up, we have the following components:

Resource Management of a Physical/Virtual Resource: A physical/ virtual resource is managed by an admission control system (such as Maui [JSC01] or CCS [KR98]) for handling the access to compute resources. It must provide functions for reserving a specific amount of the physical or virtual resource for a given time windows and for assigning workload to confirmed reservations. Functions for querying its current and future status are also necessary. If they are not available, they can be emulated in the *Abstraction Layer*.

Abstraction Layer: This layer provides a conversion from the (proprietary) world of a specific resource to an instantiation of a *VR* and vice versa. The former conversion is used to transform the status information, while the latter provides an

[1]See Section 12.3 for a detailed discussion of their functionality.

adaptation of various functions to the (proprietary) management system. Additionally, the abstraction layer may enforce access policies employed for integrating resources into a Grid environment, e.g., Globus authorization [FKTT98]. Furthermore, functionality missing in a resource's management system can be added by separate plug-ins or by techniques as described in [RSR03].

Integration Layer: The integration layer provides two features. First, it integrates multiple resources into a single virtual object, thereby allowing a consistent modeling of the underlying resources, i.e., estimating the current and the future status of the resources. Second, because a client only interacts with this layer, transparent fault recovery may be implemented here if the underlying resource management system does not provide means for substituting faulty resources.

12.2 Basic Functions

The basic functions for managing co-reservations are listed below.

Function *reserve*: Co-reservations are requested with the function *reserve* (cf. Chapter 7 for specifying requests). Because a co-reservation is seen as *Virtual Resource*, it may be subject to subsequent co-reservation requests. Hence, nested *Virtual Resources* may be constructed.

Function *cancel*: A co-reservation, including all atomic reservation parts and workload bound to them, is canceled with the function *cancel*. However, it may be possible to retain the workload within the scope of the resource management system from which the reservation was granted and let the workload be processed in best-effort manner.

Function *submit*: Workload, e.g., compute jobs or data transfers, is bound to reserved resources by the function *submit*. This is a generalization of submission functions found in common batch management systems such as OpenPBS [Hen95], LSF [ZZWD93] or CCS [KR98].

Function *release*: Workload is removed from a co-reservation by the function *release*.

Functions *query* & *probe*: Status information about the resources within a co-reservation (e.g., OS type, CPU speed, RAM size, network bandwidth & latency, idle or busy, etc.) and the workload of a co-reservation (e.g., task owner, runtime limit, required RAM size, required network bandwidth, waiting/running, etc.) is obtained by the functions *query* & *probe*. The layered structure of the *Virtual Resource*, allows both an efficient access to the information and fine-grained authorization for receiving different levels of details.

12.3 Advanced Functions

Advanced functions include mechanisms to schedule workloads from external components (cf. Section 12.3.1), to aggregate resources (cf. Section 12.3.2), and to change resources for implementing fault recovery (cf. Section 12.3.3).

12.3.1 External Workload Scheduling

Todays local resource management systems only allow to bind workload to reservations, but the actual scheduling of this workload is performed by the local scheduler. This is sufficient for workloads containing independent entities – i.e., single jobs or data transfers – or if the reservation's owner is satisfied with local scheduling policies.

In contrast, workloads whose entities depend on each other may be handled more efficiently by an external scheduler. Essentially, a reservation owner should be allowed to configure (i.e., optimize) the scheduling policies applied to his workload according to his needs.

Adjusting local scheduling policies to the user's needs can be achieved in two ways. First, the local scheduler can be reconfigured such that resources and workload bound to a reservation are managed in a different way. For example, the Maui scheduler [JSC01, Mau04] allows to change the calculation of job priorities and to adjust scheduling properties by attaching quality-of-service levels both to reserved resources and jobs. In general, this approach depends on the built-in flexibility in the local resource management systems.

Second, the local scheduler can be bypassed or switched off for the reserved resources. Then an external system takes over the control over admission of workload to the reserved resources. We favor this approach, because it is more flexible in which scheduling algorithm can be used.

A *VR*'s resource management system essentially must provide a *query* function to obtain status information about the resources and the workload (cf. Section 12.2) and functions for executing and terminating workload elements. The latter functions are listed below.

Function *execute*: The execution of a workload element (on a subset of the reserved resources), e.g., starting a compute job, starting a data transfer, etc., is started by the function *execute*.

Function *terminate*: The execution of a workload element is stopped by the function *terminate*.

12.3.2 Resource Aggregation

Although, the Grid potentially provides an abundant number of resources, reservation requests may fail because no single resource provider matches the requested amount. In such situations a reservation request requiring a large capacity can be split into

several atomic reservation requests, each requiring a smaller amount of capacity. This requires, however, that the considered workload may be split into smaller portions and distributed to several resources. Due to the many constraints and side-effects, the splitting is a difficult task by itself which must be done by the requester.

The integration layer of the *VR* may provide a consistent view of multiple atomic reservations. Thus, a user does not notice the scattering of a reservation over multiple resources. The integration layer may also implement load balancing strategies to execute a workload as efficiently as possible.

Another application of resource aggregation capabilities is the adaptation of the *Virtual Resource*'s capacity to variable workload demand. For example, in utility computing demand predictions [AC05] may be used to increase/decrease the performance of a *VR* by transparently adding/removing atomic reservation parts.

12.3.3 Fault Recovery

In practice, a situation might occur where the reserved resources cannot be claimed, i.e., by executing workload entities on them. This may be due to resource defects or due to other workload elements that claim the requested resources with a higher priority. When the resource provider is able to substitute the missing resource by some spare one, the problem can be solved locally. Otherwise the whole reservation (or parts thereof) may be moved to another provider by replacing some atomic reservation parts. We follow the latter approach, because it does not depend on specific features provided by the local resource providers. If no alternative resource is available, the workload processing still may continue in *best-effort* manner or the whole co-reservation will be canceled. Which policy is appropriate depends on the workload and the user's requirements.
User dependent fault recovery may be implemented by associating fault recovery rules with a co-reservation request. These rules give the integration layer guidelines on how to cope with faults. The following example illustrates a co-reservation with two atomic parts $A1$ and $A2$ which both have been confirmed, but failed afterwards. In this case, the following recovery rules will be applied.

```
A1: fault recovery =
  confirmed & unavailable --> reissue request;
  requested & failed  --> done, cancel co-reservation

A2: fault recovery =
  confirmed & unavailable --> done, continue in best-effort;
  requested & failed  --> done, continue in best-effort
```

When part $A1$ becomes unavailable, an alternative resource is looked for (`reissue` action). If the search fails, part $A1$ changes its status to done (cf. Fig. 6.1) and the whole co-reservation is canceled. In contrast, if part $A2$ becomes unavailable or the original request failed, the workload bound to the co-reservation is handled in best-effort manner.

