

A Practical Approach to Distributed Metascheduling

Janko Heilgeist^{*†}, Thomas Soddemann^{*}, and Harald Richter[†]

^{*}Fraunhofer SCAI, Sankt Augustin, Germany

Email: {janko.heilgeist, thomas.soddemann}@scai.fraunhofer.de

[†]Clausthal Technical University, Clausthal-Zellerfeld, Germany

Email: harald.richter@tu-clausthal.de

Abstract—The paper describes a metascheduler for high-performance computing (HPC) grids that is build upon a distributed architecture. It is modelled around cooperating peers represented by the local proxies deployed by each participating site. These proxies exchange job descriptions between themselves with the aim of improving user-, administration-, and grid-defined metrics. Relevant metrics can include, e.g., reduced job runtimes, improved resource utilization, and increased job turnover. The metascheduler uses peer-to-peer algorithms to discover under-utilized resources and unserved jobs. A selection is made based on a simplified variant of the Analytic Hierarchy Process that we adapted to the special requirements imposed by the Grid. It enables geographically distributed stakeholders to participate in the decision and supports dynamic evaluation of the necessary utility values. An exemplary decision making process is presented, where user and resource provider jointly decide upon the resource where a job will be computed. Finally, we identify four intrinsic problems that obstruct the implementation of metaschedulers in general.

Keywords-Grid computing; metascheduling; resource discovery; decision making

I. INTRODUCTION

The problem of optimally scheduling jobs across loosely coupled distributed compute resources is still to be solved. Products such as Gridway [2] or Platform's LSF [3, 4] promise to provide out of the box solutions. On closer examination, most of these solutions still have problems ingrained in their design. A major drawback from our point of view is that despite the fact that resources are distributed, some of them work in a centralized fashion and resemble a staging queue in a classical batch system. Others are somewhat distributed, but intrusive as far as the interaction with a site's local batch scheduling system is concerned.

Unlike batch schedulers, a metascheduler supports the exchange of job descriptions across the boundaries of different sites. Such a migration can arise either directly from a

user's explicit request for a remote resource or indirectly from a metascheduler's attempt to perform a grid-wide load-balancing. In the latter case, it is the metascheduler's responsibility to discover the best destination. Yet, existing batch schedulers provide no separate point of entry for metaschedulers. A metascheduler can therefore not control the underlying hardware but has to use the site's local batch schedulers — or, more commonly, a grid middleware. It has to extract the job description from a queue at the source site and submit it, customarily assuming the original user's identity, into a target queue. However, migration should be transparent to end users, who should, optimally, never notice that their computations were performed non-locally.

The architecture of a metascheduler can be designed to be either *centralized* or *distributed* [5]. A *centralized* metascheduler is controlled by a dedicated entity that is installed at a single site and has, typically, all the required information for a decision on whether, when and where to migrate a job. That is, it collects data on which sites participate in the grid, which hardware they provide, what the speed of their connection to the grid is, which load their hardware has to bear, and to which degree their queues are filled, etc.

On the other hand, in a *distributed* (or *decentralized*) architecture the metascheduler is split up into multiple independent instances that cooperate among each other. Each instance is separately deployed and represents its site in the grid, that is, it is responsible for all jobs entering and leaving its site via the grid. We call such an instance a *proxy* of the metascheduler. Naturally, a proxy has only limited information to act on compared to the centralized design, as it only gathers data locally and from its neighbors for performance reasons. It is therefore necessary to provide a proxy with sufficient additional input to perform its scheduling. This input can be obtained, e.g., by inter-proxy communication, by a shared pool of available jobs, or by overlapping the local job pools of adjacent sites [5, 6].

While the centralized concept of metascheduling is easier to design, implement, deploy, and maintain, its drawbacks nevertheless outweigh its benefits. We see three crucial

[†]Present address: QAware GmbH, Aschauer Str. 32, 81549 München, Germany; Email: janko.heilgeist@qaware.de

Extended version of the paper originally presented at the International Conference on Advanced Engineering Computing and Applications in Sciences, 2009 [1].

problems that need to be addressed: First, a centralized met scheduler always represents a single point-of-failure. Problems with the met scheduler will have an immediate impact on the grid because access to remote resources is disrupted. Failures like a broken Internet link separate parts of a grid from the central scheduler. Furthermore, a successful attack on the scheduler compromises a grid.

Second, the centralized design suffers from reduced scalability when faced with increasing demand. The met scheduler represents a serious bottleneck as it is solely responsible for the migration of jobs in the grid. As the number of sites, users, and jobs grows, it will become more and more difficult to collect all the information that is required to determine the optimal schedule. The common countermeasure to deploy multiple backup servers running the met scheduler software prolongs the decline in performance but leads to additional costs and complexities.

Finally, the largest problem of the centralized architecture is of a political nature. Grids span across multiple administrative zones, companies and institutions, countries, or continents. Each site has its own local policies that a central met scheduler knows nothing about. This can simply be to prefer local users contending for the limited resources or can be as strict as national laws to prevent certain user-groups from accessing HPC resources. Additionally, the willingness of administrators to relinquish control over their hardware to an external entity is unpredictable.

All of the above problems do not emerge in a distributed design. Resilience is increased because a proxy failure affects mostly its home site. Link failures don't disable the grid, as separate sub-grids will continue to function independently. The overall performance scales with the size of the grid as the computation of the schedule is distributed across the grid nodes. And finally, political issues are mitigated by each site's ability to configure its proxy independently.

