TU Clausthal

Clausthal University of Technology

Distributed Decision Making for Metaschedulers

Janko Heilgeist[1], **Thomas Soddemann**[1], **and Harald Richter**[2]

IfI Technical Report Series **IfI-09-06**

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Contact: wjamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Distributed Decision Making for Metaschedulers

Janko Heilgeist[1], Thomas Soddemann[1], and Harald Richter[2]

[1] Fraunhofer Institute for Algorithms and Scientific Computing SCAI,
Schloss Birlinghoven, 53754 Sankt Augustin, Germany.

[2] Department of Informatics, Clausthal University of Technology,
Arnold-Sommerfeld-Str. 1, 38678 Clausthal-Zellerfeld, Germany.

Abstract

We propose a distributed scheduling algorithm for HPC grids and clouds that allows users, resource providers, and grid community to participate in the scheduling decision. Its hierarchical representation of dynamically configurable criteria integrates arbitrary aspects into the process in a well-defined manner; thus, the algorithm can accommodate different views and policies of geographically distributed stakeholders. The heuristic permits resource providers to exactly define the share of participation allocated to each party and to retain control over their resources. It is based on the Analytic Hierarchy Process but has been extended to support the dynamic evaluation of utility values. Evaluation respects the different perception of utility values by human decision makers with regard to their dissemination on the measure scale. The algorithm is incorporated into a distributed metascheduling architecture that is modeled around cooperating peers. Its design is resilient to network and site failures and preserves the autonomy of independent providers.

1 Introduction

Many theoretical and practical advances in distributed computing have been driven by the goal to provide users with simplified access to remote resources. Grid computing, a special form of distributed computing, has been motivated by this vision since the very beginning; in fact, the phrase "grid computing" has been coined to draw an analogy to power grids, which provide easy and ubiquitous access to power (see Foster and Kesselman [16]). It suggests that access to computing resources is as easy as plugging in an appliance and that the computing power is simply there to be used. However, the barrier to entry is a lot higher. To take advantage of distributed computing, users need to bring along a deep technical understanding of the concepts and master a variety of software tools.

Moreover, resource providers are not a homogeneous mass: politically, they compete for financing, they focus on unique user groups, they are governed by different policies, and, most importantly, they value their autonomy. Technically, they employ different

software stacks, and their computing resources target different types of applications. In order to bridge the gap, it is necessary to efficiently balance the requests across the administrative boundaries of resources and, simultaneously, address all these issues. However, the problem of optimally scheduling jobs across loosely coupled, distributed computing resources is still to be solved.

In the past, the proper groundwork for grid computing has been established by harmonizing the interfaces to the local software stacks. Grid middleware provides tools with a standardized abstraction of grid resources, e.g., via the Open Grid Services Architecture (OGSA). However, common middleware such as UNICORE[†] or Globus Toolkit[‡] offers only little support for metascheduling and requires the user to explicitly specify sites that make appropriate resources available.

Other products such as GridWay Metascheduler[§] or Platform LSF[¶] address this problem and promise to offer out-of-the-box solutions; however, on closer examination they lack the interoperability that is essential in a grid environment. For example, GridWay is specifically designed to work with the Globus Toolkit and is unable to cooperate with UNICORE; in fact, GridWay has become a Globus project. On the other hand, LSF does not adhere to open standards as defined by the Organization for the Advancement of Structured Information Standards (OASIS)[‖] or the OpenGridForum (OGF)[**]. However, the primary drawback from our point of view is that they follow a centralized design even though the the resources are distributed.

Hamscher et al. [17] illustrate that *metaschedulers*, that is, schedulers which operate on top of grid middleware to balance request between remote resources, can be designed either centralized or distributed. In a *centralized* design, the metascheduler is a dedicated software entity that is installed at a fixed site; it is responsible for the balancing of all requests in the grid. In practice, the metascheduler is often replicated for security reasons, but only a single instance is active at any time. Balance is achieved by dynamically selecting a resource for each computing job and inserting the job into the target site's queue management system. Such a queueing is transient: before the job is actually dispatched for execution on the resource, it may well be relocated to a different site at the discretion of the metascheduler. To fulfill its task, the metascheduler needs to obtain all the information that is relevant before it can make a decision to migrate the job description to another resource. Therefore, it has to collect data about all sites that participate in the grid: which type of hardware do they provide, what is the speed of their connection to the grid, which portion of the resource do they dedicate to the grid, what load their resource is currently suffering, etc.

A *distributed* or *decentralized* metascheduler shares the scheduling task between a number of independent entities. Each entity is deployed separately and manages a part of the grid. It is responsible for all the jobs entering and leaving the sites in its

[†] UNICORE, http://www.unicore.eu
[‡] The Globus Alliance, http://www.globus.org
[§] GridWay Metascheduler, http://www.gridway.org
[¶] Platform Computing, http://www.platform.com
[‖] OASIS, http://www.oasis-open.org
[**] OpenGridForum, http://www.ogf.org

portion. Here, the balance is achieved by cooperation between these entities, which we call *proxies* of the metascheduler. Other than in the centralized architecture, the proxies only have a local view of the state of the computing resources. They gather data in their assigned portion of the grid, but, to obtain more information, they need to communicate with other proxies. Total information exchange is of complexity $O(n^2)$, where n is the number of proxies in the grid, and infeasible in any but the smallest grids. Yet, Hamscher et al. [17] mention designs where the exchange is done via inter-proxy communication or by maintaining a central shared pool of computing jobs, and Wang et al. [38] propose to overlap the local job pools of adjacent sites.

Despite the fact that centralized schedulers are easier to design, to implement, to deploy, and to maintain, they have three major disadvantages. First, a centralized scheduler is a single point-of-failure for the grid. Its breakdown will affect all sites and resources in the grid and make continued operation impossible if the service is not restored immediately. The deployment of fallback installations can mitigate the disruptive effects somewhat; yet in the time until the backup solutions kicks in, access to remote resources is prevented and the grid peers are separated from each other. Even with backup installations in place, the central scheduler is an interesting target for malicious attacks on the grid. A successful hack of the metascheduler compromises the whole grid at once and may allow sensitive customer data to be stolen.

Second, in a centralized architecture the metascheduler represents a bottleneck for the grid. By design, it is solely responsible for the handling of all jobs that are not computed locally on a resource. Hence, the scalability of a centralized metascheduler is of primary importance as the number of users and jobs increases. Further, with a growing number of grid resources the costs in computational power and network bandwidth rise because the central entity will have to collect and maintain up-to-date status information on all of them. Commonly, the countermeasure is to deploy multiple backup servers and spread the load evenly between them. Yet, again this will increase the costs as additional servers need to be acquired and maintained. Moreover, the boundaries between centralized and distributed designs begin to blur.

However, the key issue with a centralized architecture is not technical but stems from the political differences between resource providers. The intent of grids is to have them extend over multiple administrative zones that include companies, institutions, countries, and continents. Generally, these parties pursue opposing, or at least different, interests with respect to their participation in the grid. Industrial providers focus on financial aspects; resources controlled by academic institutions target scientific advances and research. Yet even with similar goals, the exact policies applied by a party may differ significantly. From our experience, a centralized metascheduler will have to find ways to accommodate a plethora of criteria, and, still, it will meet resistance when local administrators are supposed to relinquish control over their hardware to it.

Therefore, we favor a distributed approach, where each site deploys its own metascheduler proxy. This proxy manages all the grid jobs at its home site; thus, it can consider and observe local restrictions and policies. As each site configures its proxy separately, the political issues are largely nonexistent. A distributed design is more resilient to site and link failures, but if a proxy does break down it will only disconnect

its home site from the grid; other independent proxies are unaffected by such a down time. Finally, the scalability of a distributed design is positively improved when compared to a centralized architecture. Since every site manages itself and computes its own schedule, the load is spread evenly across all grid peers, and the performance does not depend on a single entity.

However, the lack of overview and the fact that general scheduling is computationally hard mean that, in general, generated schedules will only be an approximation of the optimal scheduling. It is the task of the decision making algorithm of a metascheduler to select a series of migrations that lead towards an acceptable schedule. In operations research (OR), decision making means the process of selecting the most qualified option from a given set of alternatives. The qualification or utility of an option has to be formally specified beforehand. It depends on the specific situation at hand and is, therefore, determined on a case-by-case basis. In any realistic scenario, the utility will depend on multiple distinct criteria. Hence, the domain of operations research that deals with these problems is also called *multicriteria decision making* (MCDM).

The criteria applied in a particular decision making process correlate with the alternatives and the optimization goal. It is implicitly assumed, that the given alternatives are mutually comparable when evaluated by the chosen criteria. Thus, the view on the alternatives is restricted to certain aspects that have been determined as relevant to the decision. As an example, it might be reasonable to compare the proverbial apples and oranges if the criteria are, e.g., nutritional value, vitamins, and sweetness.

Usually, the set of criteria is not sufficient to deterministically rank the options by their utility. During the evaluation, the algorithm can arrive at a set of so-called *Pareto-optimal*[†] alternatives: the *Pareto set*. An option is Pareto-optimal if further improvement can only be achieved by accepting a trade-off. That is, improving the utility with regard to any arbitrary subset of the criteria will simultaneously decrease the utility with regard to at least one other criterion. Fig. 1 illustrates the problem for an exemplary 2-dimensional minimization problem.

The tie between Pareto-optimal alternatives can only be broken by external means. That is, the decision making algorithm needs supplemental data besides the criteria and the alternative's utilities to actually select an element of the set. The additional information is commonly supplied by the user and describes preferences between the relevant criteria. Now, the algorithm can utilize this data to determine which utility values to improve and where to accept a deterioration.

The preference can be expressed, e.g., as a simple ranking of the criteria or more detailed by assigning explicit weights or priorities. We will further discuss both methods when we describe the algorithm used by the metascheduler.

In the remainder of this technical report, we will present our approach to metascheduling. Previously, we have presented the design of the resource discovery algorithms in Heilgeist et al. [18], and we will therefore focus on the distributed scheduling in this report. We start with an overview of traditional scheduling concepts in Section 2 and proceed to illustrate existing multicriteria decision making algorithms that are rel-

[†] Vilfredo Pareto (1848–1923); economist and sociologist.

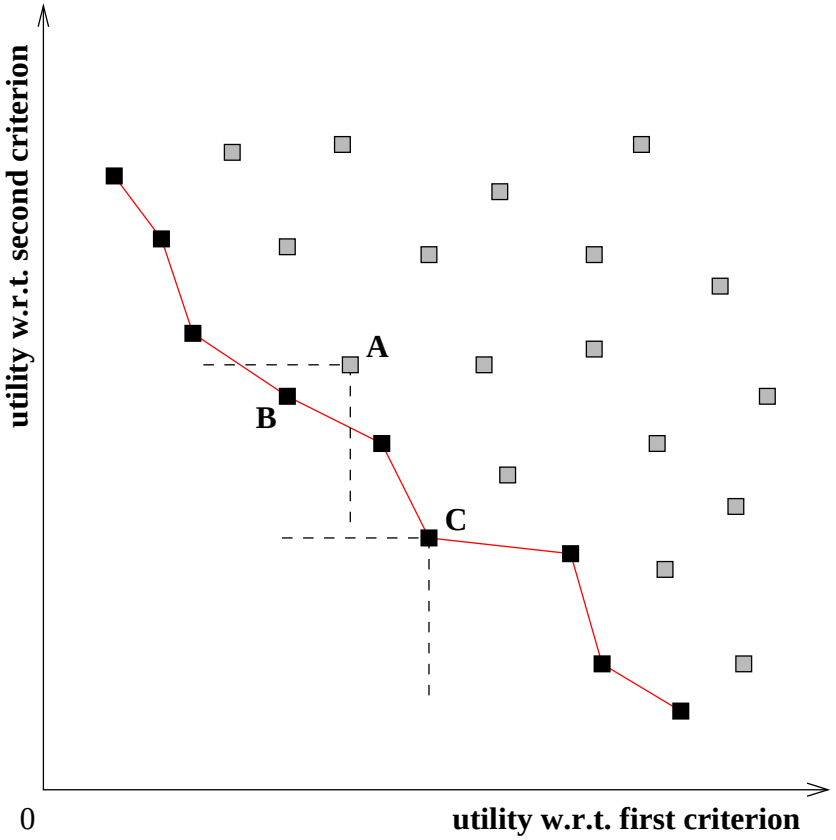


Figure 1: Pareto set of a 2-dimensional minimization problem. — Each option is depicted by a little square. The x-axis and y-axis measure the utility of an option with regard to two independent criteria. Due to minimization, the optimization is directed towards the origin. A solid line connects the options of the Pareto set. Note, that option A is strictly dominated by option B and, therefore, not Pareto-optimal. On the other hand, option C is Pareto-optimal.

evant to our idea in Section 3. Then, we describe how scheduling and decision making are related in Section 4 and explain the general steps to build a schedule from individual decisions. We propose a specific hierarchical heuristic for distributed scheduling in Section 5 and present details on the practical aspects in Section 6. Finally, we conclude the report with conclusions and an outlook in Section 7.

2 Scheduling Concepts

2.1 Portable Batch System

The Portable Batch System (PBS) [10, 19] is a queue based batch job and resource management software. Although it has been superseded by its descendants TORQUE Resource Manager[†], OpenPBS[‡], and PBSPro[‡], the original design is still used in these products. PBS consists of three major components: a single *job server*, one or more *job schedulers*, and one or more *job executors*. These three components are also called `pbs_server`, `pbs_sched`, and `pbs_mom`, respectively, after their processes. Further, PBS provides a set of commands broken up into user commands, operator commands, and administrator commands. User commands generally operate on the jobs, while operator and administrator commands manipulate and monitor the batch system. The latter two command sets require elevated privileges.

The job server is the controlling instance of the installation. It manages job queues, accepts or rejects jobs, and places them into the queues. Further, it maintains an accounting log. All commands interact directly with the server. Management of the queues requires administrational privileges. Besides creating and deleting queues, an administrator can enable or disable queues; this controls whether new jobs can enter a queue. In addition, queues can be started or stopped which controls whether jobs in a queue are supposed to be executed or not. Every queue can have constraints assigned to it. Constraints are expressed with regard to the requirements of the jobs they may contain. The job server enforces these constraints when a job requests to be appended to a queue.

Queues are divided into *routing queues* and *executing queues*. Routing queues are used to automatically sort jobs into a set of associated execution queues. They analyze the requirements of a job and match them to the constraints of the associated execution queues. The job is routed into the first queue which has sufficiently lax constraints to accept it. In contrast, execution queues hold the input for the job scheduler. They contain the jobs that are eligible for execution. The manual notes, that the job server does not actually enforce any of these limits. It will execute jobs out of order, jobs from stopped queues, etc. if directed by the job scheduler or any other client with administrative privileges.

The execution and monitoring is controlled by the job executors. Each node, that is, each compute resource with a single operating system image, is being managed

[†]Cluster Resources, `http://www.clusterresources.com`
[‡]Altair Engineering, `http://www.pbsgridworks.com`

by a separate job executor. They listen on a privileged port and accept commands from the job server or other legitimate clients. Upon receiving a job, they provide a shell environment as similar as possible to the user's login shell. A prologue script is executed before the job, that can be used for additional preparation of the environment and stage-in of the input files. Afterwards, the job is forked in a separate child process. Hence, the executor is also called MOM or `pbs_mom` because it is the mother of all jobs running on its node.

The job executor continuously monitors its node and the states of the running jobs. Eventually, jobs either finish their computations, abort with an error, or exceed their allotted runtime; in the latter case, the job is killed by the job executor. Every run of a computing job is followed by the execution of an epilogue script. This script allows an administrator to clean up the environment and, in preparation for the next job, return it to a stable state. Further, it performs the stage-out phase, that is, the epilogue script can be used to copy the results of the job into the user's home directory.

The scheduling is performed by one or more job schedulers. Such a process is considered a normal client with administration privileges. It runs in separate process and is regularly contacted by the job server to select jobs for execution. The job server initiates a scheduling cycle if a new job becomes eligible for execution, a running job finishes, an administrator forces a cycle, and periodically after a configurable time interval. The scheduler queries the resource monitor parts of the MOMs to obtain the current state of the resource. Further, it has access to the job server queues and can select one or more jobs to be started immediately.

Due to the modular concept of PBS, the scheduler is easily replaceable. In fact, the manual notes that the default scheduler "is intended to be a jumping off point for a real scheduler to be written" [10, page 32]. PBS provides several APIs to simplify the implementation of the scheduling logic. The Batch Scheduling Language (BaSL) is an enriched C dialect with explicit support for notions such as server, queue, and job. Further, the scheduler can be developed in Tcl, pure C, or a third-party Java API that is described by Nakada et al. [25].

The default scheduler is rule-based. In every scheduling cycle, it first sorts the queues by their configurable priority and subsequently schedules each queue in decreasing order. The jobs in a queue are sorted by the CPU time they request. The scheduler places the shortest job first and only continues to the next queue if it has scheduled all jobs in the current queue. Strong preference is given to jobs that have been waiting for more than 24 hours as they are considered starving.

The default scheduler can be adjusted with a wide array of configuration parameters. Priorities can be freely assigned to the queues and the job sort order can be defined based on, e.g., requested CPU time, requested memory, job priority, walltime, or fair sharing. Different rules can be specified for either holidays or workdays. For workdays the time may be divided into intervals of either prime time or non-prime time. Again, each type of interval can be associated with a unique set of scheduling rules. Finally, specific queues may be active only during particular frames of time; hence, an administrator can dedicate a recurring slice of the resource to, e.g., selected projects or user groups.

However, the TORQUE administrator manual warns that "the default TORQUE

scheduler, `pbs_sched`, is very basic and will provide poor utilization of your cluster's resources. Other options, such as Maui Scheduler or Moab Workload Manager are highly recommended" [12, appendix L.9]. In practice, TORQUE is therefore rarely used on its own. E.g., in the German D-Grid[†] it is deployed in combination with the Maui Cluster Scheduler (see Section 2.2).

2.2 Maui Cluster Scheduler

The Maui Cluster Scheduler[‡] is an open source job scheduler that is deployed in concert with third-party batch systems such as PBS, LoadLeveler[§], Sun GridEngine[¶], or LSF. It uses the capabilities of the batch systems to execute and monitor jobs while taking over the scheduling responsibility. Maui provides the administrator of a resource with extensive configuration options to control the scheduling process. These options and the actual algorithms employed by Maui have been documented by Jackson et al. [21].

Maui features a number of improvements over basic queue-based scheduling. Its algorithm schedules jobs enqueued in the batch system in order of dynamic priority values. Thus, job priorities provide an alternative to a rigid first-come, first-served (FCFS) execution order. Further, Maui allows some leeway in its order of execution to improve the utilization of a resource and reduce the average job turnaround time. This style of batch handling is called *backfill scheduling*. Finally, Maui offers support for *advance reservations*, that is, parts of a resource can be booked in advance. Hence, execution of a job can be planned ahead and the necessary resources preempted.

The scheduling process is carried out in iterative cycles. Similar to PBS, each cycle is triggered by an event such as job or resource state change, start or end of a reservation, an external command, or, periodically, upon expiration of a timer. In each cycle, a fixed sequence of steps is executed. First, Maui honors previously registered advance reservations. Jobs associated with a reservation are dispatched immediately if applicable. Afterwards, the scheduler builds a list of all jobs eligible for execution and prioritizes the list. Stepping through the list in order of decreasing priority, it schedules a configurable number of jobs. These jobs are either executed immediately, if sufficient resources are free, or an advance reservation is made for the earliest point of availability. Finally, two backfilling runs are performed over the remaining jobs in the prioritized queue. In the first pass, vacancies in the schedule are filled with jobs that satisfy a set of strict throttling policies. In the second pass, a set of more broad policies is used to select candidate jobs.

Maui determines eligibility and priority of a job based on various static and dynamic attributes. These include, e.g., job credentials, quality of service (QoS) levels, fairshare records, and scheduling-related runtime properties. The job credentials encompass static identifiers such as the the job owner's user account, the group he belongs to,

[†] D-Grid Initiative, http://www.d-grid.de
[‡] Cluster Resources, http://www.clusterresources.com
[§] IBM Corp., http://www.ibm.com
[¶] Sun Microsystems, http://www.sun.com/software/sge/

or the project in which he participates. Further, each job is associated with a job class — an equivalent to the job queue found in other batch scheduling systems.

Subject to its credentials, a job can request individual QoS levels. An improved quality level can incorporate, e.g., access to specialized hardware or software licenses, preferred treatment by the scheduling algorithm, or exemption from certain restrictions. Therefore, QoS levels are restricted and their usage is limited by access control lists (ACL). A request for enhanced service levels is automatically granted or denied based on the credentials.

Finally, fairshare targets further control the standards provided to a job. Although interpretations of fairshare differ, they generally "involve a mechanism which controls the distribution of delivered resources across various job attribute-based dimensions" [21, page 97]. As the name suggests, the idea is to guarantee a fair share of the resource to each user. Maui achieves this goal by collecting and storing utilization records on a per-credential basis. For each job executed on a resource, its requirements in processors, memory, disk space and swap space are expressed in *processor equivalents* (PE). The amount of PEs consumed by a job is credited against its associated user, group, and project accounts. Then, Maui adjusts priority and eligibility values based on these historical records to permit a balanced access to the scheduled resource.

The priority of a job is determined as a weighted sum of various measures. The weights in this sum are specified by the administrator of a resource. They belong to the basic configuration of Maui and are usually constant once they are established. Other factors are based on a job's associated attributes or dynamic measures. The exact formula for the priority of a computing job J is

$$\begin{aligned}\text{jobPriority}_J := {} & \text{serviceWeight} \cdot \text{serviceFactor}_J \\ & + \text{resourceWeight} \cdot \text{resourceFactor}_J \\ & + \text{fairShareWeight} \cdot \text{fairShareFactor}_J \\ & + \text{directSpecWeight} \cdot \text{directSpecFactor}_J \\ & + \text{targetWeight} \cdot \text{targetFactor}_J \\ & + \text{bypassWeight} \cdot \text{bypassFactor}_J,\end{aligned}$$

where an index of J denotes a value that depends on the specific job. In the following, we will explain the motivation behind $\text{directSpecFactor}_J$ and targetFactor_J. For further details regarding the remaining factors, we refer the reader to Jackson et al. [21].