Part III

Conclusion

Chapter 13

Summary

This thesis proposes CORES – a **generic** framework for *specifying*, *processing* and *using* co-reservations of resources in the Grid. The developed mechanisms acknowledge the characteristics of Grid environments – most importantly the **autonomy of the resources** and the **lack of global information** – and provide rich means to **balance the goals** of all stakeholders. Particularly, CORES provides

- *a mathematical formulation of the co-reservation problem*,
- *a simple yet powerful language for describing co-reservation requests and resources*,
- *a versatile mechanism for determining the future status of the resources*,
- *an optimization approach for mapping requests to co-reservation candidates*,
- *goal-driven mechanisms for allocating resources to a co-reservation candidate*, and
- *a concept for embedding reservations into Grid environments*.

It is easy to see, that the problem of reserving multiple resources in advance can be modeled as optimization problem. The *very* generic form

$$\begin{array}{rll} \min & f(\mathbf{x}) & \\ \text{subject to} & h(\mathbf{x}) = 0 \quad \text{and} & g(\mathbf{x}) \geq 0 \end{array}$$

of an optimization problem, however, is of little use. Therefore, this thesis provides the means to define the functions $h(\mathbf{x})$, $g(\mathbf{x})$ and $f(\mathbf{x})$ at the necessary – yet convenient to use – level of detail.

The **mathematical formulation** (cf. Chapter 5) of the co-reservation problem introduces the concept of **properties** which capture information about the requests and the resources. This information may depend on the problem variables composing the vector $\mathbf{x} \in TDQ^L$, where TDQ is the domain space of a single atomic request and L is

the number of parts of a co-reservation. In this thesis, the variables were the **start time** (domain T), the **duration** (domain D) and the **service level** (domain Q) of a reservation. The properties are used to define both the constraints (h and g) and the utility (f). Besides generic forms of constraints, the mathematical formulation provides means to define **temporal** and **spatial** relationships among the parts of a co-reservation.

The **Simple Reservation Language** (cf. Chapter 7) builds upon the well-known *Condor ClassAds* [RLS98] to support symmetric matching between requests and resources. By grouping attributes into scopes and supporting attributes with pre-defined semantics the use of the language is kept simple despite the more complex scenarios. Relationships between request parts are facilitated by identifying each attribute with a part and referencing them in constraints.

The mechanisms for determining the future status of the resources (cf. Chapter 9) allow an efficient and flexible calculation of the properties. Instead of separately asking for the future status at individual points of the space TDQ, **flexible** probe requests are issued. The resources may provide as much information as their privacy policy allows. By calculating distributions of **time-qos-slots** the resources can trade-off the accuracy of the properties' values with the computational complexity. We evaluated the *probing* mechanisms through extensive simulations. The evaluation demonstrated the effectiveness of the approach and revealed a specific workload pattern which may cause significant job delays.

The mapping of requests to co-reservation candidates (cf. Chapter 10) demonstrates two implementations of the mathematical formulation of the the co-reservation problem. The **Integer Programming** (IP) model aims at very fine-grain solutions, i.e., virtually any element of the space TDQ^L may be selected in the mapping. The IP model, however, severely restricts the modeling capabilities of the properties which also affects the representation of the constraints and objectives. Moreover, the approach does not scale to typical sizes of Grid environments. Because the IP model offers no adequate means to reduce the complexity, it is not well suited for mapping requests to co-reservation candidates. The **Binary Programming** (BP) model significantly reduces the solution granularity by considering a configurable number of points of the space TDQ^L only. This has the effect that any property can be modeled, but constraints and objectives are limited to linear combinations of the properties still. Also, the computational complexity is greatly reduced. Eventually, it offers a simple means to trade-off the granularity of a mapping with the computational complexity. Therefore, we claim that BP is an adequate means for mapping requests to co-reservation candidates.

The allocation of resources to a co-reservation candidate (cf. Chapter 11) must implement an **all-or-nothing** semantics. This problem is very similar to transaction commit, which was extensively studied in the fields of distributed database management and composition of web services. Therefore, we set the focus on goal-driven allocation mechanisms. The aim of such mechanisms is to satisfy certain goals such as maximizing the **reservation success rate**, minimizing the **impact** of the allocation mechanism on the utilization of the resources and minimizing the **cancelation fees**. We studied these goals in the context of **sequentially** and **concurrently** allocating resources. Par-

ticularly, we defined the above goal metrics, presented the sequential and concurrent algorithms, and illustrated schemes for handling allocation failures.
The concept of **virtual resources** (cf. Chapter 12) seamlessly embeds co-reservations into Grid infrastructures. A virtual resource contains an abstraction layer and an integration layer. The former provides a uniform interface for managing virtual resources. The latter aggregates several resources, supports a unique view on them and may extend the basic functionality with transparent fault recovery.

13.1 Outlook

While the proposed framework already supports a wide range of scenarios, it may be enhanced in several ways. Here, we only briefly describe the most interesting of them.

We assumed that granted reservations may be claimed without any exception. This is favorable for clients, but puts a high burden on the resource providers. Thus, the approach could be enhanced by letting providers vary the level of guarantee in the interval $[0, 1] \subset \mathbb{R}$. In the current model, the level is one for granted reservations (and zero for denied reservations). Introducing such flexible levels of guarantees enables further means for a provider to lower the impact of reservations, but requires additional efforts at the client side. That is, a requester might need multiple reservations for a single part of a co-reservation to ensure that at least one may be claimed.

The proposed mapping models (cf. Chapter 10) are very generic. In specific application scenarios, they may be tuned by using domain- and environment-specific information for further improving the scalability. For example, knowing that the finish time of an application is the primary optimization criteria, load data of the resources could be used to begin the search at lightly loaded periods. Also, meta-heuristics such as genetic algorithms, tabu search, etc. may be adopted for increasing the modeling capabilities. Very preliminary experiments with genetic algorithms have not lead to any performance gains tough.

Currently, all parts of a co-reservation request are considered in the optimization mechanism. Particularly, for sequential job chains this approach seems to be to strict. It could be sufficient to only reserve resource for the first two to three steps and add the ones for the higher steps if the first finished. Such scheme may not only lower the computational complexity, but also enable more dynamic applications whose control flow depends on the results of its steps.

In this work, we exploited an *all-or-nothing* semantics for allocating resources to a co-reservation candidate. In some cases, it may be interesting to employ a relaxed atomicity criterion. For example, if some network links are not technically reservable, a probabilistic reservation for these links may be considered. The level of guarantee may be raised by taking backup links into account and by adapting the execution time predictions of the whole scenario. Then, one important question is whether the reservation model copes with different levels of guarantees from the beginning or whether it works in two phases – first the "traditional" 0/1-semantics are applied, then decreased guarantee levels are considered. Assuming that low quality-of-service requirements

are more easily to obtain, there is a clear trade-off between reserving such low QoS with the traditional semantics and achieving high QoS but with lower probabilistic guarantee levels.

Bibliography

[AAA+07] Alves, Alexandre; Arkin, Assaf; Askary, Sid; Barreto, Charlton; Bloch, Ben; Curbera, Francisco; Ford, Mark; Goland, Yaron; Guízar, Alejandro; Kartha, Neelakantan; Liu, Canyang Kevin; Khalaf, Rania; König, Dieter; Marin, Mike; Mehta, Vinkesh; Thatte, Satish; van der Rijn, Danny; Yendluri, Prasad; Yiu, Alex: Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Technical Report wsbpel-2.0, OASIS, April 2007.

[Aar03] Aarseth, Sverre J.: *Gravitational N-Body Simulations: Tools and Algorithms*. Cambridge University Press, Cambridge, United Kingdom, November 2003. ISBN 0-52143-272-3.

[AC05] Andrzejak, Artur; Ceyran, Mehmet: Characterizing and predicting resource demand by periodicity mining. In: *Journal of Network and System Management*, volume 13(2):pp. 175–196, June 2005. (Accepted for a special issue on Self-Managing Systems and Networks).

[ACD+07] Andrieux, Alain; Czajkowski, Karl; Dan, Asit; Keahey, Kate; Ludwig, Heiko; Nakata, Toshiyuki; Pruyne, Jim; Rofrano, John; Tuecke, Steve; Xu, Ming: Web Services Agreement Specification (WS-Agreement). Technical Report GFD-R-P.107, Open Grid Forum (OGF), March 2007.

[ADF+01] Allen, Gabrielle; Dramlitsch, Thomas; Foster, Ian; Karonis, Nicholas; Ripeanu, Matei; Seidel, Ed; Toonen, Brian: Supporting efficient execution in heterogeneous distributed computing environments with cactus and globus. In: *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM), Denver, CO, USA*, pp. 52–52. ACM Press, New York, NY, USA, November 2001.

[Amd67] Amdahl, Gene M.: Validity of the single-processor approach to achieving large scale computing capabilities. In: *Proceedings of the AFIPS Conference, Atlantic City, N.J., USA*, pp. 483–485. AFIPS Press, April 1967.

[BAG00] Buyya, R.; Abramson, D.; Giddy, J.: Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In: *Proceedings of the 4th International Conference and Exhibition on*

High Performance Computing in Asia-Pacific Region (HPC ASIA 2000), Beijing, China. IEEE Computer Society Press, Los Alamitos, CA, USA, May 2000.

[BBES05] Brandic, I.; Benkner, S.; Engelbrecht, G.; Schmidt, R.: Qos support for time-critical grid workflow applications. In: *First International Conference on e-Science and Grid Technologies (e-Science 2005), 5-8 December 2005, Melbourne, Australia*, pp. 108–115. IEEE Computer Society, 2005.

[BGKR98] Brune, Matthias; Gehring, Jörn; Keller, Axel; Reinefeld, Alexander: RSD – Resource and Service Description. In: *Proceedings of the 12th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'98)*, volume 478 of *The Kluwer International Series in Engineering and Computer Science*, pp. 364–378. Kluwer Academic Publishers, Boston, MA, Edmonton, Alberta, Canada, 1998.

[BHL+06] Burchard, Lars-Olof; Heiss, Hans-Ulrich; Linnert, Barry; Schneider, Jörg; Kao, Odej; Hovestadt, Matthias; Heine, Felix; Keller, Axel: The Virtual Resource Manager: Local Autonomy versus QoS Guarantees for Grid Applications. In: Getov, Vladimir; Laforenza, Domenico; Reinefeld, Alexander, editors, *Future Generation Grids*, volume 2 of *CoreGrid*, pp. 83–98. Springer Science + Business Media Inc., January 2006.

[Bur04] Burchard, Lars-Olof: *Advance Reservations of Bandwidth in Computer Networks*. Ph.D. thesis, Technische Universität Berlin, Berlin, 2004. Presented on 14 July 2004.

[Cac06] The Cactus Code, Nov 2006. URL `http://www.cactuscode.org`.

[CBC06] Choudry, Bilal A.; Bertok, Peter; Cao, Jinli: Cost based web services transaction management. In: *Int. J. Web and Grid Services*, volume 2(2):pp. 198–220, 2006.

[CCG+05] Capit, Nicolas; Costa, Georges Da; Georgiou, Yiannis; Huard, Guillaume; Martin, Cyrille; Mounié, Grégory; Neyron, Pierre; Richard, Olivier: A batch scheduler with high level components. In: *Proceedings of the 5th IEEE International Symposium on Cluster computing and Grid 2005 (CCGrid05), Cardiff, Wales, UK*, volume 2, pp. 776–783. IEEE Computer Society, Los Alamitos, CA, USA, May 2005. URL `http://oar.imag.fr/papers/oar_ccgrid05.pdf`.