The direct priority specification factor $\text{directSpecFactor}_J$ is used to explicitly control the priority of a job based on political directives. Its value is specified as

$$\begin{aligned}\text{directSpecFactor}_J := {} & \text{userWeight} \cdot \text{priority}(\text{user}_J) \\ & + \text{groupWeight} \cdot \text{priority}(\text{group}_J) \\ & + \text{accountWeight} \cdot \text{priority}(\text{account}_J) \\ & + \text{qosWeight} \cdot \text{priority}(\text{qos}_J) \\ & + \text{classWeight} \cdot \text{priority}(\text{class}_J),\end{aligned}$$

where priority is an operation that returns the configured priority value of its argument. As can be seen, the factor is itself a weighted sum that combines weights and priorities. Weights reflect the relative importance that is given to a particular piece of job credential. Again, they are configurable but, commonly, established only once. Their value is independent of a job and applies to all jobs in the batch queue. Priorities, on the other hand, are related to job-specific attributes and thus more dynamic. While the priority of a user, group, or project account, a QoS level, or a job class is just as fixed, each job does carry its own set of credentials. Therefore, these attributes can be used to distinguish job with respect to their owners. Customarily, such a method of distinction is required for political reasons. Preferred access to a resource can thus be granted based on aspects such as available funding, scientific relevance, or public visibility of a party.

The target factor targetFactor_J is designed to allow the definition of and adherence to policies regarding the queue waiting time and expansion factor of a job. It is defined as

$$\begin{aligned}\text{targetFactor}_J := \\ &\max(0.0001, \text{xfTarget} - \text{expansionFactor}_J)^{-2} \\ &\quad + \max(0.0001, \text{qtTarget} - \text{queueTime}_J)^{-2},\end{aligned}$$

where xfTarget and qtTarget are the configurable targets for the job-specific expansion factor and queue waiting times. The expansion factor of a job J is determined as

$$\text{expansionFactor}_J := \frac{\text{queueTime}_J + \text{jobRuntime}_J}{\text{jobRuntime}_J},$$

that is, it sets the waiting time of a job in relation to its requested runtime and is a unitless ratio. Queue time target and the actual queue waiting time are specified in minutes. The intent behind the target factor is to allow an administrator to define objectives for the time that a job spends in the batch queue. The priority of a job is then adjusted to steer scheduling towards adherence of these policies.

The backfill scheduling algorithm orders the job queue in order of decreasing job priority. Subsequently, it iterates through the queue and dispatches jobs that can be started immediately. For jobs whose requirements can not be satisfied right away, it creates advance reservations as early as possible. Up to this step, the backfill algorithm is identical to a regular queue-based scheduling approach. Its distinction arrives from the fact, that it fills vacancies in the schedule with low-priority jobs. Thus, it is able to improve the utilization of the resource simultaneously reducing average job turnaround times. In a sense, the preferred execution of the filler jobs does violate the priority determined previously. But, jobs are only brought forward if their execution does not delay any reservation of a higher prioritized job.

High priorities signal a preferred treatment in the planning phase of the scheduling algorithm. However, they do not guarantee a preferred access to the resource. While backfilling takes care not to violate advance reservations, a so-called *pseudo-delay* [21, Section 4.1] can occur. Pseudo-delays cause a high priority job to be started later than it

would be without backfilling. The effect is caused by the inaccuracies of user's runtime estimations about their jobs. These estimates are used by the scheduler to anticipate the end of a job's execution and determine the start of a subsequent advance reservation. If such an estimate proofs to be wrong and the corresponding job aborts prematurely, a low-priority job may already have been backfilled and occupy the resource. A subsequent advance reservation is thus prevented from accessing the resource. But, simulations by Jackson et al. [21] with real-life workload traces showed that only about ten percent of the jobs are actually stalled. On the other hand, backfilling can increase the utilization of a resource by about 20 percent and up to 90 percent of small and short-running jobs are backfilled.

Various aspects of Maui's backfill algorithm can be easily customized. First, the reservation depth, that is, the number of jobs that are turned into advance reservations before backfilling starts, can be configured. An administrator can opt to create only a few reservations, thus, leaving a large degree of freedom to the backfilling algorithm. In general, this leads to better utilization of the resource. Alternatively, he can choose to create a large number of reservations and, thereby, emphasize the importance of a job's priority. Maui defaults to a single advance reservation.

Second, the order in which Maui resolves overlapping vacancies is adaptable. Vacancies can be either wide or long; that is, either many CPUs are available for a short time, or a few CPUs are idle for a long time. If vacancies overlap, they can be filled either widest or longest window first. Maui's default setting is to fill the widest vacancy first.

Finally, the scheduling algorithm can be adjusted in the way in which it selects job to fill vacancies. The default is to use the first job, that fits into the window; a strategy that is called *first-fit*. Alternatively, a *best-fit* approach can be taken. Here, a job is select that best fits into the window with regard to, e.g., seconds, processors, or processor-seconds. However, Jackson et al. [21] state that, based on their experience, there is no practical difference between these options.

2.3 Service Level Agreements

For users of a compute grid, it is desirable to obtain guarantees regarding the quality of a service provided to them. Such a guarantee can be conveyed by means of a Service Level Agreement (SLA). An SLA is an electronic contract that is automatically negotiated between service consumer and service provider. The Web Services Agreement Specification (WS-Agreement) [2] defines a standardized agreement structure for SLAs. According to WS-Agreement, a SLA is characterized by its name, its context and its terms.

The terms of an agreement describe the service to be provided under the agreement (*service terms*) and assurances regarding the quality of the service (*guarantee terms*). In the domain of scheduling, the service terms may be expressed by means of notions such as the type of hardware, the number of allocated nodes, and the job dispatch time. The guarantee terms specify in an unambiguous way the quality of service to be delivered

Term	Definition	
T_S	earliest job start time	
T_F	latest job finish time	
t_D	reserved job runtime	
N_CPU	number of reserved CPU nodes	
A	job size	$A = t_\mathrm{D} N_\mathrm{CPU}$
t_T	deadline tightness	$t_\mathrm{T} = t_\mathrm{D}/(T_\mathrm{F} - T_\mathrm{S})$
t_L	job laxity	$t_\mathrm{L} = (T_\mathrm{F} - T_\mathrm{S}) - t_\mathrm{D}$

Table 1: Set of guarantee terms used to heuristically determine the priority of a Service Level Agreement [35, 40].

by the supplier. They can be phrased in terms of constant values or bounds that denote legal values with regard to the specific service aspect.

Sakellariou and Yarmolenko present an exemplary grid architecture based on this concept in [35]. In this three-party design, the negotiations are conducted between users, brokers, and local schedulers. The brokers serve as mediators between users and local schedulers. They negotiate with one or more local schedulers to accumulate the resources required to fulfill their previously established agreement with a user. The local schedulers need to be SLA-aware to be able to participate in the negotiation process and, subsequently, honor the terms of an agreement. Further, the terms of an agreement might need to be renegotiated occasionally. This can happen if, e.g., the availability of a resource changes due to influences beyond the control of a provider.

With Service Level Agreements, actual scheduling on the grid level is replaced by the bargaining of the brokers. Instead, the local scheduling needs to be adjusted such that it supports SLAs. An approach to scheduling with agreements based on heuristics is described in [35, 40]. Here, a priority is determined for each SLA that is heuristically related to the agreement's guarantee terms. The algorithm then schedules the SLAs in order of their priority. An agreement is scheduled by reserving the requested CPUs at the earliest possible time that honors the SLA without violating previously scheduled agreements. Hence, an affinity to backfill scheduling is recognizable.

An agreements priority is defined based on a set of guarantee terms

$$\{T_\mathrm{S}, T_\mathrm{F}, t_\mathrm{D}, N_\mathrm{CPU}, A, t_\mathrm{T}, t_\mathrm{L}\}$$

(Table 1). The earliest job start time T_S and the latest job finish time T_F describe the limits between which the execution of the job has to be fully contained. The resources allocated to the job are given by the reserved job runtime t_D and the number of reserved CPU nodes N_CPU. Generally, these values will be closely related to the requirements of the compute job. Fig. 2 shows the relationship between these guarantee terms.

Further, the remaining terms are expressed with reference to the previous definitions. The job size A is characterized as the number of CPU-hours consumed by a job, that is,

$$A := t_\mathrm{D} N_\mathrm{CPU}.$$

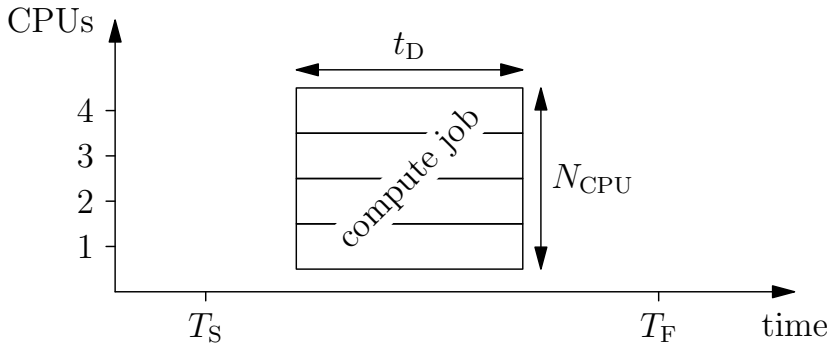


Figure 2: Basic guarantee terms for use in Service Level Agreements. — The displayed guarantee terms are: earliest job start time T_S, latest job finish time T_F, reserved job runtime t_D, and number of reserved CPU nodes N_{CPU}. Note, that the x-axis shows an arbitrary interval of time.

It denotes the actual amount of compute power consumed. Thus, jobs can be compared whether they use a few CPUs for an extended period of time or a large number of CPUs only shortly. Deadline tightness t_T and job laxity t_L are related insofar as they both provide measures for the amount of leeway given to the scheduling of the job. The deadline tightness is defined as

$$t_T := \frac{t_D}{T_F - T_S}$$

and presents the allocated runtime as a fraction of the size of the available execution interval. The job laxity

$$t_L := (T_F - T_S) - t_D$$

quantifies the actual amount of tolerance given by this interval.

Finally, the priority of an SLA is defined as

$$H := h_1 + wh_2$$

whereby h_1 and h_2 are picked from the set of guarantee terms in Table 1 and $w \in \mathbb{R}$ is a weighing coefficient. Yarmolenko and Sakellariou show that the quality of the generated schedule can vary significantly depending on various criteria. These criteria include, e.g., the pair of selected terms, the value and sign of w, and the definition of the performance metric. Furthermore, the performance strongly depends on whether the SLAs are scheduled in order of increasing or decreasing priority.

Using parameter-sweep simulations, they show that the priority $H_1 = T_F + w_1 A + w_2 t_L$ results in a higher percentage of serviced SLAs when $w_1 = 0.03$ and $w_2 = 0.14$ than if either coefficient is set to 0.0. Likewise, the utilization of the CPUs is maximized for $w_1 = -0.06$ and $w_2 = -0.6$. This configuration clearly outperforms the best

corresponding single-parameters variants where either $w_1 = -0.135$ or $w_2 = -0.78$ while the other coefficient is set to 0.0. They draw the conclusion that increasing the number of parameters can positively influence the performance.

However, these priorities are opaque and without apparent coherence. Measures such as T_S (minutes), A (CPU hours), and t_T (without unit) are added into a combined value. It is impossible to exactly predict the effect a metric will have on the produced schedule. The results just hint at a correlation between the selected metrics and the resulting profit. For example, a simulation with the priority $H_2 = \min\left(T_\mathrm{F} - 7.2t_\mathrm{D}\right)$ yielded the best CPU utilization while two simulations with $H_3 = \min\left(T_\mathrm{F} + 0.24N_\mathrm{CPU}\right)$ and $H_4 = \min\left(T_\mathrm{F} + 0.03A\right)$ resulted in the highest percentage of honored SLAs. Thus, scheduling long running jobs first seems to lead to improved CPU utilization. Similarly, penalizing jobs with large requirements apparently improves the percentage of honored SLAs. Regardless, these results are not explained in any way.

Yarmolenko and Sakellariou [40] discovered the best metrics by performing parameter-sweep simulations with different weights; then, they analyzed the achieved profit with respect to their test environment. Without simulation, there is no way to predict the weights that will yield the best results.

3 Multicriteria Decision Making

Multicriteria Decision Making (MCDM) has been a topic of research in the economic sciences for many years. Here, the primary goal is to provide managers with the tools to support economic decisions based on various aspects. In general, the methods suggested allow a set of distinct alternatives or options to be evaluated with respect to another set of arbitrary, but fixed, criteria. It is common for the algorithms to construct a utility function; that is, a function which maps an alternative to a numeric value that represents its utility with respect to the criteria. A manager is, generally, queried to express his preference for a single option or sets of options under certain aspects. Small judgements are used to derive larger preference statements. Sometimes, the methods are able to merge the judgements provided by different decision makers into an overall utility function.

We define a MCDM problem as a tuple (Ω, Γ, w), where $\Omega = \{O_1, O_2, \ldots, O_m\}$ is the set of options, $\Gamma = \{C_1, C_2, \ldots, C_n\}$ is the set of criteria, and $w\colon \Gamma \to \mathbb{R}_{\geq 0}$ is a *weighting* of the criteria. The weighting is a function mapping a criterion to its relative importance in the overall decision. In the following sections, the weighting function is generally represented as a weight vector $w = (w_j)_{1 \leq j \leq n} \in \mathbb{R}^n_{\geq 0}$. The jth entry in the weight vector is then associated with the jth criterion, that is,

$$w\colon \Gamma \to \mathbb{R}_{\geq 0} \quad :=\quad C_j \mapsto w_j.$$

Then, the solution to a decision making problem is an algorithm $\mathcal{A}$ that takes a MCDM problem and constructs a mapping from options to utility values. Formally,

$$\mathcal{A}\colon (\Omega, \Gamma, w) \mapsto u \in \mathbb{R}^\Omega_{\geq 0}$$

where u is a function that maps each option to its associated utility value with respect to the given criteria and their weights. In the following, we will describe a selection of existing MCDM algorithms that fit into this formal definition.

A major component of each of the following algorithms is a procedure to elicit judgements from a decision maker. He is presented with a set of items and asked to express his preferences with respect to a given frame of reference. In various ways, these judgements are aggregated into an overall set of relative preference values. It is this routine, that is, e.g., used to derive the weight vector resp. weighting function w. Here, the set of items corresponds to the set of criteria and the frame of reference is the notion of "relative importance with respect to the overall decision making goal". But, the routine can also be employed to compare the set of alternatives with respect to a fixed criterion C_j and, analogously, construct a partial utility vector resp. utility mapping

$$u_{C_j} := \mathcal{A}\big(\Omega, \{C_j\}, e_j\big),$$

where e_j is the jth unit vector. The utility vector represents a partial solution to the decision making problem that ignores all but one criterion. Next, the solution to the overall problem is assembled by joining the partial solutions, that is,

$$\mathcal{A}(\Omega, \Gamma, w) := F(w_1, w_2, \ldots, w_n; u_{C_1}, u_{C_2}, \ldots, u_{C_n}) \tag{1}$$

for some aggregation function F.

In the following sections we will give an overview of existing MCDM algorithms that constitute the basis for our approach to metascheduling. First, the general representation of a set of criteria as a hierarchy is described in Section 3.1. In Section 3.2 a short introduction into a well-known application of this principle is given: the Analytic Hierarchy Process (AHP) by Saaty. The AHP is not without its critics, which will be pointed out in Section 3.3. Another variant of the AHP, the Multiplicative AHP (MAHP) by Barzilai et al., addresses some of the criticisms of the original AHP; its ideas and motivation will be detailed in Section 3.4. Finally, two additional methods to compute the weights used in conjunction with the AHP will be explained in Section 3.5: the point of centralized weights by Solymosi and Dombi and the method of Rank-order Centroid (ROC) by Barron and Barrett.

3.1 Hierarchy of Criteria

The strength of MCDM algorithms hinges on the ability of a decision maker to clearly express his intentions. He is required to explicitly state all the aspects of the decision that are relevant to him, and he must be able to precisely specify the relative importance of each aspect. Otherwise, the results of the decision making process can be dissatisfying. Unfortunately, the construction of a well-defined MCDM problem can be a daunting task. In real-life situations, decisions are customarily reached either instinctively or based on a coarse set of superficial criteria. Situations are rarely analyzed in the granularity and depth required by the following algorithms. A solution to this problem was first suggested by Miller [24] in his PhD thesis and, later, popularized by Saaty [28] in conjunction with his Analytic Hierarchy Process (AHP).

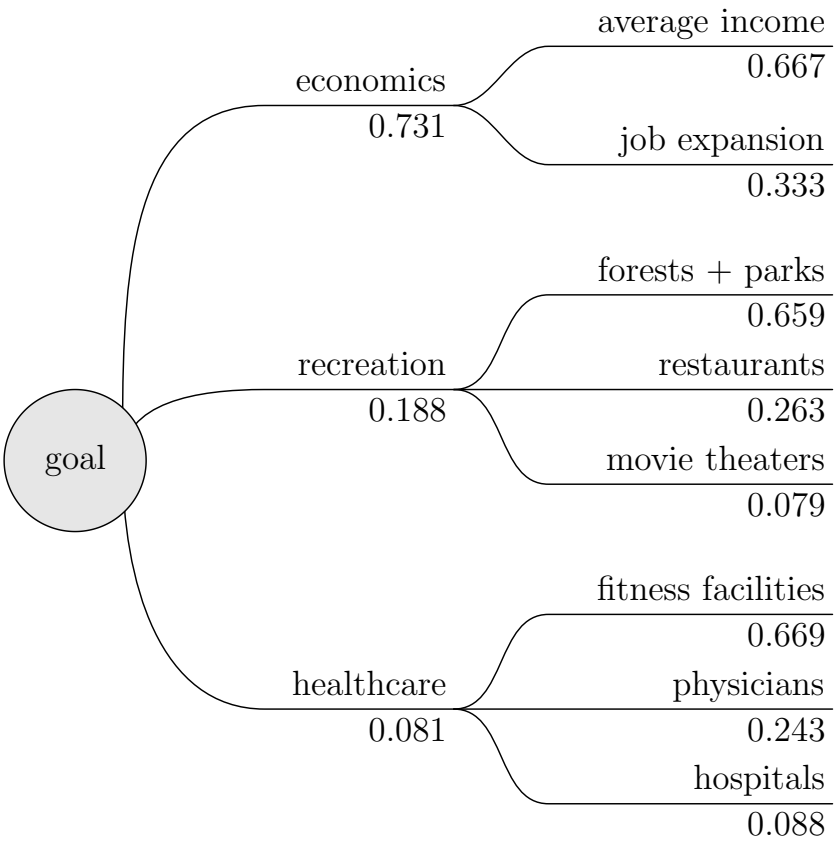


Figure 3: Exemplary criterion hierarchy in node-link representation.

Miller proposed a hierarchical design, that can considerably simplify the decomposition of a problem into independent criteria. Starting with the overall goal, each criterion is iteratively broken up into more specific subcriteria. Hinting at the tree structure that is recursively constructed, we denote these subcriteria as *child criteria*. Each child criterion describes a facet of its *father criterion*; the other way around, a criterion is fully defined by the total of its children. The decomposition process is stopped when an adequate level of detail has been reached. Like all trees, the resulting structure can be visualized as a graph, e.g., in node-link or treemap representation.

EXAMPLE 1. Saaty [29] applies the AHP to a set of criteria and numerical scores taken from an almanac to determine "the most livable cities in the United States". The resulting hierarchy is to complex to display in detail, but Fig. 3 shows an example based on his ideas.

At the root of the tree, the goal corresponds to the problem of determining how liv-

able a city is. Three criteria have been chosen as relevant with regard to this question: the economical situation, the recreational opportunities, and the available health care services. The economical situation is further decomposed into the average household income attainable and the job expansion. Recreational facilities of interest to the decision maker are national forests and parks, good restaurants, and a selection of modern movie theaters. Finally, the ability to exercise a personal fitness program, the number of physicians per 100,000 residents, and the quality of hospitals have been determined as relevant health care services.

The original example by Saaty [29] contained additional facets, namely climate, housing, crime, transportation, education, and arts, as well as a plethora of child criteria. While he provided the weights for the top-level criteria, the weights of subcriteria further down the hierarchy were not listed. For this reason, we weighted all of the criteria in this example ourselves.

The hierarchical decomposition of the overall decision problem provides two distinct advantages to the decision maker. First, the problem is separated into smaller chunks which can be evaluated separately. Using the notation of the previous section, this relationship between a father criterion and its children can be expressed via their utility functions. Let C be an arbitrary criterion, and let $C_1, C_2, \ldots, C_n$ be the set of subcriteria in which C has been decomposed. Further, let w_j be the weight of criterion C_j, for $1 \leq j \leq n$. The utility function with respect to the father criterion is then defined as

$$u_C = F(w_1, w_2, \ldots, w_n; u_{C_1}, u_{C_2}, \ldots, u_{C_n}),$$

where F is some aggregation function; commonly, but not necessarily, F is structurally related to the aggregation function in Eq. (1). Examples for such aggregation functions are, e.g., the weighted arithmetic mean employed by the traditional AHP algorithm

$$u_C(O_i) := \sum_{j=1}^{m} w_j u_{C_j}(O_i) \tag{2}$$

and the weighted geometric mean used by the Multiplicative AHP variant

$$u_C(O_i) := \prod_{j=1}^{m} u_{C_j}(O_i)^{w_j}. \tag{3}$$

Note, that in both cases the weights are assumed to be normalized, that is, $\sum_{j=1}^{m} w_j = 1$.