[CFK+98] Czajkowski, Karl; Foster, Ian; Karonis, Nick; Kesselman, Carl; Martin, Stuart; Smith, Warren; Tuecke, Steven: A resource management architecture for metacomputing systems. In: *Proceedings of the 4th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '98), Orlando, FL, USA*, volume 1459 of *Lecture Notes in Computer Science*, pp. 62–82. Springer-Verlag, 1998.

[CFK99] Czajkowski, K.; Foster, I.; Kesselman, C.: Resource co-allocation in computational grids. In: *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8)*, pp. 219–228, 1999. URL http://ieeexplore.ieee.org/iel5/6521/17411/00805301.pdf.

[CFK+02] Czajkowski, Karl; Foster, Ian T.; Kesselman, Carl; Sander, Volker; Tuecke, Steven: SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In: *Proc. of the 8th Int'l Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, Scotland, UK*, volume LNCS(2537):pp. 153–183, 2002.

[CL01] Chen, Y. T.; Lee, K. H.: A flexible service model for advance reservation. In: *Computer Networks*, volume 37(2001):pp. 251–262, 2001.

[CPEV05] Canfora, Gerardo; Penta, Massimiliano Di; Esposito, Raffaele; Villani, Maria Luisa: An approach for qos-aware service composition based on genetic algorithms. In: *Genetic and Evolutionary Computation Conference, GECCO 2005, Washington DC, USA, June 25-29, 2005*, pp. 1069–1075. ACM, 2005.

[Dow97] Downey, Allen B.: Using Queue Time Predictions for Processor Allocation. In: *IPPS '97: Proceedings of the Job Scheduling Strategies for Parallel Processing*, pp. 35–57. Springer-Verlag, London, UK, 1997. ISBN 3-540-63574-2.

[EDG08] EU DataGrid Project Homepage, September 2008. URL http://eu-datagrid.web.cern.ch/eu-datagrid/.

[EDT08] EU DataTAG Project Homepage, Sep 2008. URL http://datatag.web.cern.ch/datatag/.

[EGE08] EGEE Workload Management System, Sep 2008. URL http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/index.shtml.

[EHY02] Ernemann, Carsten; Hamscher, Volker; Yahyapour, Ramin: Economic scheduling in grid computing. In: Feitelson, Dror G.; Rudolph, Larry; Schwiegelshohn, Uwe, editors, *Proc. 8th Job Scheduling Strategies for Parallel Processing in conjunction with HPDC/GGF 5*, volume 2537 of *Lecture Notes in Computer Science (LNCS)*, pp. 128–152. Springer, 2002.

[EKL98] Evans, K.; Klein, J.; Lyon, J.: Transaction internet protocol – requirements and supplemental information. RFC 2372, 1998.

[ELLR90] Elmagarmid, Ahmed K.; Leu, Yungho; Litwin, Witold; Rusinkiewicz, Marek: A multidatabase transaction model for interbase. In: *Proceedings of the 16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia*, pp. 507–518. Morgan Kaufmann, 1990.

[ESR+07] Enke, Harry; Steinmetz, Matthias; Radke, Thomas; Reiser, Angelika; Röblitz, Thomas; Högqvist, Mikael: AstroGrid-D: enhancing astronomic science with grid technology. In: *Proceedings of the German E-Science Conference (GES)*. Baden-Baden, Germany, May 2007.

[Fei07] Feitelson, Dror G.: Parallel Workloads Archive, August 2007. URL `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[FF05] Frachtenberg, Eitan; Feitelson, Dror G.: Pitfalls in Parallel Job Scheduling Evaluation. In: *Proceedings of the 11th Workshop on Job Scheduling Strategies for Parallel Processing, Cambridge, MA, USA*, volume 3834 of *Lecture Notes in Computer Science*, pp. 257–282. Springer-Verlag, 2005.

[FGPS07] Freitag, S.; Grimme, C.; Papaspyrou, A.; Schley, L.: On the applicability of OGSA-BES to D-Grid community scheduling systems. In: *Proceedings of the German E-Science Conference (GES)*. Baden-Baden, Germany, May 2007.

[FHC04] Furniss, Peter; Haugen, Bob; Ceponkus, Alex: Business Transaction Protocol (BTP) Version 1.1. Technical Report btp-1.1, OASIS, November 2004.

[FJ07] Feingold, Max; Jeyaraman, Ram: Web Services Coordination (WS-Coordination) Version 1.1. Technical Report wstx-wscoor-1.1-spec-errata-os, OASIS, July 2007.

[FK99] Foster, Ian; Kesselman, Carl, editors: *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers Inc., 1999.

[FKL+99] Foster, Ian; Kesselman, Carl; Lee, Craig; Lindell, Bob; Nahrstedt, Klara; Roy, Alain: A distributed resource management architecture that supports advance reservations and co-allocation. In: *Proceedings of the 7th International Workshop on Quality of Service (IWQoS), London, UK*, pp. 27–36. IEEE Press: Piscataway, NJ, June 1999.

[FKTT98] Foster, Ian; Kesselman, Carl; Tsudik, Gene; Tuecke, Steven: A security architecture for computational grids. In: *Proceedings of the 5th ACM Conference on Computer and Communications Security, San Francisco, CA, USA*, pp. 83–92. 1998.

[FL07] Freund, Tom; Little, Mark: Web Services Business Activity (WS-BusinessActivity) Version 1.1. Technical Report wstx-wsba-1.1-spec-errata-os, OASIS, July 2007.

[Fos02] Foster, Ian: What is the grid? a three point checklist. In: *GRIDtoday*, volume 1(6), 2002.

[FPD+05] Fahringer, T.; Prodan, R.; Duan, Rubing; Nerieri, F.; Podlipnig, S.; Qin, Jun; Siddiqui, M.; Truong, Hong-Linh; Villazon, A.; Wieczorek, M.: Askalon: A

grid application development and computing environment. In: *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (GRID '05)*, pp. 122–131. IEEE Computer Society, Washington, DC, USA, 2005.

[FR98] Feitelson, D. G.; Rudolph, L.: Metrics and benchmarking for parallel job scheduling. In: *Lecture Notes in Computer Science*, volume 1459:pp. 1–24, 1998.

[FTL+01] Frey, James; Tannenbaum, Todd; Livny, Miron; Foster, Ian; Tuecke, Steven: Condor-g: A computation management agent for multi-institutional grids. In: *HPDC '01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, p. 55. IEEE Computer Society, Washington, DC, USA, 2001.

[Gad06] Cosmological Simulations with GADGET, November 2006. URL `http://www.mpa-garching.mpg.de/gadget/`.

[GAL+03] Goodale, Tom; Allen, Gabrielle; Lanfermann, Gerd; Massó, Joan; Radke, Thomas; Seidel, Ed; Shalf, John: The cactus framework and toolkit: Design and applications. In: *Vector and Parallel Processing - VECPAR '2002, 5th International Conference*, volume 2565 of *Lecture Notes in Computer Science*, pp. 197–227. Springer-Verlag, June 2003.

[GLO08] The Globus Project, May 2008. URL `http://www.globus.org/`.

[GLPS07] Grimme, Christian; Langhammer, Tobias; Papaspyrou, Alexander; Schintke, Florian: Negotiation-based choreography of data-intensive applications in the c3grid project. In: *Proceedings of the German E-Science Conference (GES)*. Baden-Baden, Germany, May 2007.

[GLU07] GLUE Schema 1.3 Draft 3, January 2007. URL `http://glueschema.forge.cnaf.infn.it/Spec/V13`.

[GLU08] GLUE Working Group, May 2008. URL `https://forge.gridforum.org/sf/projects/glue-wg`.

[GMS87] Garcia-Molina, Hector; Salem, Kenneth: Sagas. In: *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, San Francisco, California, May 27-29, 1987*, pp. 249–259. ACM Press, 1987.

[Gra78] Gray, Jim: Notes on data base operating systems. In: *Operating Systems, An Advanced Course*, volume 60 of *Lecture Notes in Computer Science*, pp. 394–481. Springer-Verlag Berlin Heidelberg New York, 1978.

[GRM05] GRMS, Mar 2005. URL `http://www.gridlab.org/`.

[GSW99] Greenberg, Albert G.; Srikant, R.; Whitt, Ward: Resource sharing for book-ahead and instantaneous-request calls. In: *IEEE/ACM Transactions on Networking*, volume 7(1):pp. 10–22, 1999.

[GT] The globus toolkit. URL `http://www.globus.org/toolkit/`.

[Gus88] Gustafson, John L.: Reevaluating amdahl's law. In: *Communications of the ACM*, volume 31(5):pp. 532–533, 1988. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/42411.42415.

[GVV+03] Gruber, Ralf; Volgers, Pieter; Vita, Alessandro De; Stengel, Massimiliano; Tran, Trach-Minh: Parameterisation to tailor commodity clusters to applications. In: *Future Generation Computer Systems*, volume 19(1):pp. 111–120, January 2003.

[Hen95] Henderson, R.L.: Job Scheduling under the Portable Batch System. In: *Proc. of the 1st Int'l Workshop on Job Scheduling Strategies for Parallel Processing, Santa Barbara, CA, USA*, volume LNCS(949):pp. 279–294, 1995.

[HML03] Huedo, Eduardo; Montero, Rubén S.; Llorente, Ignacio Martín: Experiences on grid resource selection considering resource proximity. In: Rivera, F. Fernández; Bubak, Marian; Tato, A. Gómez; Doallo, Ramon, editors, *European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pp. 1–8. Springer, 2003. ISBN 3-540-21048-2.

[Hoh07] Hoheisel, Andreas: Grid workflow execution service - dynamic and interactive execution and visualization of distributed workflows. In: *In Proceedings of CGW'06, Vol. II, CYFRONET, Cracow*, pp. 13–24. July 2007.

[ILO] ILOG CPLEX: High-performance software for mathematical programming and optimization. URL `http://www.ilog.com/products/cplex/`.

[Jav04] Sun Java, November 2004. URL `http://www.sun.com`.

[JSC01] Jackson, David; Snell, Quinn; Clement, Mark: Core algorithms of the maui scheduler. In: *Proceedings of the 7th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 01), Cambridge, MA, USA*, volume 2221 of *Lecture Notes in Computer Science*, pp. 87–102. Springer-Verlag, 2001.

[JSD] Job Submission Description Language (OGF Working Group). URL `https://forge.gridforum.org/sf/projects/jsdl-wg`.

[KLHW07] Klump, J.; Löwe, P.; Häner, R.; Wächter, J.: Continuous digital workflows for earth science research. In: *Proceedings of the German E-Science Conference (GES)*. Baden-Baden, Germany, May 2007.

[KM05] Kuo, Dean; Mckeown, Mark: Advance reservation and co-allocation protocol for grid computing. In: *First International Conference on e-Science and Grid Technologies (e-Science 2005), 5-8 December 2005, Melbourne, Australia*, pp. 164–171. IEEE Computer Society, 2005.

[KPS+07] Kottha, S.; Peter, K.; Steinke, T.; Bart, J.; Falkner, J.; Weisbecker, A.; Viezens, F.; Mohammed, Y.; Sax, U.; Hoheisel, A.; Ernst, T.; Sommerfeld, D.; Krefting, D.; Vossberg, M.: Medical image processing in MediGRID. In: *Proceedings of the German E-Science Conference (GES)*. Baden-Baden, Germany, May 2007.

[KR98] Keller, Axel; Reinefeld, Alexander: CCS Resource Management in Networked HPC Systems. In: *Proceedings of the Heterogenous Computing Workshop HCW'98 at IPPS, Orlando, FL, USA*, pp. 44–56. IEEE Computer Society Press, 1998. URL `http://www.upb.de/pc2/services/public/1998/98001.pdf`.

[KR01] Keller, Axel; Reinefeld, Alexander: Anatomy of a Resource Management System for HPC Clusters. In: *Annual Review of Scalable Computing*, volume 3(1):pp. 1–31, January 2001.

[Lam01] Lamport, Leslie: Paxos made simple. In: *ACM SIGACT News (Distributed Computing Column)*, volume 32(4):pp. 18–25, December 2001.

[LF03a] Little, Mark; Feingold, Thomas: A comparison of Web services transaction protocols. Technical Report BTP-vs-WS-Tx, IBM, October 2003. URL `http://www.ibm.com/developerworks/webservices/library/ws-comproto/`.

[LF03b] Liu, Chuang; Foster, Ian: A constraint language approach to grid resource selection. Technical Report TR-2003-07, Department of Computer Science, University of Chicago, March 2003.