Thus, a criterion's utility function is wholly defined by its children's weights and utility functions. In other words, it is sufficient for a user to provide three components in addition to the options:

1. the hierarchy of criteria,
2. the relative weight of each criterion, and

3. the utility functions for each *leaf* criterion.

The criterion tree can then be evaluated recursively from the leaves up to the root which, by design, corresponds to the overall goal of the decision problem. Each intermediate utility function, that is, each utility function with respect to an inner node of the tree, is derived automatically.

The second advantage of hierarchical decomposition is the reduction of criteria that need to be compared amongst each other. During the elicitation of the weight function, the decision maker is required to state precise weights for each criterion. A weight expresses the relative importance with respect to the overall decision of the criterion it is associated with. The presence of a large number of different facets can make this a demanding requirement to fulfill; even more so, if the elicitation procedure supports only a limited scale to designate the differences. Consequently, subtle variations in preference can easily be overlooked in the process. Here, the decomposition can help by narrowing the focus to one subset of criteria at a time.

Note, that the idea of a criterion hierarchy does still fit into the definition of a MCDM problem given in the previous section. The recursive application of an aggregation function can generally be eliminated by flattening the hierarchy to a single level. In the process, inner nodes of the tree are recursively replaced by their child nodes. Still, the weights of the children need to be adjusted with respect to the replaced father criterion. Let $(r_{j1}, r_{j2}, \ldots, r_{jk_j})$ be the unique sequence of criteria indices from the root of the hierarchy to the leaf criterion C_j; that is, on the way from the root to the leaf criterion the nodes are visited in the order

$$\text{root} = C_{r_{j1}} \;\rightarrow\; C_{r_{j2}} \;\rightarrow\; \ldots \;\rightarrow\; C_{r_{jk_j}} = C_j.$$

Then, recursively applying the aggregation rule in either of Eq. (2) or (3) results in the overall weight

$$w'_j = \prod_{i=1}^{k_j} w_{r_{ji}}$$

with regard to the decision making goal. Hence, the hierarchical MCDM problem can be replaced by a non-hierarchically stated problem. The effect of the adjustment of weights can be visualized best with a circular treemap. In Fig. 4 the treemap corresponding to Example 1 is displayed.

3.2 Saaty's Analytic Hierarchy Process

The Analytic Hierarchy Process (AHP) by Saaty [28] is a well-known application of the hierarchical decomposition principle. It has been applied in countless cases and domains: Saaty [29] uses the AHP to determine the most livable cities in the United States; Saaty and Takizawa [33] employ an extended variant of the AHP to plan the design of a motorcycle; Saaty [30] cites an example where a family is advised in its plans to buy a family home; Olson and Dorai [26] query their students to assess potential job offers and evaluate the results with the AHP; McCaffrey and Koski [23] describe

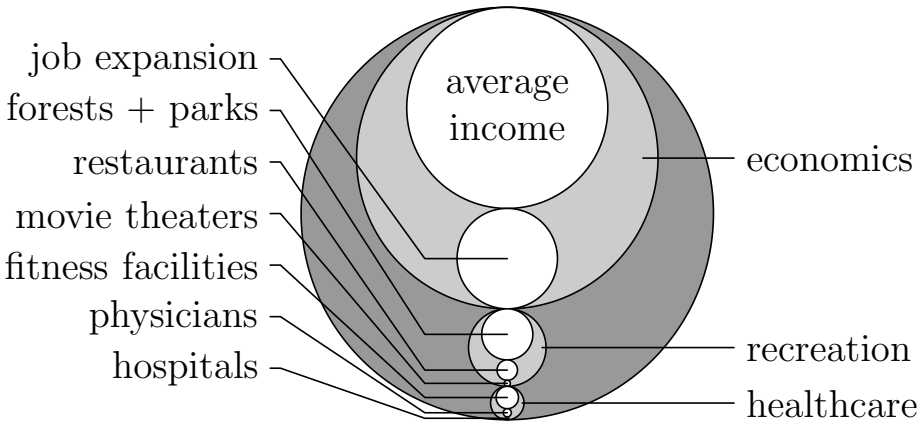


Figure 4: Circular treemap of the hierarchy in Fig. 3. — The diameter of a circle corresponds to the relative importance of the associated criterion. For numerical weight values see Fig. 3.

the MAGIQ method to calculate the quality of a software system — MAGIC is closely related to the AHP —, and Whitaker [39] presents various validation examples of the AHP. Further, an extensive list of examples for the application of AHP is given in Saaty [31].

For a motivation of the algorithms behind the AHP, let $C_1, C_2, \ldots, C_n$ be a set of criteria and let their weights be known, that is, w_i is the weight of criterion C_i. The weights can be expressed as a weight vector $w = (w_i)$. Knowing the weight vector, a $n \times n$ matrix of ratios $A = (a_{ij})$ where $a_{ij} = w_i/w_j$ can be constructed. Each entry a_{ij} represents the relative weight of criterion C_i with respect to criterion C_j. Multiplying the ratio matrix A with the weight vector w one obtains

$$Aw = \begin{bmatrix} w_1/w_1 & w_1/w_2 & \cdots & w_1/w_n \\ w_2/w_1 & w_2/w_2 & \cdots & w_2/w_n \\ \vdots & \vdots & \ddots & \vdots \\ w_n/w_1 & w_n/w_2 & \cdots & w_n/w_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = nw.$$

Hence, the weight vector is a right eigenvector of A with the eigenvalue n. Further, for every square matrix A the sum of all eigenvalues λ_i equals its trace, that is, the sum of the entries on the matrix' main diagonal, $\mathrm{tr}(A) = \sum_i a_{ii}$. For the ratio matrix,

$$\sum_{i=1}^{n} \lambda_i = \mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} 1 = n;$$

thus, w is the right eigenvector corresponding to the only non-zero eigenvalue. Now, what if the actual weight vector is unknown but a matrix of pairwise comparisons exists?

A matrix $A = (a_{ij})$ where $a_{ij} > 0$ and $a_{ji} = 1/a_{ij}$ for $1 \leq i, j \leq n$ is called *pairwise multiplicative*[†]. The matrix constructed from the weight vector in the previous paragraph is obviously pairwise multiplicative. On the other hand, a matrix can satisfy the terms of the definition without being derived in such a way. Therefore, a pairwise multiplicative matrix A is called *consistent* if there exists a weight vector $w = (w_i)$ with $a_{ij} = w_i/w_j$. The vector corresponding to a consistent matrix is only unique up to a multiplicative factor. In the following, the *normalized* weight vector is always taken as a representative for its class, that is, the vector w, where $\sum_{i=1}^{n} w_i = 1$. The sets of pairwise multiplicative matrices, normalized weight vectors, and consistent pairwise multiplicative matrices are denoted as $A^{\times}$, w^S, and C^S, respectively.

Given a matrix $A \in A^{\times}$, its element a_{ij} is interpreted as an approximation of the ratio w_i/w_j for some, as of yet, unknown weight vector $w = (w_i)$. Stated alternatively, the element a_{ij} is a single judgement, as provided by a decision maker, on the relative weight of criterion C_i with respect to criterion C_j. The goal is to reconstruct the original normalized weight vector from the pairwise comparisons expressed in the matrix. If the provided matrix is consistent, the right eigenvector belonging to the largest eigenvalue can be determined. After normalization, this eigenvector matches the sought-after weight vector. But, far more likely, the matrix will not be consistent.

The Perron-Frobenius theorem states that for each nonnegative, primitive matrix a largest eigenvalue $\lambda_{\max}$ and an associated eigenvector exist. The value $\lambda_{\max}$ is called the *dominant eigenvalue* and the eigenvector to go with it is called the *principal eigenvector*.

Theorem 1 (Perron-Frobenius theorem)**.** *Let $A \in \mathbb{R}^{n \times n}_{\geq 0}$ be a nonnegative, primitive matrix, that is, $A^k > 0$ for some $k \geq 1$, and let $\lambda_1, \lambda_2, \ldots, \lambda_s$ be its eigenvalues. Then*

1. *A has a positive, real eigenvalue $\lambda_{\max} := \lambda_1$ which is larger (in absolute value) than all other eigenvalues, i.e., $\lambda_{\max} > |\lambda_i|$ for all $2 \leq i \leq s$,*

2. *there is an eigenvector associated with $\lambda_{\max}$ that has only positive elements, and*

3. *the eigenvalue $\lambda_{\max}$ is a simple root of the characteristic polynomial; in particular, its associated eigenspaces are 1-dimensional.*

A pairwise multiplicative matrix is, by definition, positive and primitive; thus, it satisfies the preconditions of the theorem. Specifically, for consistent pairwise multiplicative matrices the principal eigenvector corresponds to the weight vector and $\lambda_{\max} = n$. Saaty [28] proved that for all $A \in A^{\times}$ the dominant eigenvalue satisfies $\lambda_{\max} \geq n$ and A is consistent if, and only if, $\lambda_{\max} = n$.

Still, the question persists whether the principal eigenvector is an acceptable approximation to the weight vector if a matrix is not consistent. Saaty states that "there is no easy way to study the sensitivity of the eigenvector w to errors in A" [28, page 239].

[†] Saaty [28] labels these matrices *reciprocal*, and Barzilai and Golany [6] dub them *pairwise multiplicative*. Throughout this report, we will consistently use the latter denomination.

intensity	verbal judgement
1	equal importance
3	moderate or weak importance
5	strong or essential importance
7	very strong or demonstrated importance
9	absolute or extreme importance

Table 2: Scale of relative importance proposed by Saaty [28, 32].

But, an estimate of the degree of consistency of a pairwise multiplicative matrix A can be given. It is

$$\begin{aligned}
& \sum_{i=1}^{n} \lambda_i = \operatorname{tr}(A) = n \\
\Longrightarrow \quad & \lambda_{\max} + \sum_{i=2}^{n} \lambda_i = n \\
\Longrightarrow \quad & -\sum_{i=2}^{n} \lambda_i = \lambda_{\max} - n \\
\Longrightarrow \quad & -\frac{1}{n-1} \sum_{i=2}^{n} \lambda_i = \frac{\lambda_{\max} - n}{n-1} \\
\Longrightarrow \quad & -\mu = \frac{\lambda_{\max} - n}{n-1},
\end{aligned}$$

where μ is the average nondominant eigenvalue. Saaty terms $-\mu$ the *consistency index* (C.I.) of a matrix [32].

Before the consistency index can be put to any use, the contents of a judgement matrix need to be further examined. Saaty defines a scale of nine grades to express the relation between any two criteria. A verbal expression is assigned to each of the odd integers between one and nine (Table 2). Even values in between can be used to express intermediate relationships or serve as a compromise. The reciprocals of these values express the inverse relationship as dictated by the definition of a pairwise multiplicative matrix. For example, if criterion C_4 is strongly preferred to criterion C_5 then we set in the judgement matrix $a_{45} := 5$ and $a_{54} := 1/5$.

To justify the choice of scale, Saaty cites psychological research that demonstrates that "an individual cannot simultaneously compare more than seven objects (plus or minus two) without being confused" [28, page 245]. Further, Saaty [32] lists 27 other scales suggested to him. He mentions that, in practice, only three scales showed good results: the scale from Table 2, a version of that scale where each integer between two and nine was multiplied with 0.9, and an exponential scale where the ith grade was assigned the value $2^{i/2}$. The latter scale was subsequently invalided by a counterexample.

Let us now return to the question of how consistent a judgement matrix is and to the

n	1	2	3	4	5
R.I.	0	0	.58	.90	1.12

n	6	7	8	9	10
R.I.	1.24	1.32	1.41	1.45	1.49

Table 3: Random consistency indices provided by Saaty [32].

discussion of its consistency index

$$\text{C.I.} = \frac{\lambda_{\max} - n}{n - 1}.$$

With increasing consistency of a matrix, its dominant eigenvalue tends to n; consequently, its consistency index approaches zero. But, comparative values are required to judge the actual degree of consistency. Saaty [32] researches the significance of the consistency index with randomized matrices. He generates pairwise multiplicative matrices filled with random judgements from the scale in Table 2. For a number of fixed n, he determines the average consistency index of 500 random $n \times n$ matrices (Table 3). The resulting values are called *random consistency indices* (R.I.) and are used as reference values.

The consistency of a given comparison matrix is evaluated by comparing its consistency index to the random consistency index matching its size. For any matrix, the ratio of C.I. to R.I. is termed its *consistency ratio* (C.R.). According to Saaty [32], the consistency ratio should generally not exceed the value of 0.10. Otherwise, he suggests that the underlying problem might not be fully understood and should be re-examined. A consistency ratio of more than 0.20 is intolerable. But, Saaty [28] also points out that getting a matrix more consistent does not mean that it matches the actual weights more closely. If a matrix is consistent, there is a logical relationship between the judgements it contains; with decreasing consistency the matrix gets more and more random. Whether the weights extracted from a comparison matrix actually represent the weights a decision maker had in mind can be verified only in practice.

Let $\mathrm{ev}_{\mathrm{N}} \colon A^{\times} \to w^{+}$ denote the operation of computing the normalized principal eigenvector of a pairwise multiplicative matrix. A common such algorithm is the *power method* or *von Mises*[†] *iteration* (see, e.g., Hohmann and Deuflhard [20, Section 5.2]). Given a matrix $A \in A^{\times}$ and an initial vector $b_0 \in \mathbb{R}^n$, the power method successively computes

$$b_{k+1} := Ab_k = A^{k+1}b_0. \tag{4}$$

If the initial vector contains a non-vanishing part in the direction of the principal eigenvector, then the sequence

$$w'_k := \frac{b_k}{\|b_k\|} \tag{5}$$

[†]Richard Edler von Mises (1883–1953); mathematician, engineer and philosopher.

will converge against a multiple of the normalized principal eigenvector. Stated alternatively, the initial vector b_0 must not be orthogonal to the eigenspace of $\lambda_{\max}$ if the algorithm is to succeed. The type of norm used in Eq. (5) is irrelevant. The speed of convergence of the power method depends on the degree of separation between the two largest eigenvalues $|\lambda_{\max}/\lambda_2|$. Assuming an acceptable consistency ratio, this requirement is satisfied by the judgement matrices.

Saaty [28] suggests an initial vector and a norm based on $\bar{e} := [1, 1, \ldots, 1]$. The norm of a vector b_k is obtained by multiplying it with the reciprocal of the sum of its elements, that is,

$$\|b_k\| := \frac{b_k}{b_k^T \bar{e}} \tag{6}$$

Further, for $\bar{e}$ to be a valid initial vector for the power method, it must be proved that it is not orthogonal to the principal eigenvector. An arbitrary vector x is orthogonal to $\bar{e}$ if, and only if, the scalar product $x\bar{e} = 0$. From the Perron-Frobenius theorem it is known that all elements of the principal eigenvector are positive and the same is, obviously, true for $\bar{e}$. Hence, $x\bar{e} > 0$ and $b_0 := \bar{e}$ can be used as start vector. The following theorem brings together von Mises' algorithm in Eqs. (4) and (5), Saaty's norm in Eq. (6), and the start vector $\bar{e}$.

Theorem 2 (Saaty [28, Theorem 6])**.** *For a pairwise multiplicative matrix $A \in A^{\times}$ and $\bar{e} = [1, 1, \ldots, 1]$*

$$\lim_{k\to\infty} \frac{A^k \bar{e}}{\|A^k \bar{e}\|} = C w_{\max},$$

where C is a constant and $w_{\max}$ is the normalized principal eigenvector associated with $\lambda_{\max}$.

Applying the algorithm to a judgement matrix, we can not only extract the weights for a set of criteria. A set of alternatives can be evaluated using the exact same method of pairwise comparison. Saaty [30] remarks the advantage of this approach: alternatives can be compared even if no objective scales exist. Such intangible criteria include, e.g., reputation, experience, beauty, or assertiveness. However, the method can also be applied if explicit measure on a ratio or non-ratio scale exist.

Finally, we have to aggregate all the weights and partial utilities into a overall utility vector with respect to the root criterion. Let $\Omega = \{O_1, O_2, \ldots, O_m$ be the set of alternatives that are supposed to be evaluated by the Analytic Hierarchy Process, and let $C_1, C_2, \ldots, C_n$ be the child nodes of an arbitrary node C of the AHP criterion tree. Further, let w_j be the weight of criterion C_j, and let $u_{C_j}(O_i)$ be the utility of alternative O_i with regard to criterion C_j. The utility of alternative O_i with regard to the father criterion C is then defined as the arithmetic mean

$$u_C(O_i) := \sum_{j=1}^{n} w_j u_{C_j}(O_i).$$

With this definition, we can recursively combine utilities and weights from the leaf nodes of the hierarchy up to the root criterion. The result is the utility vector with respect to the root criterion.

Note that the utilities at each criterion are normalized; specifically, this is true for the root criterion. This can be proved by structural induction: if the utilities are normalized for any set of child criteria in the hierarchy, they are also normalized for the parent criterion. With the definitions from the previous paragraph, let the utilities $u_{C_j}(O_i)$ be normalized for each fixed criterion C_j, that is, $\sum_i u_{C_j}(O_i) = 1$. Further, let $\sum_j w_j = 1$. Hence, at the parent criterion C we have for the utilities

$$\begin{aligned}\sum_{i=1}^{m} u_C(O_i) &= \sum_{i=1}^{m}\sum_{j=1}^{n} w_j u_{C_j}(O_i) \\ &= \sum_{j=1}^{n}\left[w_j \sum_{i=1}^{m} u_{C_j}(O_i)\right] \\ &= \sum_{j=1}^{m} w_j = 1.\end{aligned}$$

Finally, we need to verify the condition for the leafs of the hierarchy. But, as previously described, the *normalized* principal eigenvector is used to determine utilities in the leafs of the hierarchy, and it is computed for the weights of each set of child nodes belonging together. Thus, the base case is trivially true.

We provide a small toy example with two pairwise multiplicative matrices to illustrate the eigenvector method.

EXAMPLE 2. Probably everyone is familiar with the problem of selecting a flavor at the local ice cream parlor. A co-worker of the first author was kind enough to share his preferences with regard to this topic. The available flavors were (in this order) strawberry, yogurt, chocolate, stracciatella, and vanilla. Using the scale from Table 2, he expressed his likings in the judgement matrix

$$A_{\text{JM}} := \begin{bmatrix} 1 & 9 & 3 & 5 & 1 \\ 1/9 & 1 & 1/9 & 1/9 & 1/9 \\ 1/3 & 9 & 1 & 3 & 1/5 \\ 1/5 & 9 & 1/3 & 1 & 1/5 \\ 1 & 9 & 5 & 5 & 1 \end{bmatrix}.$$

We apply the power method in its variant from Theorem 2 and define

$$b_k := \frac{A_{\text{JM}}^k \bar{e}}{\|A_{\text{JM}}^k \bar{e}\|}.$$

Successively, one obtains

$$b_1^T := \begin{bmatrix} .289 & .022 & .206 & .163 & .320 \end{bmatrix},$$
$$b_2^T := \begin{bmatrix} .338 & .020 & .159 & .083 & .400 \end{bmatrix},$$
$$b_3^T := \begin{bmatrix} .341 & .024 & .147 & .087 & .401 \end{bmatrix},$$

and, after a few more iterations, the b_k tend towards the normalized principal eigenvector

$$b_6^T := \begin{bmatrix} .339 & .024 & .152 & .092 & .394 \end{bmatrix}.$$

Given a pairwise multiplicative matrix $A = (a_{ij})$ and an eigenvector $x = (x_i)$, the associated eigenvalue λ of x can be determined from any row of the matrix. Directly from $Ax = \lambda x$ follows

$$\sum_{j=1}^{n} a_{ij} x_j = \lambda x_i$$

for any fixed row i. Thus, $\lambda_{\max}$ can be computed from, e.g., the first row of A_{JM} as $\lambda_{\max} \approx 5.504$. Further, the consistency index is

$$\text{C.I.} = \frac{\lambda_{\max} - n}{n-1} = \frac{5.504 - 5}{5-1} = 0.126.$$

Using the random consistency index for $n = 5$ from Table 3, the consistency ratio is C.R. $\approx 0.126/1.12 \approx 0.11$. Not wishing to bother the decision maker again, we accept this ratio. Finally, the ranking of ice cream flavors from most preferred to least preferred is vanilla (.394), strawberry (.339), chocolate (.152), stracciatella (.092), and yogurt (.024) with the associated utilities provided in parentheses.

3.3 Criticism of AHP

The original AHP algorithm has drawn criticism from various sources. Most comments revolve around the fact, that certain seemingly innocuous changes to a problem instance can cause alternatives to switch places in the ranking. This phenomenon has become generally known as *rank reversal*.

The first example for rank reversal was published by Belton and Gear [11]. They showed that introducing a new alternative can cause previous alternatives to reverse their ranks. In their example, the rank reversal is triggered even though the pairwise comparisons between previous alternatives remain unchanged.