[LGTW04] Li, Hui; Groep, D.; Templon, J.; Wolters, L.: Predicting job start times on clusters. In: *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pp. 301–308. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7803-8430-X.

[LHC08] LHCb Homepage, September 2008. URL `http://lhcb.web.cern.ch/lhcb/`.

[Li07] Li, Hui: Machine learning for performance predictions on space-shared computing environments. In: *International Transactions on Systems Science and Applications*, volume 3(3):pp. 257–268, October 2007.

[LKS91] Levy, Eliezer; Korth, Henry F.; Silberschatz, Abraham: An optimistic commit protocol for distributed transaction management. In: *SIGMOD '91:*

Proceedings of 1991 ACM SIGMOD international conference on Management of data, pp. 88–97. ACM, New York, NY, USA, 1991.

[LLM88] Litzkow, Michael J.; Livny, Miron; Mutka, Matt W.: Condor - A Hunter of Idle Workstations. In: *8th International Conference on Distributed Computing Systems*, pp. 104–111. IEEE Computer Society, San Jose, CA, June 1988.

[Loa08] IBM cluster software: Tivoli workload scheduler LoadLeveler, May 2008. URL `http://www-03.ibm.com/systems/clusters/software/loadleveler/index.html`.

[LSHS04] Lee, Cynthia Bailey; Schwartzman, Yael; Hardy, Jennifer; Snavely, Allan: Are user runtime estimates inherently inaccurate? In: *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, New York, NY, USA*, volume 3277 of *Lecture Notes in Computer Science*, pp. 253–263. Springer-Verlag, 2004.

[LW07] Little, Mark; Wilkinson, Andrew: Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1. Technical Report wstx-wsat-1.1-spec-errata-os, OASIS, July 2007.

[Mac07] MacLaren, Jon: HARC: The Highly-Available Resource Co-allocator. In: *Proceedings (PART II) of the On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007*, volume 4804 of *Lecture Notes in Computer Science*, pp. 1385–1402. Springer-Verlag, 2007.

[Mau04] Maui Scheduler Administrator's Guide, November 2004. URL `http://www.clusterresources.com/products/maui`.

[MDS] Globus: Monitoring and discovery system. URL `http://www.globus.org/mds/`.

[ME05] Mohamed, Hashim H.; Epema, Dick H. J.: The Design and Implementation of the KOALA Co-Allocating Grid Scheduler. In: *Proceedings of the European Grid Conference 2005, Amsterdam, The Netherlands*, volume 3470 of *Lecture Notes in Computer Science*, pp. 640–650. Springer-Verlag, February 2005.

[NG] NorduGrid. URL `http://www.nordugrid.org/`.

[NHK96] Nahrstedt, Klara; Hossain, Ashfaq; Kang, Sung-Mo: Probe-based algorithm for qos specification and adaptation. In: *Proceedings of the 4th International IFIP Workshop on Quality of Service (IWQoS96), Paris, France*, pp. 89–100. March 1996.

[NLYW05] Naik, Vijay K.; Liu, Chuang; Yang, Lingyun; Wagner, Jonathan: 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, pp. 607–614. May 2005.

[OGF] Open Grid Forum. URL `http://www.ogf.org/`.

[OSM01] Ouyang, Jinsong; Sahai, Akhil; Machiraju, Vijay: CTP: An optimistic commit protocol for conversional transactions. Technical Report HP-2001-20, Software Technology Laboratory, HP Laboratories Palo Alto, January 2001.

[PBS04] PBSPro, November 2004. URL `http://www.pbspro.com/`.

[Ram01] Raman, Rajesh: *Matchmaking Frameworks for Distributed Resource Management*. Ph.D. thesis, University of Wisconsin - Madison, Madison, 2001.

[RLS98] Raman, Rajesh; Livny, Miron; Solomon, Marvin: Matchmaking: Distributed resource management for high throughput computing. In: *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, Illinois, USA*, pp. 140–146. IEEE Computer Society Press, July 1998.

[RLS03] Raman, Rajesh; Livny, Miron; Solomon, Marvin: Policy driven heterogeneous resource co-allocation with gangmatching. In: *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pp. 80–89. IEEE Computer Society Press, Washington, DC, USA, June 2003.

[Röb08a] Röblitz, Thomas: Global Optimization for Scheduling Multiple Co-Reservations in the Grid. In: *Proceedings of the CoreGRID Symposium 2008, Las Palmas de Gran Canaria, Spain*, pp. 93–109. August 2008.

[Röb08b] Röblitz, Thomas: Specifying and processing co-reservations in the grid. In: *Proceedings of the 9th Workshop on Parallel Systems and Algorithms (PASA), Dresden, Germany*, volume 124 of *Lecture Notes in Informatics*, pp. 17–26. Köllen Druck+Verlag GmbH, Bonn, February 2008.

[RR05] Röblitz, Thomas; Reinefeld, Alexander: Co-reservation with the concept of virtual resources. In: *Proceedings of the 5th IEEE International Symposium on Cluster computing and Grid 2005 (CCGrid05), Cardiff, Wales, UK*, volume 1, pp. 398–406. IEEE Computer Society, Los Alamitos, CA, USA, May 2005.

[RR06] Röblitz, Thomas; Rzadca, Krzysztof: On the placement of reservations into job schedules. In: *12th International Euro-Par Conference 2006, Dresden, Germany*, pp. 198–210. 2006.

[RSR03] Röblitz, Thomas; Schintke, Florian; Reinefeld, Alexander: From clusters to the fabric: The job management perspective. In: *Proceedings of the IEEE Intl. Conference on Cluster Computing (Cluster'03), Hong Kong, China*, pp. 468–473. December 2003.

[RSR06] Röblitz, Thomas; Schintke, Florian; Reinefeld, Alexander: Resource reservations with fuzzy requests. In: *Concurrency and Computation: Practice and Experience*, volume 18(13):pp. 1681–1703, November 2006.

[SC92] Smarr, Larry; Catlett, Charles E.: Metacomputing. In: *Commun. ACM*, volume 35(6):pp. 44–52, 1992. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/129888.129890.