EXAMPLE 3 (Belton and Gear [11])**.** Let A, B, and C be alternatives which are to be ranked with regard to the three criteria C_1, C_2, and C_3. The corresponding pairwise comparison matrices are

$$C_1 \equiv \begin{bmatrix} 1 & 1/9 & 1 \\ 9 & 1 & 9 \\ 1 & 1/9 & 1 \end{bmatrix}, \quad C_2 \equiv \begin{bmatrix} 1 & 9 & 9 \\ 1/9 & 1 & 1 \\ 1/9 & 1 & 1 \end{bmatrix}, \tag{7a}$$

and

$$C_3 \equiv \begin{bmatrix} 1 & 8/9 & 8 \\ 9/8 & 1 & 9 \\ 1/8 & 1/9 & 1 \end{bmatrix}. \tag{7b}$$

Evaluating these matrices with Saaty's algorithm results in the normalized eigenvectors

$$u_{C_1} = \frac{1}{11}\begin{bmatrix} 1 \\ 9 \\ 1 \end{bmatrix}, \quad u_{C_2} = \frac{1}{11}\begin{bmatrix} 9 \\ 1 \\ 1 \end{bmatrix}, \quad u_{C_3} = \frac{1}{18}\begin{bmatrix} 8 \\ 9 \\ 1 \end{bmatrix}. \tag{8}$$

These individual values are aggregated assuming equally weighted criteria, i.e., $w_1 = w_2 = w_3 = 1/3$, into the final scores $A \equiv 0.45$, $B \equiv 0.47$, and $C \equiv 0.08$. Correspondingly, the ranking of the alternatives is

$$B \succ A \succ C. \tag{9}$$

Now, a copy of the alternative B is introduced and labeled D. Existing judgements are not modified but taken over verbatim. The new judgement matrices are

$$C_1 \equiv \begin{bmatrix} \star & \star & \star & 1/9 \\ \star & \star & \star & 1 \\ \star & \star & \star & 1/9 \\ 9 & 1 & 9 & 1 \end{bmatrix}, \quad C_2 \equiv \begin{bmatrix} \star & \star & \star & 9 \\ \star & \star & \star & 1 \\ \star & \star & \star & 1 \\ 1/9 & 1 & 1 & 1 \end{bmatrix},$$

and

$$C_3 \equiv \begin{bmatrix} \star & \star & \star & 8/9 \\ \star & \star & \star & 1 \\ \star & \star & \star & 1/9 \\ 9/8 & 1 & 9 & 1 \end{bmatrix}.$$

The stars designate the locations of the previous comparisons. In place of these stars, the values of the corresponding 3×3 matrix from Eqs. (7a) and (7b) are inserted. The normalized eigenvectors are determined as

$$u'_{C_1} = \frac{1}{20}\begin{bmatrix} 1 \\ 9 \\ 1 \\ 9 \end{bmatrix}, \quad u'_{C_2} = \frac{1}{12}\begin{bmatrix} 9 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad u'_{C_3} = \frac{1}{27}\begin{bmatrix} 8 \\ 9 \\ 1 \\ 9 \end{bmatrix}. \tag{10}$$

Aggregating them with $w_1 = w_2 = w_3 = 1/3$ produces the final scores $A \equiv 0.37$, $B \equiv 0.29$, $C \equiv 0.06$, and $D \equiv 0.29$ resulting, thus, in the ranking

$$A \succ B \approx D \succ C. \tag{11}$$

Comparison of the rankings (9) and (11) shows that the introduction of D caused the original alternatives A and B to reverse their ranks.

Belton and Gear [11] argue that the reversal is a result of the normalization of the eigenvectors. Applying the normalization factor to an eigenvector is in fact a transformation of the weight of the corresponding criterion. Let v_i be the normalization factor of the eigenvector of criterion C_i, that is, $\mathrm{ev}_\mathrm{N}(M_i) = v_i \,\mathrm{ev}(M_i)$ where M_i is the judgement matrix with regard to criterion C_i. Because of

$$\sum_{i=1}^{n} w_i \,\mathrm{ev}_\mathrm{N}(M_i) = \sum_{i=1}^{n} w_i v_i \,\mathrm{ev}(M_i) = \sum_{i=1}^{n} (w_i v_i) \,\mathrm{ev}(M_i),$$

the normalization can be regarded as transforming each weight w_i to a new weight $w_i' = w_i v_i$. The factor v_i is, by definition, the reciprocal of the sum of the values that the options have been assigned under C_i. Thus, it depends on the number of alternatives and each single judgement provided by a decision maker. For this reason, changing the number of alternatives may — but does not necessarily — cause rank reversal in the final ranking.

EXAMPLE 3 ***(continued).*** In the Belton-Gear example the weight vector (w_1, w_2, w_3) is actually not $(1/3, 1/3, 1/3)$ but instead scaled componentwise with the corresponding normalization factors. Using the factors from Eq. (8), the actual weights for the case with three alternatives are

$$w^{(1)} = \begin{bmatrix} 1/11 \cdot 1/3 \\ 1/11 \cdot 1/3 \\ 1/18 \cdot 1/3 \end{bmatrix} = \begin{bmatrix} 1/33 \\ 1/33 \\ 1/54 \end{bmatrix}.$$

We renormalize these values for the sake of comparison and get $(0.38, 0.38, 0.23)$.

The introduction of option D leads to the normalization factors in Eq. (10). They transform the original scale to

$$w^{(2)} = \begin{bmatrix} 1/20 \cdot 1/3 \\ 1/12 \cdot 1/3 \\ 1/27 \cdot 1/3 \end{bmatrix} = \begin{bmatrix} 1/60 \\ 1/36 \\ 1/81 \end{bmatrix},$$

which yields $(0.29, 0.49, 0.22)$ after normalization.

The original judgements in Eqs. (7a) and (7b) show that the example was carefully crafted to be affected by this transformation. Alternative A is strongly preferred to alternative B with regard to second criterion. For the first criterion, the preference is exactly converse. But both criteria have an actual weight of 0.38 and, thus, no alternative can dominate the other. It is the slight preference for B with regard to the last criterion that shifts the balance in B's favor.

Adding alternative D turns this situation around. The weight of the second criterion is significantly increased from 0.38 to 0.49. Simultaneously, the weight of the first criterion is reduced to 0.29. Now, the strong preference for A over B results in a overall dominance of A. The ranks of A and B are reversed.

	compute nodes	res. runtime (in hours)	storage (in GB)
offer 1	64	2.0	20
offer 2	32	4.0	25
offer 3	64	2.5	15

Table 4: Exemplary resource offers.

Belton and Gear suggest that this is not what most decision makers would expect. They propose to scale the weight with regard to a unit that is chosen to represent the criterion. They define the unit to be the largest value of a criterion. That is, they perform normalization by multiplying an eigenvector with the reciprocal of its largest component. This new normalization operation does not result in a reversal of ranks in the exemplary decision problem above. Saaty and Vargas [34] responded by showing that other examples exist, where the Belton-Gear normalization can cause alternatives to switch ranks.

Subsequently, Barzilai and Golany [7] proved that for *any* normalization operation an example can be constructed where the original AHP will cause a rank reversal. They claim that the normalization is an artificial construct introduced to enforce the construction of a unique weight vector. Given a weight vector $w = (w_i)$, they observe that a judgement matrix $A = (a_{ij})$ contains only weight ratios $a_{ij} = w_i/w_j$, where $1 \leq i, j \leq n$. From these ratios, the original weight vector can never be fully reconstructed. Whatever algorithm is employed, the resulting weights will at best be unique up to a multiplicative factor. Barzilai and Golany argue, that Saaty's normalization is merely to avoid this problem, that is, the unknown weights are replaced by a normalized proxy vector.

Barzilai even deems the "procedure of normalizing the weights at each node of the criteria tree [...] a fundamental error" [5, page 3]. The values of an alternative with regard to two different criteria may not be comparable, because both criteria are measured in different units. Barzilai interprets the weight of a criterion as a conversion factor to a common scale. Once all units are chosen, the values may only be multiplied by a fixed overall factor without skewing the results. But, as we have seen, normalization causes each weight to be multiplied by its own factor depending on the number of alternatives and their values. We provide an example to elucidate this argument.

EXAMPLE 4. Assume that resource offers for a compute job are to be evaluated. The criteria *number of compute nodes*, *reserved runtime* in hours, and *storage* in gigabytes have been selected as relevant. The corresponding weights w_1, w_2, and w_3 then represent coefficients that convert the criterion's particular unit into a common unit, say "virtual nodes"[†]. In this scenario, w_1 is specified in virtual nodes per real compute nodes, w_2 in virtual nodes per hour, and w_3 in virtual nodes per gigabyte. A weight

[†]The Maui Cluster Scheduler employs a similar concept, where the requirements of a job like memory, disk space, and swap space are expressed in "processor equivalents". See Jackson et al. [21, Section 6.1].

vector

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0.5 \\ 2.0 \end{bmatrix} \tag{12}$$

then expresses the assumption that a virtual node is equivalent to either one real compute node, half an hour of reserved runtime, or 2 gigabytes of hard disk space.

Once these units are in place, a scaling of individual weights as it occurs during normalization distorts the goal. Taking the alternatives from Table 4 as a reference, we obtain the three normalization factors $1/160$, $1/8.5$, and $1/60$ for nodes, runtime, and storage, respectively. These transform the intended weights (12) to the new scale of

$$w' = \begin{bmatrix} w'_1 \\ w'_2 \\ w'_3 \end{bmatrix} = \begin{bmatrix} 1/160 \cdot 1.0 \\ 1/8.5 \cdot 0.5 \\ 1/60 \cdot 2.0 \end{bmatrix} = \begin{bmatrix} 1/160 \\ 1/17 \\ 1/30 \end{bmatrix}.$$

Rescaling for comparison such that $w'_1 = 1.0$, we obtain $(1.0, 9.4, 5.3)$. Suddenly, a virtual node is equivalent to either a single real compute node, 9.4 hours of reserved runtime, or 5.3 gigabytes of hard disk space.

With the same reasoning, Barzilai [5] argues that weights can not be determined independently of the measurements that have been assigned to the alternatives. A criterion's unit directly influences the magnitude and interpretation of its weight. In an environment like the metascheduler where these units are predefined and values are obtained accordingly, this argument at least is negligible.

Further, Barzilai brings forward the criticism that the coefficient ratio w_i/w_j "cannot correspond to any reasonable notion of relative importance" [5, page 3]. Again, his rationale is with respect to the interpretation of weights as conversion factors. He holds that changing the unit of a criterion does require an adjustment of the ratios to reflect the new scale. That is, the "relative importance" of a criterion varies with its scale. However, a scale transformation should not increase or decrease a criterion's importance.

EXAMPLE 4 ***(continued).*** The weight vector in Eq. (12) expressed the offered number of compute nodes, the reserved runtime in hours, and the available storage in gigabytes as virtual node equivalents. Changing the unit of the storage criterion from gigabytes to megabytes, would require an adjustment of its coefficient by a factor of 1,000, that is,

$$w'' = \begin{bmatrix} 1.0 \\ 0.5 \\ 2,000 \end{bmatrix}.$$

Nevertheless, the importance of storage space relative to the number of compute nodes and the reserved runtime would be unchanged.

Barzilai et al. [9] criticize the use of eigenvectors and weighted arithmetic mean in determining and aggregating the partial decisions [see also 4, 5, 6, 7, 8]. First, the

eigenvector solution makes the algorithm dependent on the description of the problem. Let $A \in A^{\times}$ be a pairwise multiplicative matrix that contains a decision maker's judgements. How much *more* option i is preferred to option j is then represented by a_{ij}. If the decision maker was instead asked to judge how much *less* i is preferred to j, the result would be a another matrix

$$B = (b_{ij}) = (1/a_{ij}) = A^T,$$

where $1 \leq i, j \leq n$. Barzilai et al. observe that it would be reasonable to expect the algorithm to produce inverse weights and an inverse ranking for B. Formally, if $w = \mathrm{ev_N}(A)$ then

$$\mathrm{ev_N}(B) = \mathrm{ev_N}(1 \div A) = 1 \div w,$$

where $\div$ denotes componentwise division, should be the result. Barzilai calls this property *independence of scale-inversion* [4, page 1227]. However, the eigenvector solution does not satisfy this condition.

Second, they criticize that the combination of eigenvector and weighted arithmetic mean is not a linear transformation. Let $A_i \in A^{\times}$, where $1 \leq i \leq n$, be a set of judgement matrices with a common parent node in the criteria tree and $w_i \in \mathbb{R}^{+}_{\geq 0}$ the corresponding set of weights. Formally, the lack of linearity is reflected by

$$w_1 \,\mathrm{ev_N}(A_1) + w_2 \,\mathrm{ev_N}(A_2) + \cdots + w_n \,\mathrm{ev_N}(A_n) \neq \mathrm{ev_N}(w_1 A_1 + w_2 A_2 + \cdots + w_n A_n).$$

Thus, the order of operations is relevant when the partial judgements are aggregated. The result is different, whether judgement matrices are aggregated first and evaluated afterwards or vice versa. As a solution to this problem, Barzilai et al. [9] propose a different set of evaluation and aggregation methods. They axiomatically postulate requirements that eventually lead to independence of scale-inversion and linearity. We will describe their approach in Section 3.4.

Finan and Hurley [15] describe another case of rank reversal in the presence of *wash criteria*. A wash criterion is a criterion that provides no discrimination between the alternatives. That is, all alternatives are valued equally with regard to a wash criterion. Such a criterion can be removed from a single-level hierarchy without affecting the ranking of the alternatives. Starting with a two-level hierarchy this is no longer possible. An example can be constructed where both Saaty's AHP and the Multiplicative AHP of Barzilai et al. will cause rank reversal when the wash criterion is removed.

Liberatore and Nydick [22] replied that a wash criterion may not be removed by simply normalizing the weights of the remaining sibling nodes. The removal of a criterion always requires a reassessment of the parent criterion's weights. Still, the reassessment may not prevent the rank reversal. Nevertheless, Liberatore and Nydick argue that the AHP is adequately equipped to handle wash criteria.

3.4 Multiplicative AHP

The algebraic structure and axiomatic construction of the multiplicative AHP — and its isomorphism to a purely additive AHP — have been developed gradually in Barzilai [4], Barzilai and Golany [6], Barzilai et al. [9]. Barzilai and Golany [7] added an axiomatic design for the aggregation of partial results into a final score for each alternative. A method that considers the relative power in a group of a decision maker was described by Barzilai and Lootsma [8]. We introduce the existing results in this section using the notation established by these authors.

The basic definitions of the multiplicative AHP are similar to the definitions seen in Saaty's original AHP. A matrix $A = (a_{ij})$ is called *pairwise multiplicative* if, and only if, $a_{ij} > 0$ and $a_{ji} = 1/a_{ij}$ for $1 \leq i, j \leq n$. A weight vector $w = (w_i)$ is called *multiplicative* if, and only if, $w_i > 0$ for $1 \leq i \leq n$ and $\prod_{i=1}^{n} w_i = 1$. A pairwise multiplicative matrix $C = (c_{ij})$ is called *consistent* if, and only if, a multiplicative weight vector w exists with $c_{ij} = w_i/w_j$. The sets of pairwise multiplicative and consistent pairwise multiplicative matrices and the set of multiplicative weight vectors are denoted by $A^{\times}$, $C^{\times}$, and $w^{\times}$, respectively. The set of mappings $f\colon A^{\times} \to w^{\times}$ is identified by $f^{\times}$. Under componentwise multiplication the sets $A^{\times}$, $C^{\times}$, and $w^{\times}$ are groups[†]. $C^{\times}$ is a subgroup of $A^{\times}$ and $C^{\times}$ is isomorphic to $w^{\times}$, that is, $C^{\times} \cong w^{\times}$.

Given a pairwise multiplicative matrix $A = (a_{ij})$, the goal is to construct the weights w_i that come closest to satisfying $a_{ij} = w_i/w_j$. In other words, the solution to the problem is finding a specific $f \in f^{\times}$. Further, certain desirable features of f are postulated to restrict the set of all possible functions. Through an iterative refining (see [4, 6, 9]), the set of requirements was finally reduced to the following two axioms A1 and A2.

Axiom A1 (Barzilai [4, Axiom 1]). *If A is a consistent multiplicative matrix with an underlying multiplicative weight vector w (namely $a_{ij} = w_i/w_j$ and $\prod_{k=1}^{n} w_k = 1$), then the solution is the vector w, that is, $f(A) = w$.*

Axiom A2 (Barzilai [4, Axiom 2]). *The weight w_i attributed to alternative i is independent of relative measurements among alternatives other than i.*

Axiom A1 enforces that the function does map a consistent multiplicative matrix to its related weight vector. Axiom A2 requires that a weight w_k depends only on the judgements a_{ik} and a_{kj} and is independent from all a_{ij}, where $i \neq k$ and $j \neq k$. Its main purpose is to make the resulting variant of the AHP resilient to rank reversal. By restricting a weight's dependencies to judgements regarding the corresponding alternative, existing options can be removed or additional options introduced without simultaneously affecting the weights of remaining alternatives. The eigenvector solution of Saaty does obviously not satisfy the second axiom.

The Fundamental Theorem of Barzilai et al. [4] states that the geometric mean

[†]It is trivial to proof that each structure satisfies the group axioms, that is, closure, associativity, invertibility, and existence of the neutral element.

$\mathrm{f}_{\text{geom}}\colon A^{\times} \to w^{\times}$ with

$$w_i = \left[\prod_{j=1}^{n} a_{ij}\right]^{1/n} \tag{13}$$

is the only solution that fulfills both requirements A1 and A2. It can be proved, that the geometric mean is *independent of scale-inversion*, that is,

$$\mathrm{f}_{\text{geom}}(1 \div A) = 1 \div \mathrm{f}_{\text{geom}}(A).$$

Further, the geometric mean is *homomorph*, that is,

$$\mathrm{f}_{\text{geom}}(A * B) = \mathrm{f}_{\text{geom}}(A) * \mathrm{f}_{\text{geom}}(B).$$

Here, division $\div$ and multiplication $*$ are to be applied componentwise.

Still, the geometric mean is only sufficient to determine the weights corresponding to a single multiplicative matrix. It corresponds to the eigenvector solution in Saaty's AHP. Solving a multicriteria decision making problem further involves aggregating the single-criteria results. I.e., a counterpart to Saaty's use of the arithmetic mean is necessary. Barzilai and Golany [7] state three axioms that represent desirable properties for such an aggregation rule $F(w_1, w_2, \ldots, w_n; A_1, A_2, \ldots, A_n)$ that aggregates n weights w_i, where $\sum_{i=1}^{n} w_i = 1$, and judgement matrices A_i.

In preparation for the axioms, we define $\phi^{\times}$ to be set of all mappings $\phi\colon A^{\times} \to C^{\times}$, that is, the set of all mappings from multiplicative matrices to consistent multiplicative matrices. Further, let $P_i\colon A^{\times} \to A^{\times}$ be the operator that removes the ith row and column from an $n \times n$ matrix and produces, thus, an $(n-1) \times (n-1)$ matrix. Let $P = P_{i_1} \circ P_{i_2} \circ \cdots \circ P_{i_l}$ be an arbitrary sequence of l such operators. The matrix produced by P or any P_{i_k} is called a *principal minor* of the operator's input matrix.

Axiom B1 (Barzilai and Golany [7, Axiom 1])**.** *The aggregation rule F satisfies*

$$F(w_1, w_2, \ldots, w_n; P(A_1), P(A_2), \ldots, P(A_n)) \\ = P(F(w_1, w_2, \ldots, w_n; A_1, A_2, \ldots, A_n)),$$

where the operator P denotes taking a (fixed) principal minor of the appropriate matrices.

Axiom B2 (Barzilai and Golany [7, Axiom 2])**.** *If the input matrices of F are consistent, so is its output matrix:*

$$A_k \in C^{\times} \quad k = 1, 2, \ldots, n \\ \Rightarrow F(w_1, w_2, \ldots, w_n; A_1, A_2, \ldots, A_n) \in C^{\times}.$$

Axiom B3 (Barzilai and Golany [7, Axiom 3])**.** *For some $\phi \in \phi^{\times}$,*

$$F(w_1, w_2, \ldots, w_n; \phi(A_1), \phi(A_2), \ldots, \phi(A_n)) \\ = \phi(F(w_1, w_2, \ldots, w_n; A_1, A_2, \ldots, A_n)).$$

The axioms B1, B2, and B3 rely on the isomorphism $C^{\times} \cong w^{\times}$. It allows us to express the result of an aggregation F as a matrix instead of the usual weight vector. Axiom B1 is motivated by the requirement that adding or removing alternatives must not influence the scores of the remaining options. Axiom B2 postulates that once a consistent matrix is reached, the algorithm must produce only consistent matrices further on. And finally, axiom B3 demands that aggregation and evaluation be exchangeable. Obviously, Saaty's combination of eigenvector solution and arithmetic mean satisfies neither of the axioms. It can be proved, that the weighted geometric mean

$$F(w_1, w_2, \ldots, w_n; A_1, A_2, \ldots, A_n) = \prod_{i=1}^{n} A_i^{w_i}, \tag{14}$$

on the other hand, does satisfy each of the axioms B1, B2, and B3.

Barzilai and Lootsma [8] look into the subject of relative power amongst participants in a group of decision makers. Let p_d be the relative power of the decision maker d, $1 \leq d \leq g$, with $\sum_{d=1}^{g} p_d = 1$. Let r_{ijkd} be the judgement of alternative O_j compared to alternative O_k with regard to criterion C_i as given by decision maker d. The combined judgement of the group of decision makers respecting the relative power of each participant can then be expressed as the weighted geometric mean

$$r_{ijk} = \prod_{d=1}^{g} r_{ijkd}^{p_d}. \tag{15}$$

Assembling all previous steps into an overall expression, we can get a compact representation for the final score $u(O_j)$ of an alternative O_j. The pairwise comparisons involving the alternative O_j are combined using Eq. (13), the results aggregated across all criteria using Eq. (14), and the relative power of each decision maker is respected using Eq. (15) leading, finally, to

$$u(O_j) = \left[\prod_{k=1}^{m} \prod_{i=1}^{n} \prod_{d=1}^{g} r_{ijkd}^{w_i p_d} \right]^{1/m}. \tag{16}$$

This combination of two weighted geometric means and a regular geometric mean can be evaluated in any order.

The effect of the relative power p_d of a participant on the aggregated judgement r_{ijk} can be further assessed. Barzilai and Lootsma represent the original judgement on a geometric scale

$$r_{ijkd} = \exp(\gamma \delta_{jkd}),$$

where $\gamma \in \mathbb{R}$ is a scaling factor and the value δ_{jkd} represents a verbal judgement (see Table 5) that expresses the preference of decision maker d for alternative O_j over alternative O_k. They suggest to set the scaling factor depending on the topic judged to, e.g., $\ln 2$ for a resulting progression factor of 2.

Given a scenario with two decision makers, let ω be the relative power of the more powerful party. Further, let δ_{jk} be the judgement of the powerful decision maker and

δ_{jkd}	verbal judgement
-8	very strong preference for k over j
-6	strong preference for k over j
-4	definite preference for k over j
-2	weak preference for k over j
0	indifference
+2	weak preference for j over k
+4	definite preference for j over k
+6	strong preference for j over k
+8	very strong preference for j over k

Table 5: Verbal judgements proposed by Barzilai and Lootsma to express the preference for an alternative. [8]

let his weaker counterpart provide the judgement φ_{jk}, $\delta_{jk} \neq \varphi_{jk}$. Using the weighted geometric mean (15) to determine the aggregated judgement ϑ_{jk} leads to

$$\exp\big(\gamma \tfrac{\omega}{\omega+1}\delta_{jk}\big)\exp\big(\gamma \tfrac{1}{\omega+1}\varphi_{jk}\big) = \exp\big(\gamma\vartheta_{jk}\big).$$

The relative power ω can then be expressed independently of the geometric scale γ as

$$\omega = -\frac{\varphi_{jk} - \vartheta_{jk}}{\delta_{jk} - \vartheta_{jk}}. \tag{17}$$

Since $|\vartheta_{jk}| < |\delta_{jk}|$, the powerful decision maker can at best hope for $|\varphi_{jk}| = 7$ if he himself judges $|\delta_{jk}| = 8$. Let his opponent be diametrically opposed, that is, $\varphi_{jk} = -\delta_{jk} = -8$. Setting these values into Eq. (17) gives a relative power of $\omega = 15$ that is required for the powerful decision maker to dominate his opponent. At least in a two-party scenario a power ratio of 15:1 is sufficient to express the complete range of possibilities.

Thus, Barzilai and Lootsma propose to use a geometric scale with progression factor $\sqrt{2}$ to judge the relative power of a decision maker. The values 1, 2, 4, 8, 16 represent the verbal expressions *equally*, *somewhat more*, *definitely more*, *much more*, and *vastly more* powerful. Reciprocals represent corresponding inverse expressions and intermediate values can be used to express uncertainty. Using this scale, the power of decision makers in a group is compared in a pairwise multiplicative matrix. The relative power coefficients p_d can then be determined by applying the geometric mean.

Analogously to the multiplicative structure, the AHP can be expressed purely additive. In short, a *pairwise additive* matrix satisfies $a_{ji} = -a_{ij}$ and an *additive* weight vector is defined by $\sum_{i=1}^{n} w_i = 0$. Note the difference to Saaty's definition of a normalized vector! A pairwise additive matrix is called *consistent* if, and only if, it satisfies $c_{ij} = w_i - w_j$ for some additive weight vector. The sets corresponding to the multiplicative formulation are denoted by A^+, C^+, w^+, and f^+. Again, under componentwise addition the sets A^+, C^+, and w^+ are groups. C^+ is a subgroup of A^+ and C^+ is isomorphic to w^+, that is, $C^+ \cong w^+$.

More interestingly, $A^{\times} \cong A^{+}$, $C^{\times} \cong C^{+}$, and $w^{\times} \cong w^{+}$ under a componentwise application of a logarithm with an arbitrary fixed base and the corresponding inverse exponential function. Thus, the theory developed for the multiplicative AHP can be translated to corresponding results for the purely additive AHP. E.g., the arithmetic mean $\mathrm{f}_{\text{arith}} \colon A^{+} \to w^{+}$ with

$$w_i = \frac{1}{n} \sum_{j=1}^{n} a_{ij} \tag{18}$$

is the only mapping that satisfies the additive formulations of axioms A1 and A2. Further, it is independent of scale-inversion, that is,

$$-\mathrm{f}_{\text{arith}}(A) = \mathrm{f}_{\text{arith}}(-A)$$

and homomorph, that is,

$$\mathrm{f}_{\text{arith}}(A + B) = \mathrm{f}_{\text{arith}}(A) + \mathrm{f}_{\text{arith}}(B).$$

Barzilai and Golany [6] use the purely additive formulation of the AHP to explore the *measure of inconsistency* $\sum_{i,j=1}^{n} \epsilon_{ij}^2$ of a judgement matrix, where $\epsilon_{ij} = a_{ij} - (w_i - w_j)$ is the error of a single judgement. Expressing the weights w_i and w_j through the arithmetic mean from Eq. (18) leads to

$$\begin{aligned}
\epsilon_{ij} &= a_{ij} - \frac{1}{n}\left[\sum_{k=1}^{n} a_{ik} - \sum_{k=1}^{n} a_{jk}\right] \\
&= a_{ij} + \frac{1}{n}\left[\sum_{k=1}^{n} a_{jk} + \sum_{k=1}^{n} a_{ki}\right] \\
&= \frac{1}{n} \sum_{k=1}^{n} (a_{ij} + a_{jk} + a_{ki}).
\end{aligned}$$

Let $A = (a_{ij})$ be a consistent additive matrix with the corresponding weight vector $w = (w_i)$, then

$$\begin{aligned}
a_{ij} + a_{jk} &= (w_i - w_j) + (w_j - w_k) \\
&= w_i - w_k \\
&= a_{ik} = -a_{ki}.
\end{aligned}$$

Thus, in a consistent case $a_{ij} + a_{jk} + a_{ki} = 0$ and the error $\epsilon_{ij} = 0$. In all other cases, ϵ_{ij} is the average inconsistency over all triples with a fixed i and j. A similar expression can be given for the multiplicative AHP based on $\epsilon_{ij} = \log a_{ij} - (\log w_i - \log w_j)$. [4, 6]

Note that although there is an isomorphism between the multiplicative and additive structures of the AHP both formulations are self-contained. Barzilai and Golany see the mixing of multiplicative and additive features in Saaty's AHP as one of its major problems: "[Rank reversal] is a symptom of inherent problems with the AHP: [...] non-multiplicative procedures (the weighted arithmetic mean and the eigenvector) are imposed on an intrinsically multiplicative structure" [7, page 63].

3.5 Rank-order Centroid

Solymosi and Dombi [36] propose an algorithm motivated by geometric ideas that determines approximated weights for a set of criteria. A decision maker is asked to iteratively compare subsets of criteria. Each judgement describes a hyperplane that separates the space of weights into a valid and an invalid half. Combining all judgements leads to a subspace bounded by a polyhedron. The polyhedron contains the weight vectors that satisfy all judgements simultaneously. Then, the centroid of the polyhedron is taken as an approximation to the real weights. Solymosi and Dombi call the centroid the point of *centralized weights*.

Let $\mathcal{P}(\Gamma)$ be the powerset of the set of all criteria $\Gamma = \{C_1, C_2, \ldots, C_n\}$. It is assumed, that a decision maker inherently has an unknown mapping of criteria to weight values in his mind, $w\colon \mathcal{P}(\Gamma) \to \mathbb{R}_{\geq 0}$. The mapping is assumed to be *additive*, that is, $w(a \cup b) = w(a) + w(b)$ for all $a, b \in \mathcal{P}(\Gamma)$, where $a \cap b = \emptyset$, and *standardized*, that is, $w(\Gamma) = 1$. Using this mapping, a preference relation $\succ$ can be defined as

$$a \succ b \quad :\Longleftrightarrow \quad w(a) > w(b) + \delta \tag{19}$$

with a nonnegative constant $\delta \in \mathbb{R}_{\geq 0}$. Further, we define an indifference relation $\approx$ as

$$a \approx b \quad :\Longleftrightarrow \quad \neg(a \succ b) \wedge \neg(b \succ a). \tag{20}$$

Note that the relation to the left of the double arrow in (19) compares sets of criteria; the relation to the right compares real numbers. All relations in (20) compare sets of criteria.

Definition 1 (Pirlot and Vincke [27, Definition 3.1])**.** *A reflexive relation $\mathbf{R} = (\mathbf{P}, \mathbf{I})$ on a finite set A is a semiorder if, and only if, there exist a real-valued function g, defined on A, and a nonnegative constant q such that, for all $a, b \in A$,*

$$\begin{cases} a\mathbf{P}b & \Longleftrightarrow \quad g(a) > g(b) + q, \\ a\mathbf{I}b & \Longleftrightarrow \quad |g(a) - g(b)| \leq q, \end{cases}$$

or, equivalently,

$$a\mathbf{R}b \quad \Longleftrightarrow \quad g(a) \geq g(b) - q.$$

Obviously, the preference relation $\succ$ and the indifference relation $\approx$ satisfy the requirements of this definition. Hence, $\succsim = (\succ, \approx)$ is a semiorder with the corresponding real-valued representation $w\colon \mathcal{P}(\Gamma) \to \mathbb{R}_{\geq 0}$. Note, that the indifference relation $\approx$ is generally not transitive, that is,

$$(a \approx b) \wedge (b \approx c) \not\Rightarrow (a \approx c).$$

It is transitive if, and only if, $\delta = 0$.

Let $\vec{w} = (w_i)_{1 \leq i \leq n+1}$ be an extended weight vector with the unknown weights $w_i = w(C_i)$, $1 \leq i \leq n$, and the threshold $w_{n+1} = \delta$. From the assumption that

the mapping w be standardized, it follows that $\sum_{i=1}^{n} w_i = 1$. Thus, the first n unit vectors e_i of $\mathbb{R}^{n+1}$ represent extremal but valid weight vectors. They define a regular polyhedron that contains all other weight vectors with $w_{n+1} = 0$. Allowing $w_{n+1} \geq 0$ adds a positive infinite extension in the $(n+1)$th dimension.

Every judgement $a \succ b$ or $\neg(a \succ b) \Leftrightarrow b \succsim a$ can be written as a linear inequation using the weights w_i. Let $a = \{C_{i_1}, \ldots, C_{i_g}\} \in \mathcal{P}(\Gamma)$ and $b = \{C_{j_1}, \ldots, C_{j_h}\} \in \mathcal{P}(\Gamma)$, where $a \cap b = \emptyset$. Using the definitions and the additive assumption about the mapping w, we have

$$\begin{aligned} a \succ b &\iff w(a) > w(b) + \delta \\ &\iff -w(a) + w(b) + \delta < 0 \\ &\iff -\sum_{k=1}^{g} w(C_{i_k}) + \sum_{k=1}^{h} w(C_{j_k}) + w_{n+1} < 0 \end{aligned}$$

and, analogously,

$$\begin{aligned} b \succsim a &\iff w(b) \geq w(a) - \delta \\ &\iff w(a) - w(b) - \delta \leq 0 \\ &\iff \sum_{k=1}^{g} w(C_{i_k}) - \sum_{k=1}^{h} w(C_{j_k}) - w_{n+1} \leq 0. \end{aligned}$$

Thus, the inequality corresponding to the ith judgement can be expressed as either $\sum_{j=1}^{n+1} a_{ij} w_j < 0$ or $\sum_{j=1}^{n+1} a_{ij} w_j \leq 0$ with $a_{ij} \in \{-1, 0, +1\}$. The actual value a_{ij} is given indirectly by

$$a_{ij} = \begin{cases} t_{ij} & \text{if the } i\text{th judgement is } a \succ b, \\ -t_{ij} & \text{if the } i\text{th judgement is } \neg(a \succ b), \end{cases}$$

and

$$t_{ij} = \begin{cases} -1 & \text{if } C_j \in a, \\ 0 & \text{if } C_j \notin a \cup b, \\ +1 & \text{if } C_j \in b \text{ or } j = n+1. \end{cases}$$

Each type of inequality divides the space of weight vectors into a valid and an invalid half. The border between both subspaces is defined by a hyperplane in $\mathbb{R}^{n+1}$. Formally, the hyperplane of the ith judgement is defined by

$$A_i \vec{w} = 0$$

with $A_i = (a_{ij})_{1 \leq j \leq n+1}$.

The algorithm described by Solymosi and Dombi iteratively determine the polyhedra produced by these cuts through the space of valid weight vectors. After each judgement, the centroid is determined as an approximation to the real weights. It is

defined as the arithmetic mean of a polyhedron's vertices. The centroid always falls inside the bounded space and, therefore, satisfies all judgements. Further, it minimizes the maximum error due to its position in the center of the polyhedron. At any point, the decision maker can accept the approximated weights and terminate the algorithm. Hence, the runtime of the algorithm is dynamic and depends on the intended quality of the approximation.

Instead of repeating the rather complicated matrix operations given by Solymosi and Dombi [36], we provide a small example for 3 criteria (see Fig. 5).

EXAMPLE 5. Let $\Gamma = \{C_1, C_2, C_3\}$ be the set of criteria. Initially, the bounding polyhedron P_0 is defined by the first three unit vectors e_1, e_2, and e_3 of $\mathbb{R}^4$ and a positive infinite extension in direction of e_4. This shape can be mapped to the $\mathbb{R}^3$ as displayed in Fig. 5a.

The first judgement provided by the decision maker is $\neg(C_1 \succ C_2)$. We have $\neg(C_1 \succ C_2) \iff w_2 \geq w_1 - \delta$ and, thus, the hyperplane

$$H_0 : \quad \begin{bmatrix} 1 & -1 & 0 & -1 \end{bmatrix}^T \vec{w} = 0.$$

Intersecting hyperplane H_0 with the polyhedron P_0 leads to a new polyhedron P_1 defined by its vertices e_2, e_3, $p_1 = [\frac{1}{2}, \frac{1}{2}, 0, 0]$, $p_2 = [1, 0, 0, 1]$, and a positive infinite extension in direction of e_4 (see Fig. 5b).

The second judgement is $C_1 \succ C_3 \iff w_1 > w_3 + \delta$. Its corresponding hyperplane

$$H_1 : \quad \begin{bmatrix} -1 & 0 & 1 & 1 \end{bmatrix}^T \vec{w} = 0.$$

intersects with P_1 leading to a new polyhedron P_2. Fig. 5c shows that this cut removes the positive infinite extension in direction of e_4. The remaining space has the shape of a tetrahedron with the vertices e_2, p_1, p_2, and $p_3 = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0]$.

The third judgement $C_2 \succ C_3$ leads to the inequality $-w_2 + w_3 + \delta < 0$. Thus, the hyperplane is

$$H_2 : \quad \begin{bmatrix} 0 & -1 & 1 & 1 \end{bmatrix}^T \vec{w} = 0.$$

It is displayed in Fig. 5d. Intersecting the hyperplane with P_2 removes the tip of the tetrahedron, leaving the polyhedron bounded by the vertices e_2, p_1, p_2, $p_4 = [\frac{2}{3}, \frac{1}{3}, 0, \frac{1}{3}]$, and $p_5 = [\frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}]$.

Finally, the last judgement $\neg(C_1 \succ \{C_2, C_3\})$ corresponds to the inequality $w_1 - w_2 - w_3 - \delta \leq 0$. Touching only one of the remaining polyhedron's boundaries, the hyperplane

$$H_3 : \quad \begin{bmatrix} 1 & -1 & -1 & -1 \end{bmatrix}^T \vec{w} = 0.$$

(see Fig. 5e) does not intersect with it. Thus, the judgement is consistent with previous relations and provides no new information. Inverting the judgement would have lead to the same hyperplane but would have invalidated the opposite half of the space. In that case, it had removed the complete polyhedron and had left no valid weight vectors remaining.

The polyhedron of valid weight vectors is displayed in Fig. 5f. Its centroid

$$\vec{w}_c = \frac{1}{5}e_2 + \frac{1}{5}p_1 + \frac{1}{5}p_3 + \frac{1}{5}p_4 + \frac{1}{5}p_5 \approx \begin{bmatrix} 0.40 \\ 0.54 \\ 0.07 \\ 0.17 \end{bmatrix}$$

represents the weights returned by the algorithm. Further, the second dot in Fig. 5f marks the weights originally used to construct this example, that is, $w_1 = 0.46$, $w_2 = 0.38$, $w_3 = 0.16$, and $\delta = 0.09$.

A simplified variant of the centralized weights method is proposed by Barron and Barrett [3]. They call their algorithm *Rank-order Centroid* (ROC) and require as input only a ranking of the criteria

$$C_{i_1} \succsim C_{i_2} \succsim \cdots \succsim C_{i_n}.$$

Assuming a strict preference $\succ$, the centroid is calculated just like in the method of centralized weights. This leads to the centroid with

$$w(C_{i_k}) = \frac{1}{n}\sum_{j=k}^{n}\frac{1}{j}. \tag{21}$$

If equality resp. indifference is allowed, then the weights obtained by (21) are averaged for each set of indifferent criteria. Note, that the weight of a criterion depends only on the criterion's position in the ranking and the total number of criteria. Thus, for any given n the weights can be computed statically and stored in a table.

EXAMPLE 6. Let $\Gamma = \{C_1, C_2, C_3, C_4\}$ be the set of alternatives and

$$C_2 \succ C_4 \approx C_3 \succ C_1$$

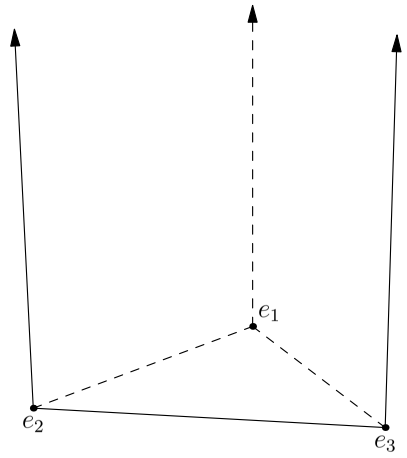
the ranking provided by the decision maker. First, we assume strict preference and obtain in the ranked order

$$w(C_2) = \frac{1}{4}\sum_{j=1}^{4}\frac{1}{j} = \frac{25}{48}, \qquad w(C_4) = \frac{1}{4}\sum_{j=2}^{4}\frac{1}{j} = \frac{13}{48},$$
$$w(C_3) = \frac{1}{4}\sum_{j=3}^{4}\frac{1}{j} = \frac{7}{48}, \qquad w(C_1) = \frac{1}{4}\sum_{j=4}^{4}\frac{1}{j} = \frac{3}{48}.$$

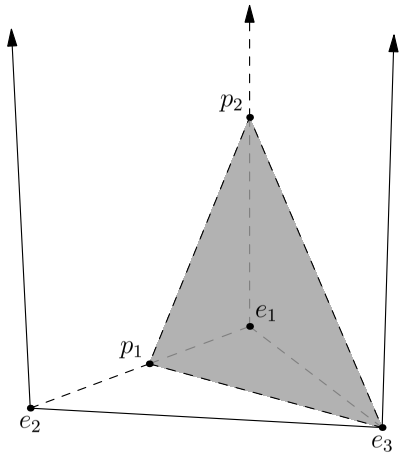
Afterwards, we incorporate the equality of C_3 and C_4 by averaging the corresponding weights

$$w(C_3) = w(C_4) = \frac{1}{2}\left[\frac{13}{48} + \frac{7}{48}\right] = \frac{10}{48}.$$

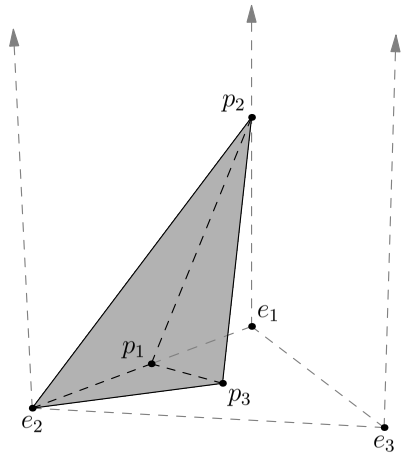
The final weights are $w_1 \approx 0.06$, $w_2 \approx 0.52$, $w_3 \approx 0.21$, and $w_4 \approx 0.21$.



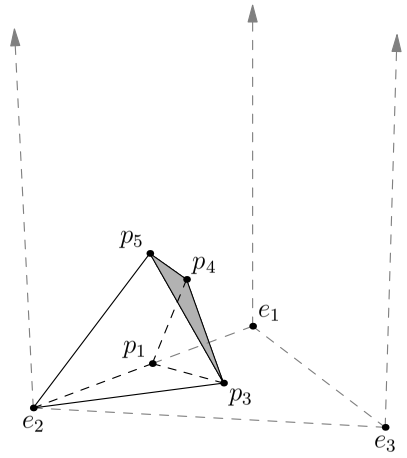
(a) Initial space of all weight vectors $w \in \mathbb{R}^4$ with $\sum_{i=1}^{3} w_i = 1$. All valid weight vectors are contained in the displayed column with a triangular base.



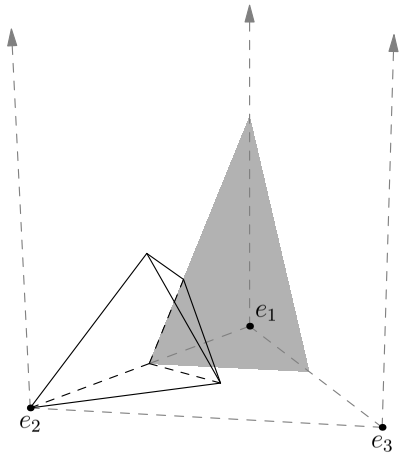
(b) The hyperplane representing the judgement $\neg(c_1 \succ c_2)$ removes the tetrahedron-shaped space bounded by the vertices e_1, e_3, p_1, and p_2.



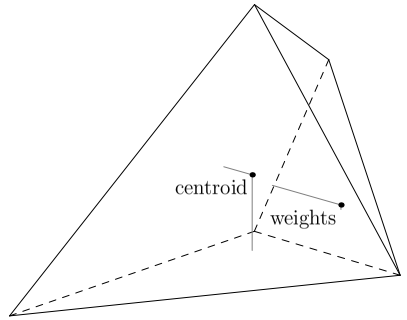
(c) The judgement $c_1 \succ c_3$ invalidates most of the weight vectors leaving only the tetrahedron given by the vertices e_2, p_1, p_2, and p_3.



(d) The last pairwise judgement $c_2 \succ c_3$ reduces the space of valid weight vectors to the polyhedron limited by the vertices e_2, p_1, p_3, p_4, and p_5.



(e) The judgement $\neg(c_1 \succ \{c_2, c_3\})$ is consistent with previous judgements and does not further restrict the space of valid weights.



(f) Magnified view of the final space of valid weight vectors. The centroid is returned by the ROC method; the other dot marks the weights used to construct the example.