[SCJG00] Snell, Quinn; Clement, Mark; Jackson, David; Gregory, Chad: The Performance Impact of Advance Reservation Meta-Scheduling. In: *Proceedings of the 6th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '00), Cancun, Mexico*, volume 1911 of *Lecture Notes in Computer Science*, pp. 137–153. Springer-Verlag, 2000.

[SF05] Siddiqui, Mumtaz; Fahringer, Thomas: Gridarm: Askalon's grid resource management system. In: *Proceedings of European Grid Conference 2005, Amsterdam, The Netherlands*, pp. 122–131. February 2005.

[SFT98] Smith, Warren; Foster, Ian; Taylor, Valerie: Predicting application run times using historical information. In: *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 122–142. 1998.

[SFT00] Smith, Warren; Foster, Ian; Taylor, Valerie: Scheduling with Advanced Reservations. In: *Proceedings of the 14th International Symposium on Parallel and Distributed Processing, Cancun, Mexico*, pp. 127–132. IEEE Computer Society, Washington, DC, USA, May 2000.

[Sge08] SUN Grid Engine, May 2008. URL `http://gridengine.sunsource.net/`.

[SGG04] Silberschatz, Abraham; Galvin, Peter Baer; Gagne, Greg: *Operating System Concepts*. John Wiley & Sons, Inc., 7th edition, 2004.

[SM05] Sauter, Patrick; Melzer, Ingo: A Comparison of WS-Busi- nessActivity and BPEL4WS Long-Running Transactions. In: *Proceedings of the 14. ITG/GI-Fachtagung Kommunikation in Verteilten Systemen (KiVS 2005)*. February 2005.

[Spr05] Springel, Volker: The cosmological simulation code GADGET-2. In: *Monthly Notices of the Royal Astronomical Society*, volume 364(4):pp. 1105–1134, 2005.

[SR06] Stokes-Rees, Ian James: *A REST Model for High Throughput Scheduling in Computational Grids*. Ph.D. thesis, Univ. Oxford, Oxford, 2006. Presented on 28 Nov 2006.

[STF99] Smith, Warren; Taylor, Valerie; Foster, Ian: Using run-time predictions to estimate queue wait times and improve scheduler performance. In: Feitelson, Dror G.; Rudolph, Larry, editors, *Job Scheduling Strategies for Parallel Processing*, pp. 202–219. Springer Verlag, 1999.

[TDK03] Tangmunarunkit, Hongsuda; Decker, Stefan; Kesselman, Carl: Ontology-based Resource Matching in the Grid – The Grid meets the Semantic Web. In: *Proceedings of the First Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID'03)*, volume 2870 of *Lecture Notes in Computer Science*, pp. 706–721. Springer-Verlag, Budapest, Hungary, 2003.

[Tea07a] Team, GridWay: Gridway 5.2 documentation: Installation and configuration guide. Technical Report GridWay-5.2-ConfGuide, Universidad Complutense de Madrid, February 2007.

[Tea07b] Team, GridWay: Gridway 5.2 documentation: User guide. Technical Report GridWay-5.2-UserGuide, Universidad Complutense de Madrid, February 2007.

[TGGL82] Traiger, Irving L.; Gray, Jim; Galtieri, Cesare A.; Lindsay, Bruce G.: Transactions and consistency in distributed database systems. In: *ACM Transactions on Database Systems*, volume 7(3):pp. 323–342, 1982. ISSN 0362-5915. doi:http://doi.acm.org/10.1145/319732.319734.

[TNK+08] Takefusa, Atsuko; Nakada, Hidemoto; Kudoh, Tomohiro; Tanaka, Yoshio; Sekiguchi, Satoshi: GridARS: An Advance Res- ervation-Based Grid Co-allocation Framework for Distributed Computing and Network Resources. In: *Proceedings of the 13th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 07), Seattle, WA, USA, June 17, 2007. Revised Papers*, volume 4942 of *Lecture Notes in Computer Science*, pp. 152–168. Springer-Verlag, 2008.

[TOR08] TORQUE, Sep 2008. URL `http://www.clusterresources.com/pages/products/torque-resource-manager.php`.

[Uni08] UNICORE - Distributed computing and data resources, May 2008. URL `http://www.unicore.eu`.

[VBW04] Venugopal, Srikumar; Buyya, Rajkumar; Winton, Lyle: A grid service broker for scheduling distributed data-oriented applications on global grids. Technical Report GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, February 2004.

[Vmw08] VMware, May 2008. URL http://www.vmware.com/.

[Wol07] Wolf, Armin: Spezifikation der D-Grid-Beschreibungssprache D-GRDL und ihrer Nutzung im Grid-Computing. Technical report, Fraunhofer FIRST, November 2007.

[WPH07] Wieczorek, Marek; Prodan, Radu; Hoheisel, Andreas: Taxonomies of the multi-criteria grid workflow scheduling problem. Technical Report TR-0106, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, August 2007. URL http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0106.pdf.

[WV02] Weikum, Gerhard; Vossen, Gottfried: *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. ISBN 1-55860-508-8.

[WWZ05] Wäldrich, Oliver; Wieder, Philipp; Ziegler, Wolfgang: 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, pp. 782–791. September 2005.

[Xen08] The Xen virtual machine monitor, May 2008. URL http://www.cl.cam.ac.uk/research/srg/netos/xen/.

[YTA03] Yuan, Lihua; Tham, Chen-Khong; Ananda, Akkihebbal L.: A probing approach for effective distributed resource reservation. In: *Proceedings of the Second International Workshop on Quality of Service in Multiservice IP Networks, Milano, Italy*, pp. 672–688. Springer-Verlag, February 2003. ISBN 3-540-00604-4.

[ZBN+04] Zeng, Liangzhao; Benatallah, Boualem; Ngu, Anne H.H.; Dumas, Marlon; Kalagnanam, Jayant; Chang, Henry: Qos-aware middleware for web services composition. In: *IEEE Transactions on Software Engineering*, volume 30(5):pp. 311–327, 2004.

[ZMMS05] Zhao, Wenbing; Moser, L. E.; Melliar-Smith, P. M.: A res- ervation-based coordination protocol for web services. In: *Proceedings of the IEEE International Conference on Web Services (ICWS '05), Orlando, Florida, USA, July 11-15, 2005*, pp. 49–56. IEEE Computer Society, Washington, DC, USA, 2005.