Figure 5: Exemplary application of the ROC method with three criteria. — Black lines represent the edges of the polyhedron that contains the valid weight vectors: solid edges are visible from the viewpoint; dashed edges are concealed. The cut-off surface corresponding to the current judgement is highlighted in gray. Gray, dashed lines represent edges of the original column of valid weights that have been removed by previous cuts.

Olson and Dorai [26] compared the results produced by Saaty's AHP against the results obtained by the ROC method. Using judgements provided by their students, they showed that the rankings produced in an exemplary hierarchical multicriteria decision making problem differed only slightly. McCaffrey and Koski [23] give an introduction into the Multi-Attribute Global Inference of Quality (MAGIQ) technique used to determine the quality of a software system. MAGIQ relies on the ROC method to weigh the metrics considered in the competitive analysis.

4 Decision Making and Scheduling

Metascheduling is concerned with the scheduling of jobs across independent resources in distinct administrative domains. Hence, the migration of a job description requires multiple decisions to be made by different, remote parties. Two such scenarios occur frequently in a grid: *resource requests* and *resource announcements*. The former denotes a request made for a resource that matches an existing job's requirements, and the latter denotes a request made for a job that can be used to improve the utilization of an idle resource. Ultimately, both situations are similar: the objective is to find a good combination of job and resource and execute the former on the latter. Bringing the previous sections together, it can be said that multicriteria decision making is a major responsibility for every scheduler.

In a resource request scenario, the migration is initiated by a site that wishes to discard an existing job. First, the source site has to decide which of the queued jobs provides the least utility and should be relinquished; it then issues a search request and waits for resource offers. A grid peer that receives the request checks whether it satisfies the stringent conditions of the requirements description, and, if so, whether the job has enough utility to make an acceptance worthwhile. Subsequently, the grid peer decides which of the computing slots on its resource will be offered to the job. It serializes the description of the offer and transfers it back to the source site. Now, the source site collects all received resource offers and evaluates them. It picks the offer that suggests the most utility and migrates the job description.

The illustrated workflow incorporates four separate decision making processes. Goal of the first decision is to select a migration candidate; that is, the set of options encompasses all jobs queued at the source site. Three parties, each of them inspired by different motives, will be affected by the result: the job owner, the resource provider, and the grid community. Below, we will explain how the interests can be combined into a set of criteria with associated weights. Following the selection, the next two decision making processes take place on the remote peer to evaluate whether a resource will be offered to a request and, afterwards, which computing slot to present. We merge both events into a single decision by letting the metascheduler determine the utility of executing the provided job in each separate window of resource availability. Based on the results, a slot on the resource is only offered to the requesting job if the increased utility warrants it. Hence, the set of options is composed of each available slot on the resource. Note, that the sites making a decision in this and the

previous situation are not identical. Commonly, the resources will be owned by different resource providers and, consequently, governed by different goals. Following a distributed approach, this is a perfectly valid situation. Eventually, the last decision in the workflow is made on the original resource where the job is queued. Evaluating all the resource offers received as replies to the request, a target for the migration is selected. Parties interested in this decision will usually be the job owner and the grid community. The source site may participate; however, it holds no stakes in the result as it will relinquish the job anyway. Instead, it may be more useful to allow the destination sites to submit their objectives to be considered in the decision making process.

The decision makers in our scenario can be divided into three separate groups: users, resource providers, and grid community. First, job owners pursue their own objectives only and, generally, define the utility with respect to the features offered to them by the resource providers. We consider it too restrictive to confine users to a single set of goals to be applied uniformly across all their jobs; hence, every job must be assumed to have its own associated set of decision criteria. Users define a fixed set of criteria for each of their jobs that is attached to the its description and transmitted with the request. Criteria that are of interest to the average user include, e.g., reserved number of CPUs, disk space, memory, expected waiting time, and allocated job runtime.

Second, resource providers can be expected to work towards their own benefits, too; but, commonly, they also opt to consider additional criteria that improve the situation for their local users. We expect resource providers to define an overall set of criteria to govern all their decisions in a specific situation in the workflow. Thus, providers may opt for simplicity and define a single set, but they may as well define separate criteria with respect to job selection, resource offering, and offer selection. From our experience, resource providers consider criteria such as job owners, user groups, projects, utilization of the resource, average waiting time, improved access to the resource for local users, and length of job queues.

Finally, the grid community is introduced as a kind of global decision maker that pursues gridwide improvements and binds together the mostly selfish acting resource providers and job owners. Its goal is defined by a fixed set of criteria that expresses the essential prospects of a grid such as balanced load across all resources, fair access for all users in the grid, etc. There is no conceptual problem in allowing the grid community to put down distinct sets for different types of situations.

A resource announcement is issued by an idle computing resource that plans to accept additional jobs and improve its utilization. It creates a specification of the services offered to a candidate job, serializes it, and announces it to the grid. Each grid peer that receives the offer evaluates its job queue to determine candidates whose requirements can be satisfied by the remote resource. The peer optionally selects a subset of these jobs and transmits their descriptions back to the idle site. Here, these requests are collected and a decision is made to accept on of the associated jobs. Note that the second half of the process is, in fact, a stripped down resource request scenario: the peer evaluates its job queue to select migration candidates, requests to be allocated the idle resource, and, if accepted, migrates the job description.

The workflow for a resource announcements seems to contain a new decision making process: a site that selects the window of availability to be announced. However, in practice the former situation often requires no explicit selection to be made; either the resource has a low utilization and just announces the general availability of computing power or it offers a dedicated window that has been reserved for grid usage by a resource's administration. All the remaining decisions in the workflow correspond to events in the sequence of steps given for a resource request.

Note, that all of the previously mentioned criteria represent pure optimization examples. An additional class of aspects are the stringent conditions. These conditions include, e.g., the general eligibility of a user to utilize a resource, the availability of required applications and libraries, and the type of hardware supported by a user-supplied application. A stringent condition is either satisfied or it is not satisfied. There is no middle ground for elaboration. Thus, this class of conditions can not be handled by an optimization algorithm.

In the following, we assume that stringent conditions are accounted for by the resource discovery framework. That is, the list of available resources generated by the framework contains only those offers that satisfy all stringent conditions spelled out by a job's requirements. This assumption imposes no artificial hardness on the resource discovery process. It can be easily enforced, that a metascheduler replies only to a request if it can actually provide the requested service.

In addition to the illustrated processes, there is another detail to be accounted for in the migration of a job description. An offer obtained by a site will usually be valid only for a limited time. Specifically, the offering site may reserve the offered timeslot, e.g., until acknowledged or canceled by the receiving site, for a limited time only, or it may not provide any guarantees on the period of availability at all. Each of these scenarios requires the receiving site to react accordingly by, respectively, either canceling unused offers in a timely fashion, acknowledging offers within their restricted lifetime, or reissuing a request if previous offers are withdrawn.

In an unstructured grid of independent peers, either of these cases can occur, that is, a metascheduler will have to support each scenario. For this reason, an MCDM algorithm is useful that ranks the offers in order of decreasing preference. Then, the metascheduler iterates through this list and tries to acknowledge the offers subsequently. The first offer that is still available is accepted and the remaining offers explicitly canceled. Hence, an algorithm will always return the best possible resource for a job regardless of the guarantees given by remote peers.

5 AHP for Scheduling

Using the examples of Maui batch scheduler and service level agreements, we have seen that MCDM problems in the scheduling domain are commonly solved by combining various attributes by means of a weighted sum. The extensibility of these specialized decision making algorithms is limited as they are solely focused on supporting the scheduling process. A decision is usually restricted to a fixed set of criteria that are

hard-coded into the scheduling software. Thus, customers must rely on the developers of the scheduler to support all relevant criteria, and they are precluded from adding additional facets to a decision. Further, weights are exclusively defined by the administrator of a resource; hence, decisions only take into account the interests of the resource provider. Participation by additional decision makers such as the job owner or the grid community is impossible. We argue that a hierarchical heuristic based on the Analytic Hierarchy Process (AHP) can be a dynamic alternative to existing procedures.

The hierarchical structure of a criterion tree defines a concise representation of a decision maker's goal. Its recursive evaluation combines the hierarchy, the explicit criteria, and their weights into an overall ranking of the options. An aggregation algorithm such as the arithmetic or geometric mean is a generic procedure that is independent of the particular characteristics. With respect to the utilities, the inner nodes of the tree are recursively defined by the weights of their children and the utilities there. Hence, the metascheduler only needs to know how the options are supposed to be evaluated in the leaves of the tree; that is, it must be able to map a pair of an option and criterion to a numeric utility value. In practice, it is unnecessary to provide a mapping for every explicit option; instead, it is sufficient to define rules for each generic type of option, e.g., job or resource offer, and let the rules describe how to generate the value for a particular criterion. Moreover, this knowledge does not need to be hard-coded into the metascheduler; it can be provided by configuration, by extensions, or programmatically through scripting languages. E.g., our metascheduler represents a criterion by a unique qualified name as defined by the XML schema language, that is, as a string composed of a namespace URI and a local part. For example, the following strings are syntactically valid identifiers for criteria:

- `{http://example.com}WaitingTime`
- `{http://myprovider/myresource}UserID`
- `{urn:mysched:criterion}QueueLength`
- `{urn:mygrid:criteria:20090604}AvgJobRuntime`

Further, our metascheduler defines an interface which may be implemented to describe how to map a criterion and an option to a numeric utility. Thus, the scheduler can be easily extended to support additional criteria by any interested party.

However, such a change requires the installation of an additional module into a particular instance of the metascheduler. Note, that this is not only an issue of our implementation: the evaluation of the leaf criteria takes place at the junction between metascheduler and local batch scheduler. Therefore, any modifications will have to be implemented in cooperation with the resource provider hosting a metascheduler instance. Depending on the intentions, the support of a new criterion can be handled locally and independently by each resource provider. As long as a criterion will only be used in local decisions, there is no requirement to have other providers acknowledge and accept the change. While every grid peer could use its own set of criteria, the existence of a basic vocabulary of common criteria is essential to the functioning of

this approach to metascheduling. In practice, we expect some sites to have their own specialized facets that may represent particular features or optional information. To let users take advantage of these extensions and to prevent them from adding invalid criteria to their hierarchies, we envision that they will be able to obtain a list of supported patterns via a user interface.

The definition of the weights can be obtained by any of the previously presented algorithms: pairwise comparisons as in AHP and MAHP or an ordering as in ROC. Neither of the procedures requires any higher mathematics and all of them may be used even by a casual user — with a little support from the user interface. The necessary input data, that is, a matrix for the pairwise comparisons or a list for the ordering, can be acquired even through a simple HTML based web interface. Once the weights have been obtained, they remain static for the lifetime of the corresponding hierarchy. Hence, both can be stored according to their intended purpose as, e.g., a configuration file within the metascheduler installation or attached to the job description. For example, our implementation of the metascheduler represents each criteria tree as an XML document and, if need be, embeds it into the Job Submission Description Language (JSDL) document of a job.

The Analytic Hierarchy Process and its multiplicative cousin explicitly define the weighted geometric mean as an aggregation function. Unfortunately, this approach can not be used together with our metascheduling design as it requires each of the decision makers to use the same hierarchy in their evaluation. However, one particular strength of our algorithm is exactly the ability to let participants define their own criteria. Hence, we took a different course and interpret each parties opinion as a unique aspect of the overall decision. As such, we join the separate trees under a common root and introduce the three inner criteria *user*, *resource provider*, and *grid community*. Consequently, we can employ the existing mechanisms to evaluate the new meta tree. Note, that this idea is probably not an original innovation; while we have not seen any practical examples of this approach publicized before, e.g., Aczél and Saaty describe their aggregation method as comparable "with what one would obtain if each of the participants were to be included as a decision maker in the hierarchy" [1, page 93].

Yet, this construction introduces a new issue: there is no procedure to prevent different decision makers from using identical criteria inside their trees. As a result, the branches of the overall hierarchy may have interdependencies which is explicitly forbidden by the standards of Miller [24] and Saaty [28]. Saaty and Takizawa [33] describe how the AHP can be extended to work for generic graphs of criteria, where an edge between a pair of criteria represents a dependency. The resulting algorithm is the Analytic Network Process (ANP). However, the handling of the algorithm is more delicate, the structures less comprehensible, and, most importantly, the automatic joining of the parties' subgraphs is only possible for a few elementary cases. Further, in our application each singular decision of a metascheduler is but a minuscule event with temporary effects. A few minor inaccuracies are negligible and do not influence the overall scheduling process negatively. Note, that with the same reasoning, the criticism of rank reversal in the AHP can be disregarded with respect to scheduling. In a worst case scenario, rank reversal lets the scheduler select the second best option in a given

situation; generally, it goes unnoticed because none of the reversed options would have been chosen anyway. Thus, we take the illustrated way and join opposing opinions into a common tree. Nevertheless, our application of the AHP does obviously not stand up to Saaty's rigorous requirements. Hence, we only refer to it as *heuristic* scheduling based on the AHP.

Historically, scheduling algorithms have not allowed the participation of additional decision makers. Therefore, resource providers are reluctant to allow users and grid community to take part in the scheduling, and they will probably not accept a design where those parties have an equal share in the decision. A solution to this dilemma are the weights that join the differing opinions at the root of the overall decision tree; we call them *root weights*. Normalization of the partial utilities at every node, as prescribed in Saaty's AHP, equalizes the judgements of user, resource provider, and grid community. Stated alternatively, after normalization all statements are considered as equally strong and of identical priority. Next, the weights at root of the decision tree apply and define the share of participation granted to a particular party. By reserving the definition of these weights to the providers of the specific resources where a decision is made, the power of control over their resources is returned to them. In addition, this step guarantees that even in a grid environment the individual resource providers preserve their autonomy.

Still, the scheduling is less a dictatorship than a "controlled democracy". While the providers have the authority to define the root weights, we assume that users and grid community will closely examine the share of participation granted to them. The weight allocated to the grid community will probably become a major point of discussion and political controversy in the formation of a computing grid. Though users will lack the power to pressure for specific weights, they will always be able to vote with their feet. With different resources available to them, they can choose the resource that most closely matches their expectations. Hence, we expect the exact values of the root weights to become crucial sales arguments.

Altogether, in our opinion the Analytic Hierarchy Process provides for a unique approach to abandon inflexible structures and achieve a more dynamic scheduling design.

6 Utility Computation

MCDM problems as defined in Section 3 introduce two types of numerical values that are distinct and yet closely related: weights and utilities. Recall that weights are used to represent the importance of a criterion, while utilities express the "usefulness" of an option. Both types of values constitute relative quantities that are only meaningful in comparison of related items. As such, they represent abstract, subjective assessments of the presumed impact a criterion or option has on the problem. But, despite their similarities, the concepts underlying weights and utilities are often quiet distinct.

A criterion is an abstract notion that helps a decision maker to structure the problem in his mind. Thus, its relative importance is subjective and almost always based on social factors such as experience, agreement, or "gut feeling". A criterion may be

based on a physical principle, but it may as well be just a thought and nothing more. E.g., the waiting time, as a criterion to evaluate a set of resource offers, is backed by the physical principle of time, whereas the personal preference for one computing resource over another is not directly quantifiable. Nevertheless, the scale on which the weight of either criterion is expressed is a ratio scale.

An option, on the other hand, can be an abstract notion or a concrete object. The utility of an option is only ever defined with respect to some fixed criterion. Depending on this criterion, the utility may be backed by either a physical unit scale, e.g., a waiting time of two hours vs. a waiting time of one hour, or a ratio scale, e.g., resource A is preferred twice as strongly as resource B. The possible presence of objective quantities marks the major difference to weights. However, the utility of an option is not directly specified by its value on the physical scale. An additional mapping takes place, that relates objective quantities to subjective quantities on a ratio scale. In general, the type of mapping function is arbitrary, but in practice a few notable types occur more frequently than others. Fig. 6 displays a selection of these more common mapping functions[†].

Mappings of the types depicted in Fig. 6a and 6b describe the most basic linear relationship between a measure on a unit scale and the utility on a ratio scale. The assumption behind both types is, that increasing the measure by a constant value will always, respectively, increase and decrease the utility by a fixed amount, too. Often, this is an approximation to the real interactions taking place. An example for a maximization criterion that can be approximated as a linear mapping is, e.g., the number of CPUs provided by a resource offer. Yet, assuming that more compute power will always increase the utility of an offer is generally fallacious because most applications do not scale indefinitely.

The type in Fig. 6c represents a linear relationship with an inner maximum. Once that maximum is reached, further increasing the underlying measure does actually decrease the utility of an option. Saaty [30] suggests temperature as an example for this mapping: low temperatures are uncomfortable and increasing the temperature creates a more enjoyable environment. But, at some point this process is reversed and, eventually, higher temperatures result in hostile conditions. Edwards and Barron [13] mention scratching as another example: it may be pleasant to scratch an itch, but too much scratching may lead to painful skin irritations.

A mapping of the type in Fig. 6d corresponds to a piecewise linear relationship with diminishing returns. Increasing the associated measure does indeed improve the utility. Yet, the amount of utility added by a fixed increment on the unit scale decreases. The reversed case, where the utility decreases with additional improvements on the measure scale, is not explicitly displayed. With respect to job scheduling, an example for a criterion with diminishing returns is, e.g., the aforementioned number of offered CPUs. In many cases, parallel applications scale well up to a number of cores that depends on the employed algorithm. After this number is reached, the impact of additional cores on the total compute time gets less and less. For a few problems, this criterion may even

[†]Edwards and Barron [13] mention the mapping types 6a, 6b, and 6c, too. They add the direct specification of an utility value as a fourth type.

Figure 6: Different types of functions mapping objective measures to subjective utilities.

prove to be a case of a mapping of the type in Fig. 6c.

The nonlinear mapping as displayed in Fig. 6e is just a representative of a large class of vastly different utility functions: parabolas, exponential functions, sine- and cosine-based mappings, and many more. We will defer the discussion of this type to Section 6.1, where we propose a mapping that is based on the hyperbolic tangent. Likewise, the stepwise mapping displayed in Fig. 6f is representative for the utility function discussed in Section 6.2.

The metascheduler which we propose is modular with respect to the utility mappings it supports. While any type of function can be added, we provide only two default implementations: a nonlinear mapping on the basis of the hyperbolic tangent and a configurable stepwise mapping. They cover most of the scenarios that can come up in practice. Both procedures will be explained in the next sections. Note, that the functions described above do not produce normalized utilities per se. MCDM algorithms that require such values, such as Saaty's AHP, need to perform an additional normalization step before the utilities can be used.

6.1 Evaluation with the Hyperbolic Tangent

The utility of an option is generally relative and depends on the availability of other alternatives. With the exception of a few situations where external influences control his preferences, a decision maker will define his utility values based on a holistic view of the options he is offered. An overabundance or scarcity of a particular facet may, respectively, devalue or revalue options that provide that feature. That is, the utility of the exact same measure on the unit scale may be perceived differently depending on what other options have to offer. Theoretically, this problem is solved by algorithms like AHP, MAHP, and ROC in such a way that the measure is never directly mapped to a utility value. Instead, the decision maker provides subjective pairwise comparisons or rankings even though objective measures exist. Hence, a measured value just serves as information to the human user. Saaty notes that "in general, the numbers obtained from such a scale are merely stimuli for the memory [...] and have no intrinsic significance" [30, page 11].

Unfortunately, the environment in which a distributed metascheduler is deployed does not allow for interactive evaluation. Metascheduler proxies make their decisions indirectly and without immediate impetus from either resource provider, grid community, or user. A call back to any of these parties would have to occur asynchronously and would place the associated job in a pending state. Further processing of the job could only take place after a reply had been received. In the majority of cases, this behavior is unacceptable. It prevents the metascheduling from smoothly balancing the load and migrating job descriptions between resources. More so, it would require additional staff just to support a task that is supposed to be carried out by the metascheduler automatically.

Hence, it is essential to provide an algorithm that can dynamically assign utility values to options. Fig. 7 shows an example for the challenges faced in the construction of such an algorithm. It presents three sets of 10 options each with arbitrarily assigned

example	measures				
(a)	1.82	1.89	2.45	4.13	4.69
	5.60	5.81	6.93	*7.00*	*11.00*
(b)	*7.00*	7.50	7.80	8.40	8.64
	8.82	9.42	9.96	10.92	*11.00*
(c)	*7.00*	*11.00*	11.56	13.16	13.32
	13.64	15.08	15.32	17.40	17.56

Table 6: Random measures used in Fig. 7 and Fig. 8.

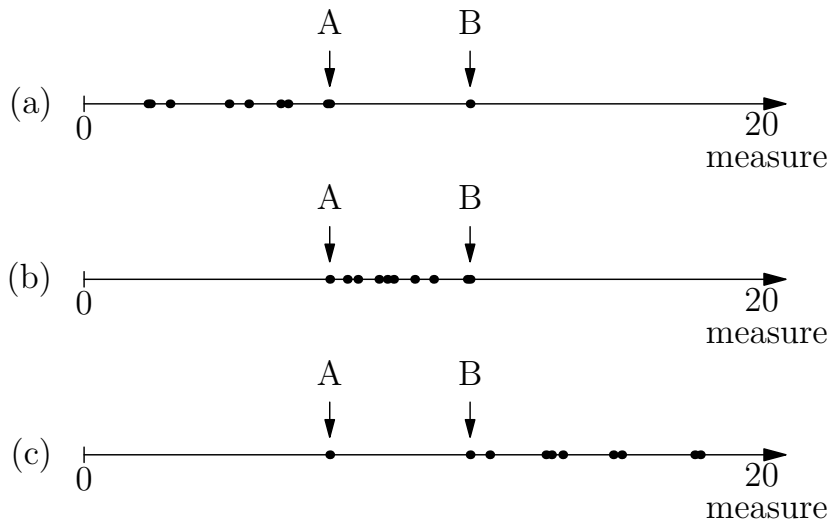


Figure 7: Exemplary cases showing the holistic perception of utility. — The dots mark the measures on the unit scale of distinct options. The two dots A and B represent a pair of measures that is identical in each set. Nevertheless, the utility of this pair is perceived differently depending on the measures exhibited by the other options in the set.

measures (see Table 6). Two measures A and B appear in every set and are highlighted with small arrows. Despite their identity, the utilities of the associated options will be perceived differently by a human decision maker. In case (a) the two measures are larger than all other values occurring; conditional to the type of criterion, i.e., minimizing or maximizing, the utility of A and B will be either very low or very high. In the second case (b) the two measures are positioned in a rather dense cluster with the remaining values. Here, the corresponding utilities will be slightly above and slightly below the average. Finally, case (c) depicts the opposite situation of case (a). Consequently, the utilities of the options associated with A and B are reversed.