[ZZWD93] Zhou, S.; Zheng, X.; Wang, J.; Delisle, P.: Utopia: A load sharing facility for large, heterogenous distributed computer systems. In: *Software – Practice & Experience*, volume 23(12):pp. 1305–1336, December 1993.

Appendix A

Glossary

Assignment: See matching.

Atomic Reservation: An atomic reservation ensures that its owner may allocate the specified resource capacity (quality-of-service) for an agreed period of time.

Atomic Reservation Candidate: An atomic reservation candidate is defined as a time-qos-slot plus a set of properties such as the reservation fee. Often, the term *reservation candidate* is used as abbreviation.

Atomic Reservation Request: An atomic reservation request defines the requirements on a single resource. Often, the term *request* is used as abbreviation.

Backfilling: Backfilling refines FCFS by allowing some requests to be executed out of order. On parallel computers, backfilling increases resource utilization. However, backfilling needs to be properly configured to avoid *starvation* of large jobs. Usually, this is achieved by only allowing backfilling if the N jobs at the head of a FCFS queue are not delayed by "out of order" jobs.

Backlog: The backlog quantifies the load of system at a specific moment in time, e.g., the current time. For parallel computers, it is defined as the sum of the remaining execution times j_{ret} and j_{eet} of the running jobs (RUN) and the waiting jobs ($WAIT$) times the allocated number of processors j_{np} divided by the total number of processors N, i.e.,

$$\left(\sum_{j \in RUN} (j_{ret}\, j_{np}) + \sum_{j \in WAIT} (j_{eet}\, j_{np}) \right) / N \,.$$

Batch Job: See non-reservation job.

Best-Effort Resource Management: A resource management is called *best-effort* if decisions on workload distribution are made on all available information (e.g., the current load of a system), but without any means to guarantee quality-of-service levels.

Book-Ahead Time: The book-ahead time of a request is the period of time between the current time and the earliest start time of a reservation.

Broker: A broker matches user requests with resource offers.

Client: See consumer.

Co-Reservation: A co-reservation contains multiple atomic reservations.

Co-Reservation Candidate: A co-reservation candidate is a combination of reservation candidates, one for each part of a co-reservation request.

Co-Reservation Request: A co-reservation request is constructed by combining multiple atomic reservation requests and specifying their relationships (temporal and spatial).

Consumer: A consumer owns some piece of work to be executed on a resource or by a set of resources. Consumers issue co-reservation requests.

Eligible Resource: See resource candidate.

Estimated Reservation Success Rate: The metric p^{esr} states the likeliness of successfully reserving a time-qos-slot. The estimation is "valid" for the current time only.

FCFS: First-Come-First-Served (FCFS) is a policy by which requests are executed in the order of their receipt.

Fitness: The metric p^{fit} states how well a time-qos-slot fits into a schedule.

Flexible Request: See moldable request.

Front-end: The front-end is a machine which hosts the resource's management system. For example, the front-end of a compute cluster hosts the scheduling system.

GT: The Globus Toolkit (GT) is a middleware for Grid computing.

GRS: See Grid Reservation Service.

Grid Reservation Service: The Grid Reservation Service (GRS) coordinates the processing of co-reservation requests.

Local Reservation Service: The Local Reservation Service (LRS) determines properties of time-qos-slots and processes reservation requests.

Local Resource Management System: A Local Resource Management System (LRMS) implements the policies of a provider for managing the workload at its resource.

LRMS: See Local Resource Management System.

LRS: See Local Reservation Service.

LWF: Least Work First is a scheduling scheme where jobs are executed in the order of their "work size". The work size is calculated as the product of the job's estimated execution time and its number of processors.

Matching: A matching assigns a request to a resource.

Metric p^{esr}: See estimated (reservation) success rate.

Metric p^{fit}: See fitness.

Moldable Request: A moldable or flexible request specifies ranges for its parameters. Thus, the reservation system may negotiate a suitable execution time window and service level with the resource providers.

Non-reservation Job: A non-reservation job is the "normal" job in a batch system. It is submitted, queued and executed without a reservation, i.e., in best-effort manner.

Normal Job: See non-reservation job.

OGF: The Open Grid Forum (OGF).

Probing: Determines the future status of resources.

Provider: A provider is an entity which offers resources.

QoS, qos, QoS-level: See Quality-of-Service.

Quality-of-Service: The quality of a service (QoS, qos, QoS-level) describes how much capacity is requested or allocated to a request. We use the term *service level* interchangeably for quality-of-service. Example service levels are the number of processors and the network bandwidth.

RC: See Resource Catalog.

Resource: A resource is a piece of hardware or service which can perform certain tasks such as executing a program, transferring data, etc.

Resource Candidate: A resource candidate satisfies all static requirements of an atomic request and vice-versa (the request satisfies the requirements of the resource).

Resource Catalog: The Resource Catalog (RC) is a directory service (also called registry) which provides information about the (mainly) static characteristics of resources.

Response Time: The response time of a request covers the period from its submission till its execution ends.

Rigid Job: See rigid request.

Rigid Request: We call a request rigid if its requirements do not allow negotiations. That is, all parameters like the start time, the service level, etc. are given without any flexibility.

Sandbox: The environment for executing a single compute job is called *sandbox*. Setting up a sandbox involves providing necessary software, setting environment variables, making input data accessible. When the job has finished, the sandbox is removed from the system.

Units: Throughout the thesis we use the following units

Unit	Description
KB	A "memory" unit for 2^{10} bytes (long: kilobyte(s))
MB	A "memory" unit for 2^{20} bytes (long: megabyte(s))
GB	A "memory" unit for 2^{30} bytes (long: gigabyte(s))
TB	A "memory" unit for 2^{40} bytes (long: terabyte(s))
PB	A "memory" unit for 2^{50} bytes (long: petabyte(s))
Mbit/s	A "throughput" unit for 2^{20} bits per second
Gbit/s	A "throughput" unit for 2^{30} bits per second

User: See consumer.

Virtual Organization: A Virtual Organization (VO) comprises users of several institutions and let them share their aggregated resources.

Waiting Time: The waiting time is the period of time that lasts from the submission (reception) of a request till it begins its execution.