The utility of an option depends on the option's relative position with respect to the bulk of remaining alternatives. In fact, we see the three possible regions of measures that need to be considered in determining the utilities:

1. A significant agglomeration of measures that contains a majority of the distributed value. We call this region the *dominant cluster* of measures.

2. The range next to the dominant cluster which contains all the *less* preferable values. Considering a maximization criterion this would mean the region ranging from the absolute zero to the left boundary of the dominant cluster; considering a minimization criterion, on the other hand, this would mean the region stretching from the right boundary of the dominant cluster to infinity.

3. The range of values *more* preferable than the values in the dominant cluster. Similarly, this would include the portion of the x-axis to the left of the dominant cluster for minimization criteria, and, conversely, to the right of the dominant cluster for maximization criteria.

It is reasonable to expect the utility values of the more and less preferred options to be, respectively, above and below the mean utility value. Likewise, the dominant cluster contains the options of average quality; hence, they can be expected to be mapped to values of approximately mean utility. Yet, assigning each such option exactly the same mean utility would neglect what small differences the measures show. For this reason, we design the mapping such that it spreads the utilities of the dominant cluster to a significant portion of the range of mean utility values. At the same time, we consider it desirable to bound the maximum and minimum utility of more and less preferable alternatives. Thus, extreme outliers can not distort the range of utilities. With respect to the dominant cluster, this would compress the image of the utility mapping to a small range of values and hide difference between these average options.

Moreover, we consider this to be corresponding to the way a human decision maker would think. Assume that the majority of options sits in a clear-cut cluster and a few outliers are positioned at various distances to the left and right of the cluster. In this situation, the outliers are, respectively, either distinctly better or worse than the average option. The exact benefit they provide when compared between themselves is less relevant than the difference compared to the majority of remaining options. Once a certain distance to the cluster is reached, further improvement or decline has no additional significant impact on the utility of an option. The utilities must simply reflect the fact that

such outliers should be, respectively, absolutely preferred or rejected in comparison to the options in the dominant cluster.

To transfer these ideas into an algorithm, the first step in the automatic evaluation procedure is to determine the dominant cluster. We define a clustering of a set of measures and the dominance of a cluster as follows:

Definition 2. *Let $R = \{r_1, r_2, \ldots, r_n\} \subset \mathbb{R}$ be an arbitrary set of real values and $C = (C_1, C_2, \ldots, C_m)$ be a partition of R, that is, a sequence of nonempty, pairwise disjoint subsets of R such that each element of R is in exactly one of these subsets. Then, we call C a* clustering of *R with the clusters C_i where $1 \leq i \leq m$ if, and only if,*

$$i < j \quad \Longrightarrow \quad \forall x \in C_i \, \forall y \in C_j : \; x < y,$$

*that is, the subsets are strictly successional. Further, we call a cluster C_i s-*dominant *if, and only if, it contains at least a fraction s of the total number of values, $|C_i| > s \cdot |R|$.*

The definition allows for more than one s-dominant cluster with $s < 0.5$ to exist in a clustering of measures. For $s \geq 0.5$ the dominant cluster is unique and we can speak of *the* dominant cluster. Which of the dominant clusters in the former case is subsequently used to define the utility mapping depends on the type of criterion considered. For a maximization criterion we choose the cluster with the largest measures, that is, the right-most dominant cluster. The remaining dominant clusters do contain a significant portion of the elements, yet these elements are all less preferred than the elements in the chosen dominant cluster. Therefore, they are dismissed as agglomerations of inferior utility. Conversely, for a minimization criterion we select the left-most dominant cluster with the smallest measures.

Algorithm 1 describes how the complete set of s-dominant clusters can be determined in a constructive process. Its sequence of steps depends on the definition of a distance function δ that returns the distance between two clusters of values. We have had good results with *average linkage clustering*, where the distance is defined as

$$\delta(A, B) := \frac{1}{|A| \cdot |B|} \sum_{a \in A} \sum_{b \in B} |a - b|,$$

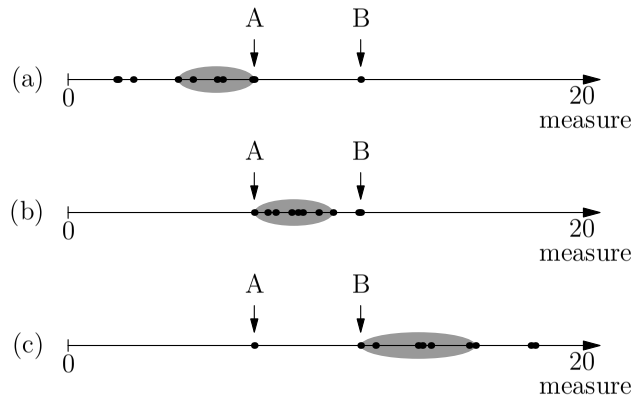
but, in general, we believe the particular distance function to be of secondary significance. In our metascheduler, the clustering is implemented as a separate module such that the whole clustering algorithm and the distance function can both be easily exchanged. The results of clustering with Algorithm 1 are illustrated in Fig. 8a which repeats the examples from Fig. 7 but highlights the 0.5-dominant clusters.

Further, the effect of the automatic utility mapping, which we will describe in the following, is depicted in the subfigures (b), (c), and (d) of Fig. 8 for a minimization criterion. The dominant cluster is translated into a rectangle which we call the *control box* of the utility mapping. Let $m_1, m_2, \ldots, m_k$ be the measures in the dominant cluster, and let n be the number of all measures. Width and horizontal position of the rectangle are matched to the exact width and position of the dominant cluster, that is, $\Delta m := |m_k - m_1|$ and the center of the control box is placed at $\frac{1}{2}(m_1 + m_k)$.

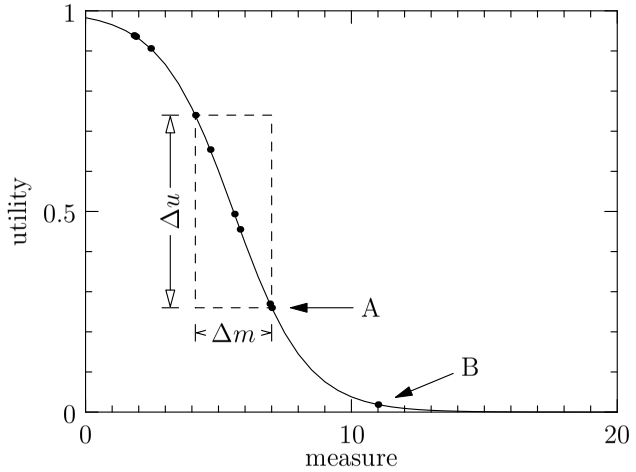
Algorithm 1 Find s-dominant clusters of R

Require: $R = \{r_1, r_2, \ldots, r_n\} \subset \mathbb{R}$ and $0 \leq s' = s \cdot |R| < n$
Ensure: C contains all s-dominant clusters of R

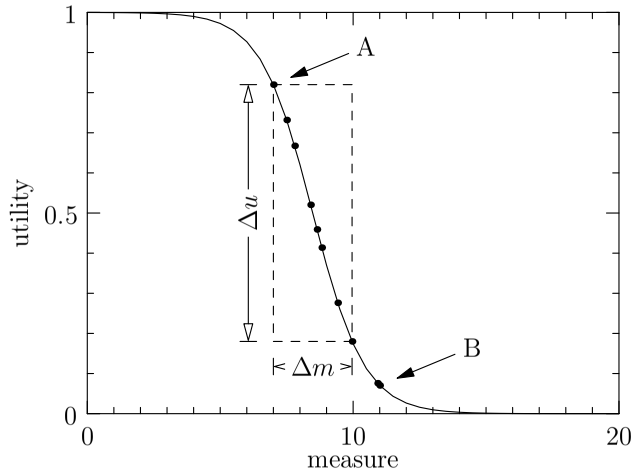
1: $C \leftarrow \big\{\{r_1\}, \{r_2\}, \ldots, \{r_n\}\big\}$
2: {loop until an s-dominant cluster has accumulated}
3: **while** $\neg\exists C_D \in C : \ |C_D| > s'$ **do**
4: {find minimum distance between any two clusters}
5: $d \leftarrow \infty$
6: **for all** $A, B \in C$ where $A \neq B$ **do**
7: $d \leftarrow \min(d, \delta(A, B))$
8: **end for**
9: {merge all clusters that have minimum distance}
10: **for all** $C_1, \ldots, C_k \in C$ where $\delta(C_i, C_{i+1}) = d$ for all $1 \leq i < k$ **do**
11: $C \leftarrow \{C_1 \cup \cdots \cup C_k\} \cup C \setminus \{C_1, \ldots, C_k\}$
12: **end for**
13: **end while**
14: {remove all non-s-dominant clusters}
15: **for all** $A \in C$ where $|A| \leq s'$ **do**
16: $C \leftarrow C \setminus \{A\}$
17: **end for**
18: **return** C



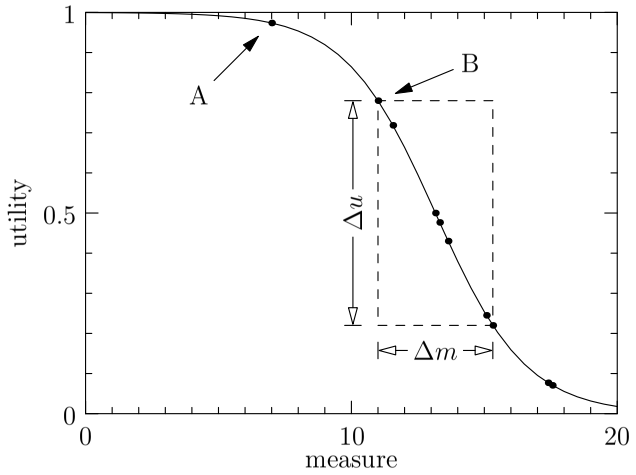
(a) Examples of s-dominant clusters constructed by Algorithm 1. — The random measures are provided in Table 6. For each set of measures, the 0.5-dominant cluster is highlighted by a gray ellipse.



(b) Automatic utility function corresponding to example (a) of Fig. 8a.



(c) Automatic utility function corresponding to example (b) of Fig. 8a.



(d) Automatic utility function corresponding to example (c) of Fig. 8a.

Figure 8: Effect of the s-dominant clusters on the automatically determined utilities.

Height and vertical position of the rectangle are chosen to encompass a significant portion of the mean utility values. For this reason, the vertical position is such that the center of the rectangle is mapped to the utility value $1/2$. The height Δu is expressed proportional to the size of the dominant cluster. We define the height of the control box to be $\Delta u := 0.8k/n$. The factor 0.8 is chosen somewhat arbitrarily to allow for some margin at the top and bottom of the control box. The reason therefore is that the dominant cluster can sometimes contain all measures, i.e., $k = n$; even in this case, the height of the control box must satisfy $\Delta u < 1$, that is, it must not be the full height of the image of the utility mapping. Otherwise, our utility mapping based on the hyperbolic tangent would leave the intended range $(0, 1)$ of valid utility values.

In the following two subsections, we will derive the explicit utility functions to be employed for the automatic evaluation.

6.1.1 Maximization Criteria

Above, we have described the considerations that went into the design of an automatic utility mapping. Specifically, we mentioned that the utility values should be bounded above and below to avoid misrepresentation of minor differences in the dominant cluster of options. For this reason, we opted to base our utility mapping on the hyperbolic tangent which asymptotically tends to $y = -1$ and $y = 1$ when x approaches, respectively, negative and positive infinity. Hence, the behavior of the curve is suitable for a maximization criterion, where larger input values have to yield larger output values.

As we have previously described, the MCDM algorithms require that utility values are positive real numbers. Hence, the hyperbolic tangent needs to be scaled and shifted to map its image from $(-1, 1)$ into the range $(0, 1)$ of valid utilities, i.e., $\frac{1}{2}(1+\tanh(x))$. Using the identity

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

we have

$$\begin{aligned} u(x) &= \frac{1}{2}\left(1 + \tanh\left(x\right)\right) \\ &= \frac{1}{2}\left(1 + \frac{e^x - e^{-x}}{e^x + e^{-x}}\right) \\ &= \frac{1}{2}\left(\frac{e^x + e^{-x}}{e^x + e^{-x}} + \frac{e^x - e^{-x}}{e^x + e^{-x}}\right) \\ &= \frac{1}{2}\frac{2e^x}{e^x + e^{-x}} \\ &= \frac{e^x}{e^x\left(1 + e^{-2x}\right)} \\ &= \frac{1}{e^{-2x} + 1} \end{aligned}$$

as a starting point for the real mapping function.

The hyperbolic tangent transitions through the mean utility values of its image $(0, 1)$ for a rather narrow range of input values. Specifically, the middle 98.7% of output values are produced by the input values in the range $[-2.5, 2.5]$. Thus, the function needs to be further scaled and shifted to match its range of transition with the dominant cluster determined previously. For this purpose, we introduce four control parameters Δm, Δu, x_e, and u_e as well as two auxiliary parameters λ and δ. The auxiliary parameters will be expressed in terms of the control parameters and are meant solely to support a well-arranged representation of the utility function as

$$u_C(x) := \frac{1}{e^{\lambda(x-\delta)} + 1}. \tag{22}$$

They will be defined such that the following two conditions are satisfied for the curve of the automatic utility mapping:

1. The curve runs through the *lower left and upper right* corners of a rectangle R that is defined by its width Δm, its height Δu, and its center that has been placed in the point of symmetry of the hyperbolic tangent.

2. The curve runs through the point (x_e, u_e).

The first condition forces the scaling of the hyperbolic tangent to match the extends of the rectangle R and the second condition fixes the absolute position of the curve. Obviously, the rectangle is supposed to coincide with the control box which we previously derived from the dominant cluster. For generality, we will disregard the specific constraints imposed upon the control box with respect to its vertical position.

Let (x_s, u_s) be the center of the control box and simultaneously the point of symmetry of the hyperbolic tangent. It is sufficient to ensure that the curve runs through the lower left corner of the rectangle R. Due to symmetry, the curve will then run through the upper right corner, too. First, we determine the coordinates of the point of symmetry: x_s and u_s. It is known, that this point is the vertical midpoint of the image of the starting function $u(x)$, i.e., $u_s = 1/2$. Hence, we have

$$\begin{aligned}
& u_C(x_s) = 1/2 \\
\Longrightarrow \quad & \frac{1}{e^{\lambda(x_s-\delta)} + 1} = 1/2 \\
\Longrightarrow \quad & e^{\lambda(x_s-\delta)} = 1 \\
\Longrightarrow \quad & \lambda\,(x_s - \delta) = 0 \\
\Longrightarrow \quad & x_s = \delta.
\end{aligned}$$

Further, the lower left corner of the rectangle R can be expressed as $(x_s - \frac{1}{2}\Delta m, u_s - \frac{1}{2}\Delta u)$. With $(x_s, u_s) = (\delta, 1/2)$, we have

$$\begin{aligned}
& u_C\left(x_s - \frac{1}{2}\Delta m\right) = u_s - \frac{1}{2}\Delta u \\
\Longrightarrow \quad & u_C\left(\delta - \frac{1}{2}\Delta m\right) = \frac{1}{2} - \frac{1}{2}\Delta u
\end{aligned}$$

$$
\begin{aligned}
\Longrightarrow \quad & \frac{1}{e^{\lambda\left(\left(\delta-\frac{1}{2}\Delta m\right)-\delta\right)}+1} = \frac{1}{2}-\frac{1}{2}\Delta u \\
\Longrightarrow \quad & \frac{1}{e^{-\frac{1}{2}\lambda\Delta m}+1} = \frac{1-\Delta u}{2} \\
\Longrightarrow \quad & e^{-\frac{1}{2}\lambda\Delta m}+1 = \frac{2}{1-\Delta u} \\
\Longrightarrow \quad & e^{-\frac{1}{2}\lambda\Delta m} = \frac{2}{1-\Delta u}-\frac{1-\Delta u}{1-\Delta u} \\
\Longrightarrow \quad & -\frac{1}{2}\lambda\Delta m = \ln\left(\frac{1+\Delta u}{1-\Delta u}\right) \\
\Longrightarrow \quad & \lambda = -\frac{2}{\Delta m}\ln\left(\frac{1+\Delta u}{1-\Delta u}\right).
\end{aligned}
$$

Thus, we have an explicit representation of the auxiliary scaling factor λ in terms of the parameters defining the control box.

Similarly, we can take the second condition into account. Applying the mapping to x_e, we obtain

$$
\begin{aligned}
& u_C(x_e) = u_e \\
\Longrightarrow \quad & \frac{1}{e^{\lambda(x_e-\delta)}+1} = u_e \\
\Longrightarrow \quad & e^{\lambda(x_e-\delta)}+1 = \frac{1}{u_e} \\
\Longrightarrow \quad & e^{\lambda(x_e-\delta)} = \frac{1-u_e}{u_e} \\
\Longrightarrow \quad & \lambda\left(x_e-\delta\right) = \ln\left(\frac{1-u_e}{u_e}\right) \\
\Longrightarrow \quad & \delta = x_e-\frac{1}{\lambda}\ln\left(\frac{1-u_e}{u_e}\right).
\end{aligned}
$$

Note, that the auxiliary shifting parameter δ depends on the second auxiliary parameter λ. To avoid this dependency and gain independent scaling and shifting parameters, we set

$$\delta = x_e-\frac{1}{\lambda}\delta',$$

where

$$\delta' := \ln\left(\frac{1-u_e}{u_e}\right).$$

Replacing δ in Eq. 22 with this equality, translates the original representation of the

utility mapping $u_C(x)$ into

$$\begin{aligned} u_C(x) &= \frac{1}{e^{\lambda(x-\delta)}+1} \\ &= \frac{1}{e^{\lambda\left(x-x_e+\frac{1}{\lambda}\delta'\right)}+1} \\ &= \frac{1}{e^{\lambda x-\lambda x_e+\delta'}+1} \\ &= \frac{1}{e^{\lambda(x-x_e)+\delta'}+1}. \end{aligned}$$

Finally, we keep this new representation and define the utility mapping for maximization criteria as

$$u_C(x) = \frac{1}{e^{\lambda(x-x_e)+\delta'}+1}, \tag{23}$$

where

$$\lambda = -\frac{2}{\Delta m}\ln\left(\frac{1+\Delta u}{1-\Delta u}\right)$$

and

$$\delta' = \ln\left(\frac{1-u_e}{u_e}\right).$$

6.1.2 Minimization Criteria

For minimization criteria we proceed analogously to the construction given in the previous subsection on maximization criteria. The function is based on a shifted and scaled hyperbolic tangent, too; however, we note that in this case smaller measures must lead to larger utility values. Hence, we mirror the hyperbolic tangent at the axis $y = 0$ before we fit its image to the range $(0, 1)$. We obtain the start mapping

$$\begin{aligned} u'(x) &= 1 - \frac{1}{2}(1+\tanh(x)) \\ &= 1 - \frac{1}{e^{-2x}+1}. \end{aligned}$$

and, again, introduce two auxiliary parameters, this time called μ and ϵ, that allow us to express the utility mapping as

$$u'_C(x) := 1 - \frac{1}{e^{\mu(x-\epsilon)}+1}. \tag{24}$$

Reflecting the hyperbolic tangent has resulted in a curve that passes through the opposing corners of the control box when compared with a maximization criterion. Specifically, we require the curve to satisfy the following two conditions:

1. The curve runs through the *upper left and lower right* corners of a rectangle R that is specified by its width Δm and its height Δu and whose center is placed in the point of symmetry of the curve.

2. The curve runs through the point (x_e, u_e).

Note, that the first condition can alternatively be replaced with a condition that requires the curve only to pass through the upper left corner of R. Due to the rectangle's position centered at the point of symmetry, the curve will automatically pass through the lower right corner, too.

Further, the upper left corner of R can be expressed relatively to the rectangle's center (x'_s, u'_s) as $(x'_s - \frac{1}{2}\Delta m, u'_s + \frac{1}{2}\Delta u)$. We omit the derivation of $(x'_s, u'_s) = (\epsilon, 1/2)$ as it is identical to the reasoning for maximization criterion. Instead, we directly proceed to insert the point into Eq. (24) and obtain

$$
\begin{aligned}
& u'_C\left(x'_s - \frac{1}{2}\Delta m\right) = u'_s + \frac{1}{2}\Delta u \\
\Longrightarrow \quad & u'_C\left(\epsilon - \frac{1}{2}\Delta m\right) = \frac{1}{2} + \frac{1}{2}\Delta u \\
\Longrightarrow \quad & 1 - \frac{1}{e^{\mu\left(\left(\epsilon - \frac{1}{2}\Delta m\right) - \epsilon\right)} + 1} = \frac{1}{2} + \frac{1}{2}\Delta u \\
\Longrightarrow \quad & \frac{1}{e^{-\frac{1}{2}\mu\Delta m} + 1} = \frac{1}{2} - \frac{1}{2}\Delta u \\
\Longrightarrow \quad & \frac{1}{e^{-\frac{1}{2}\mu\Delta m} + 1} = \frac{1 - \Delta u}{2} \\
\Longrightarrow \quad & e^{-\frac{1}{2}\mu\Delta m} + 1 = \frac{2}{1 - \Delta u} \\
\Longrightarrow \quad & e^{-\frac{1}{2}\mu\Delta m} = \frac{2}{1 - \Delta u} - \frac{1 - \Delta u}{1 - \Delta u} \\
\Longrightarrow \quad & -\frac{1}{2}\mu\Delta m = \ln\left(\frac{1 + \Delta u}{1 - \Delta u}\right) \\
\Longrightarrow \quad & \mu = -\frac{2}{\Delta m}\ln\left(\frac{1 + \Delta u}{1 - \Delta u}\right).
\end{aligned}
$$

Likewise, the second condition allows us to derive

$$
\begin{aligned}
& u'_C(x_e) = u_e \\
\Longrightarrow \quad & 1 - \frac{1}{e^{\mu(x_e - \epsilon)} + 1} = u_e \\
\Longrightarrow \quad & \frac{1}{e^{\mu(x_e - \epsilon)} + 1} = 1 - u_e \\
\Longrightarrow \quad & e^{\mu(x_e - \epsilon)} + 1 = \frac{1}{1 - u_e}
\end{aligned}
$$

$$
\begin{aligned}
\implies \quad & e^{\mu(x_e-\epsilon)} = \frac{1}{1-u_e} - \frac{1-u_e}{1-u_e} \\
\implies \quad & e^{\mu(x_e-\epsilon)} = \frac{u_e}{1-u_e} \\
\implies \quad & \mu\left(x_e-\epsilon\right) = \ln\left(\frac{u_e}{1-u_e}\right) \\
\implies \quad & \epsilon = x_e - \frac{1}{\mu}\ln\left(\frac{u_e}{1-u_e}\right).
\end{aligned}
$$

We define

$$\epsilon' := \ln\left(\frac{u_e}{1-u_e}\right),$$

and note that $\mu = -\lambda$ and, because $\ln\left(\frac{1}{x}\right) = -\ln(x)$, also $\epsilon' = -\delta'$. Hence, we can transform Eq. (24) like in the previous section and obtain

$$u'_C(x) = 1 - \frac{1}{e^{\lambda(x-x_e)-\delta'}+1}.$$

Due to the equality

$$
\begin{aligned}
1 - \frac{1}{e^E+1} &= \frac{e^E+1}{e^E+1} - \frac{1}{e^E+1} \\
&= \frac{e^E}{e^E+1} \\
&= \frac{e^E}{e^E\left(1+e^{-E}\right)} \\
&= \frac{1}{e^{-E}+1},
\end{aligned}
$$

we can represent a concise formula for the utility mapping for minimization criteria as

$$u'_C(x) = \frac{1}{e^{-\lambda(x-x_e)+\delta'}+1}, \tag{25}$$

where

$$\lambda = -\frac{2}{\Delta m}\ln\left(\frac{1+\Delta u}{1-\Delta u}\right)$$

and

$$\delta' = \ln\left(\frac{1-u_e}{u_e}\right).$$

This representation mirrors the representation in Eq. 23.

6.1.3 Practical Considerations

In the theoretical derivation of the utility mapping, we presumed knowledge of an explicit point (x_e, u_e) which was required to determine the exact position of the utility curve. In practice, it is favorable to consolidate this point with the center of the control box and to vertically center the control box in the image range $(0, 1)$. As a consequence, $u_e = 1/2$ is predefined and

$$\begin{aligned}\delta' &= \ln\left(\frac{1-u_e}{u_e}\right)\\ &= \ln\left(\frac{1-\frac{1}{2}}{\frac{1}{2}}\right)\\ &= \ln(1)\\ &= 0,\end{aligned}$$

which simplifies Eq. (23) and (25) slightly. We already explained above how to define the remaining control parameters with respect to the dominant cluster.

6.2 Direct Mapping

Using the hyperbolic tangent to compute utility values is well-suited for criteria that are based on continuous measures such as load or waiting time; for these criteria, the continuous utility function reflects the shape of the input values. A few discretely measured aspects, such as the length of the job queue, map equally well to the continuous hyperbolic tangent. Though the input measures are discrete, there is little difference between the utility of consecutive input values. However, sometimes utilities are simply not supposed to transition smoothly because successive measures represent distinctly unique features.

One such criterion that occurs frequently with scheduling is the user ID. Users of a computing resource are customarily associated with a unique integer identifier that is used as their representation by the operating system. Consecutive user IDs express no relationship between the associated users; yet, often ranges of user IDs are logically associated with generic groups of users. On a Debian Linux system, e.g., the root user has the ID zero, system accounts reserved by the operating system range from one to 99, unreserved system accounts have values of up to 999, and regular users are identified by IDs starting with 1,000 and ending at 29,999; the ID 65,534, the second to last value that fits into a 16 bit integer, represents the user “nobody” which has no permissions at all. It is common for administrators to subdivide the user IDs in the range 1,000–29,999 and associate the subranges with specific local user groups.

The mapping of user IDs to utilities can be instrumented to incorporate individual priorities of the jobs of users into the decision. Yet, consecutive user IDs do not correspond to small changes in their priorities; more often, whole ranges associated with different user groups are supposed to be mapped to fixed utility values (see Fig. 9). To

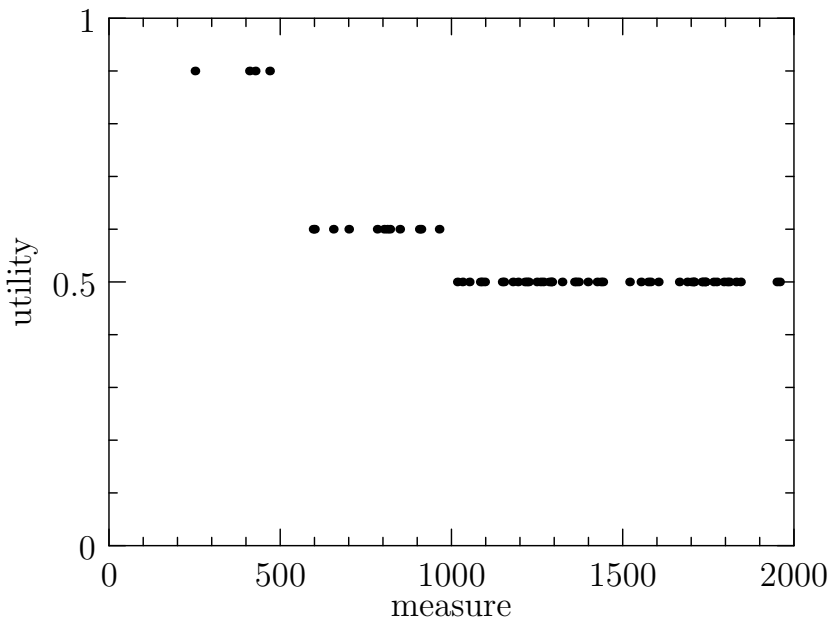


Figure 9: Exemplary mapping of ranges of measures to fixed utilities. — In this figure the criterion *user ID* is divided into three ranges which are mapped to specific utilities: the range $[250, 500)$ contains the IDs of administration and staff, the range $[500, 1000)$ contains local or favored user IDs, and the range $[1000, 2000)$ contains the remaining grid users. The utility reflects the priority of the users in each group.

achieve this kind of mapping, the hyperbolic tangent must be abandoned for a step function that produces discrete output values. Obviously, it is impossible to automatically determine the utility values because this mapping is an artificial construct.

Even though this concept of representing unique entities with numeric values is prevalent in computer systems it is not restricted to this domain, e.g., humans are identified by their social security number, goods are labeled with barcodes, and animals may be injected with a RFID chip. However, in real world multicriteria decision making the numeric ID is rarely mapped directly to utilities. Here, the entities are seen as unique individuals. Determining their utility is precisely why procedures such as the Analytic Hierarchy Process or Rank-order Centroid have been developed. Still, for the reasons given in the previous section, pairwise comparisons and rankings can not be queried interactively in a metascheduler.

Therefore, we provide a second mapper module next to the hyperbolic tangent, where ranges of input values can be mapped to specific utilities. Ranges can be expressed as lower bounded, upper bounded, bounded in both directions, or as individual values. Bounds can be given as exclusive or inclusive, and an epsilon value can be added to exact values to express fuzziness. This design emulates the possibilities supported in the specification of the Job Submission Description Language (JSDL). Likewise, the mapping is specified as an XML fragment in the configuration of the metascheduler or embedded in a job's JSDL description. Any positive numeric value can be used as a utility; the resulting mapping is normalized before it is employed in the AHP.

However, while a direct mapping is supposedly easy to configure, it can also be misused. Its major disadvantage is, that its non-continuous utility function may look unreasonably reasonable at first sight. At the boundaries of the intervals, small changes in the measure will produce large changes in the corresponding utilities. In general, these differences in utility between almost indistinguishable alternatives do not reflect the preferences of a decision maker correctly. Take, e.g., a stepwise mapping of waiting time intervals: an offer that promises a short waiting time of up to two hours is mapped to a high utility, and remaining offers are penalized with a low utility with respect to this criterion. Now let two offers exhibit, respectively, 119 minutes and 121 minutes of waiting time. Except for a deadline that may warrant such a strong demarcation, the two offers will generally be of similar utility to a user. Hence, the arbitrary boundary placed at exactly 120 minutes produces a distorted image of the real interests. For this reason, any use of the direct mapping module should be carefully circumstantiated.

7 Conclusions and Outlook

In this technical report, we have described a generic scheduling heuristic that incorporates arbitrary criteria by different decision makers. In general, the heuristic can be used to prioritize jobs in local batch queues, service level agreements in grids, or job descriptions and resource offers in metaschedulers. We illustrated how the heuristic can be employed in a distributed architecture to create a fault-tolerant and scalable metascheduler. The design represents a politically acceptable method of gridwide decision making

as it offers a configurable share of participation to users, resource providers, and grid community. Still, it respects the reservations by resource providers to relinquish control over a resource to other parties; to appease this most influential group of decision makers, it retains the right to specify the power of each participant to them. Finally, we presented a procedure based on the hyperbolic tangent to dynamically determine the relative utility of a set of options for most criteria.

Working on the algorithms of our metascheduler, we collected first-hand experience with local resource management systems of HPC sites and clusters. Hence, we were able to gain detailed insights as to how the input queues of those systems interact with the scheduling algorithms to influence the load of a resource. Further, we witnessed the political issues that dominate in national and international grid projects such as the German D-Grid[†] and the European DEISA grid[‡]. In cooperation with Sommerfeld and Richter [37], we have isolated four intrinsic problems that will have to be solved before metascheduling can become a fully working feature of computing grids.

First, metaschedulers generally have no direct control over the scheduling processes at grid sites. They use the interfaces provided by a grid middleware to monitor resources, access job queues, and submit or extract jobs. However, this also means that from the point of view of local resource management systems, the metaschedulers assume only the role of power users. Accordingly, they have to compete with regular users over access to the resources, and, as a consequence, they can not generate optimal schedules.

Second, the lack of control entails that the scheduling policy of the site scheduler is generally not in concord with the goals of the grid scheduler. The prioritization of the jobs queued in a resource management system is performed by the system itself; hence, the decisions made by the metascheduler are always of subordinate priority. Scheduling directions given to the metascheduler can only be implemented on a best effort basis.

Third, sensitive metrics that describe the state of a resource are usually not available to regular users. Generally, this is a political decision as well as a technical issue. Resource providers are reluctant to disclose the utilization of their resources, so as not to give competitors an advantage over them. Particularly publicly funded research institutes fear the loss of monetary support if low utilization of their resources was to be publicized. From our experience, the only statistics commonly available are the number of running and waiting jobs, yet some sites do not even disclose this information. To conclude, local schedulers currently do not offer sufficient data to derive a good schedule.

Finally, predicting the future performance of a resource based on current indicators is a delicate task. Resources are highly utilized and waiting times for computing jobs can range from minutes to hours. For most jobs, the time spend in a job queue considerably exceeds the actual execution time. Hence, the prediction of queue waiting times is a significant factor when the utility of an offered resource has to be determined. Yet, the behavior of queue waiting time and queue length over time is generally chaotic, and

[†]D-Grid Initiative, http://www.d-grid.de

[‡]Distributed European Infrastructure for Supercomputing Applications, http://www.deisa.org

these values can vary by a factor of thousand within minutes. However, scheduling always relies on such predictions.

Therefore, future work will have to focus on finding solutions to these challenges. Besides, we will aim our attention at developing a discrete event simulation of the proposed scheduling heuristic. Thus, we hope to underline its benefits with respect to the criteria expressed in a hierarchy.

References

[1] J. Aczél and T. Saaty, "Procedures for synthesizing ratio judgements," *Journal of Mathematical Psychology*, vol. 27, no. 1, pp. 93–102, Mar. 1983. [Online]. Available: http://dx.doi.org/10.1016/0022-2496(83)90028-7

[2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, *Web Services Agreement Specification (WS-Agreement)*, Open Grid Forum, March 2007.

[3] F. Barron and B. Barrett, "Decision quality using ranked attribute weights," *Management Science*, vol. 42, no. 11, pp. 1515–1523, Nov. 1996. [Online]. Available: http://dx.doi.org/10.1287/mnsc.42.11.1515

[4] J. Barzilai, "Deriving weights from pairwise comparison matrices," *Journal of the Operational Research Society*, vol. 48, no. 12, pp. 1226–1232, Dec. 1997.

[5] ——, "Notes on the Analytic Hierarchy Process," in *Proceedings of the 2001 NSF Design and Manufacturing Research Conference*, Jan. 2001, pp. 1–6.

[6] J. Barzilai and B. Golany, "Deriving weights from pairwise comparison matrices: The additive case," *Operations Research Letters*, vol. 9, no. 6, pp. 407–410, 1990. [Online]. Available: http://dx.doi.org/10.1016/0167-6377(90)90062-A

[7] ——, "AHP rank reversal, normalization and aggregation rules," *INFOR*, vol. 32, no. 2, pp. 57–64, May 1994.

[8] J. Barzilai and F. Lootsma, "Power relations and group aggregation in the Multiplicative AHP and SMART," *Journal of Multi-Criteria Decision Analysis*, vol. 6, no. 3, pp. 155–165, 1997. [Online]. Available: http://dx.doi.org/10.1002/(SICI)1099-1360(199705)6:3<155::AID-MCDA131>3.0.CO;2-4

[9] J. Barzilai, W. Cook, and B. Golany, "Consistent weights for judgements matrices of the relative importance of alternatives," *Operations Research Letters*, vol. 6, no. 3, pp. 131–134, 1987. [Online]. Available: http://dx.doi.org/10.1016/0167-6377(87)90026-5

[10] A. Bayucan, R. Henderson, L. Jasinskyj, C. Lesiak, B. Mann, T. Proett, and D. Tweten, *Portable Batch System - Administrator Guide*, Numerical Aerospace

Simulation Systems Division, NASA Ames Research Center, August 1998, revision 1.1.12.

[11] V. Belton and T. Gear, "On a short-coming of Saaty's method of analytic hierarchies," *Omega*, vol. 11, no. 3, pp. 228–230, 1983. [Online]. Available: http://dx.doi.org/10.1016/0305-0483(83)90047-6

[12] *TORQUE admin manual*, Cluster Resources, Inc., 2008, version 2.3.

[13] W. Edwards and F. Barron, "SMARTS and SMARTER: Improved simple methods for multiattribute utility measurement," *Organizational Behavior and Human Decision Processes*, vol. 60, no. 3, pp. 306–325, dec 1994. [Online]. Available: http://dx.doi.org/10.1006/obhd.1994.1087

[14] M. Ehrgott, *Multicriteria Optimization*, 2nd ed. Berlin/Heidelberg, Germany: Springer, 2005.

[15] J. Finan and W. Hurley, "The Analytic Hierarchy Process: can wash criteria be ignored?" *Computers & Operations Research*, vol. 29, no. 8, pp. 1025–1030, 2002. [Online]. Available: http://dx.doi.org/10.1016/S0305-0548(00)00100-3

[16] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[17] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of job-scheduling strategies for grid computing," in *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*. London, UK: Springer, 2000, pp. 191–202.

[18] J. Heilgeist, T. Soddemann, and H. Richter, "Algorithms for job and resource discovery for the meta-scheduler of the DEISA grid," in *Advanced Engineering Computing and Applications in Sciences, 2007. ADVCOMP 2007. International Conference on*, Nov. 2007, pp. 60–66. [Online]. Available: http://dx.doi.org/10.1109/ADVCOMP.2007.4401899

[19] R. Henderson, "Job scheduling under the Portable Batch System," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Berlin/Heidelberg, Germany: Springer, 1995, vol. 949, pp. 279–294. [Online]. Available: http://dx.doi.org/10.1007/3-540-60153-8_34

[20] A. Hohmann and P. Deuflhard, *Numerical Analysis in Modern Scientific Computing: An Introduction*, 2nd ed., ser. Texts in Applied Mathematics. Berlin/Heidelberg, Germany: Springer, 2003, vol. 43.

[21] D. Jackson, Q. Snell, and M. Clement, "Core algorithms of the Maui scheduler," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Berlin/Heidelberg, Germany: Springer, 2001, vol. 2221, pp. 87–102. [Online]. Available: http://dx.doi.org/10.1007/3-540-45540-X_6

[22] M. Liberatore and R. Nydick, "Wash criteria and the Analytic Hierarchy Process," *Computers & Operations Research*, vol. 31, no. 6, pp. 889–892, 2004. [Online]. Available: http://dx.doi.org/10.1016/S0305-0548(03)00041-8

[23] J. McCaffrey and N. Koski, "Test run: competitive analysis using MAQIG," *MSDN Magazine*, October 2006. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc300812.aspx

[24] J. R. Miller, III, "The assessment of worth: a systematic procedure and its experimental validation," Ph.D. dissertation, Massachusetts Institute of Technology, 1966. [Online]. Available: http://hdl.handle.net/1721.1/13513

[25] H. Nakada, A. Takefusa, K. Ookubo, M. Kishimoto, T. Kudoh, Y. Tanaka, and S. Sekiguchi, "Design and implementation of a local scheduling system with advance reservation for co-allocation on the grid," in *Computer and Information Technology, 2006. CIT '06. The Sixth IEEE International Conference on*, Sept. 2006, pp. 65–65. [Online]. Available: http://dx.doi.org/10.1109/CIT.2006.71

[26] D. Olson and V. Dorai, "Implementation of the centroid method of Solymosi and Dombi," *European Journal of Operational Research*, vol. 60, no. 1, pp. 117–129, 1992. [Online]. Available: http://dx.doi.org/10.1016/0377-2217(92)90339-B

[27] M. Pirlot and P. Vincke, *Semiorders: Properties, Representations, Applications*. Springer, 1997.

[28] T. Saaty, "A scaling method for priorities in hierarchical structures," *Journal of Mathematical Psychology*, vol. 15, no. 3, pp. 234–281, Jun. 1977. [Online]. Available: http://dx.doi.org/10.1016/0022-2496(77)90033-5

[29] ——, "Absolute and relative measurement with the AHP. the most livable cities in the United States," *Socio-Economic Planning Sciences*, vol. 20, no. 6, pp. 327–331, 1986. [Online]. Available: http://dx.doi.org/10.1016/0038-0121(86)90043-1

[30] ——, "How to make a decision: the Analytic Hierarchy Process," *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, Sept. 1990. [Online]. Available: http://dx.doi.org/10.1016/0377-2217(90)90057-I

[31] ——, "Highlights and critical points in the theory and application of the Analytic Hierarchy Process," *European Journal of Operational Research*, vol. 74, no. 3, pp. 426–447, May 1994. [Online]. Available: http://dx.doi.org/10.1016/0377-2217(94)90222-4

[32] ——, *Mathematical Methods of Operations Research*. Mineola, NY, USA: Dover Publications Inc., 2004.

[33] T. L. Saaty and M. Takizawa, "Dependence and independence: from linear hierarchies to nonlinear networks," *European Journal of Operational Research*, vol. 26, no. 2, pp. 229–237, Aug. 1986. [Online]. Available: http://dx.doi.org/10.1016/0377-2217(86)90184-0

[34] T. Saaty and L. Vargas, "The legitimacy of rank reversal," *Omega*, vol. 12, no. 5, pp. 513–516, 1984. [Online]. Available: http://dx.doi.org/10.1016/0305-0483(84)90052-5

[35] R. Sakellariou and V. Yarmolenko, "Job scheduling on the grid: Towards SLA-based scheduling," in *High Performance Computing and Grids in Action*, ser. Advances in Parallel Computing, L. Grandinetti, Ed. Amsterdam, The Netherlands: IOS Press, 2008, vol. 16, pp. 207–222.

[36] T. Solymosi and J. Dombi, "A method for determining the weights of criteria: The centralized weights," *European Journal of Operational Research*, vol. 26, no. 1, pp. 35–41, 1986. [Online]. Available: http://dx.doi.org/10.1016/0377-2217(86)90157-8

[37] D. Sommerfeld and H. Richter, "A two-tier approach to efficient workflow scheduling in MediGRID," in *Grid-Technologie in Göttingen - Beiträge zum Grid-Ressourcen-Zentrum GoeGrid*, U. Schwardmann, Ed. Göttingen, Germany: GWDG, 2009, vol. 74, pp. 39–51. [Online]. Available: http://www.gwdg.de/forschung/publikationen/gwdg-berichte/gwdg-bericht-74.pdf

[38] Q. Wang, X. Gui, S. Zheng, and Y. Liu, "De-centralized job scheduling on computational grids using distributed backfilling: Research articles," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 14, pp. 1829–1838, Dec. 2006. [Online]. Available: http://dx.doi.org/10.1002/cpe.1032

[39] R. Whitaker, "Validation examples of the Analytic Hierarchy Process and Analytic Network Process," *Mathematical and Computer Modelling*, vol. 46, no. 7–8, pp. 840–859, Oct. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.mcm.2007.03.018

[40] V. Yarmolenko and R. Sakellariou, "An evaluation of heuristics for SLA based parallel job scheduling," in *Proceedings. 20th International Parallel and Distributed Processing Symposium*. Piscataway, NJ, USA: IEEE, 2006, p. 8 pp.