However, with only limited information to act on, distributed met schedulers usually produce sub-optimal schedules. While it would be theoretically possible to achieve full information at every proxy through a complete data exchange between all sites, the cost of $O(n^2)$ is prohibitive and makes this idea infeasible. Determining good schedules is an algorithmic problem that is, as of now, best tackled by contenting oneself with approximations. Overall, we favor a distributed approach to met scheduling over the centralized design.

In the remaining sections of this paper, we will describe a distributed architecture that is currently being implemented for the Distributed European Infrastructure for Supercomputing Applications (DEISA 2, [7]). There, its decentralized design will be fully exploited in an international grid environment of autonomous sites.

In Section II, we present the architecture of the met scheduler and describe the general interaction between a proxy and its peers. The P2P algorithms required for this

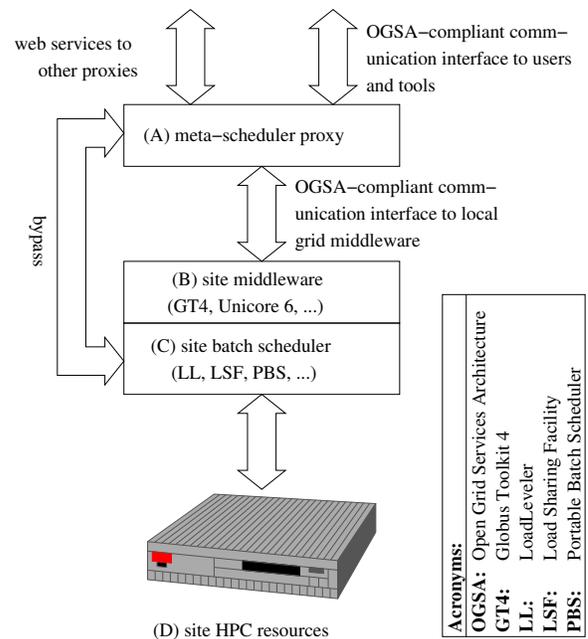


Figure 1: Architecture of the distributed met scheduler at the site level. Relevant references are OGSA [8], GT4 [9, 10], UNICORE 6 [11], LL [12], LSF [3, 4], PBS [13].

communication are portrayed in Section III and the idea of situation-based selection is introduced. Afterwards, the implementation of the met scheduler is explained in Section IV, where we will also present the details of the decision making algorithm. A full example of decision making is shown in Section V, where a decision must be made, which of a given set of resources is selected to execute a job. Then, an overview of related work on methods for resource discovery is given in Section VI. Finally, we finish this paper in Section VII with a conclusion and an outlook.

II. ARCHITECTURE

The architecture uses cooperating peers rather than a centralized master. Grid administrators install a local proxy software, whose block diagram is displayed in Figure 1. The figure shows the met scheduler proxy (A), the grid middleware (B), and the local batch scheduler (C). The proxy will have to interact with users, remote proxies, grid middleware, and local batch schedulers. Towards the user, the proxy provides web services that are compatible with the standards of the Open Grid Services Architecture (OGSA, [8]). Thereby, users and administrators can continue to use existing tools and client software to monitor jobs and hardware, oblivious to the fact that they communicate with a proxy.

Among themselves, the proxies will communicate using custom-designed but open web services. However, the communication will be mostly restricted to the search for available computing resources and the exchange of scheduling

information. For the actual migration of the job descriptions, the proxies rely on the grid middleware. The interface between proxy and middleware is compatible to the OGSA standard. Thus, the proxy can use any OGSA-supporting middleware. Popular examples include, e.g., Globus Toolkit (GT4, [9, 10]) and UNICORE 6 [11].

The actual metascheduling work of the proxies is performed in three stages by 1.) deciding that a job is to be migrated, 2.) discovering available resources, and 3.) deciding to which target site the job is to be migrated. In the first stage, the proxy determines which job to migrate away from the current site. Jobs may be selected for migration, e.g., because they require resources not available locally, access remote data, or reduce utilization of the resource by fragmenting the schedule. Which criteria are considered is usually determined by the site operator.

Afterwards, *resource discovery* is carried out in an *active* as well as *passive* fashion. In the first case, a proxy actively searches for available resources, while in the second case it only listens for messages sent by other proxies announcing underutilized resources. Having received a collection of offers from its peers, a proxy re-enters the *decision making* phase with the goal to select the best offer to accept. Now, the criteria considered include, e.g., the bandwidth available at a remote site, the cost incurred by using a resource, or other potential benefits offered to the user.

The remote side in this scenario influences the sequence of events twice: first, when it receives a request for computation time, it opts to either provide an offer itself or to ignore the request. Second, if it has disseminated an announcement about available resources, it chooses between all the candidates that took an interest in this offer. The criteria that control the decision in this case could be selected to, e.g., prefer jobs that best utilize the vacant resource, are immediately available for execution, or belong to a site that should be recompensed for previously accepting jobs itself.

The preceding paragraphs exhibited the fact that scheduling is based on local pieces of information and policies. While the examples focused mostly on criteria that resource providers find profitable, the decision making process can be easily extended to include measures that benefit users or the grid community. Currently, we restrict ourselves to the criteria *queue size*, *average waiting time*, and *waiting time*. They cover an acceptable range of interests insofar as their optimization is designed to result in improved utilization of resources, increased fairness, and greater customer satisfaction. Furthermore, these values can be simply determined and should be deducible from the information provided by any existing local batch scheduler. The real accomplishment will be to find a weighting of the criteria that considers the advantages of all parties involved in a grid environment.

In addition to the illustrated processes, there are other details 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. An offering site may 1.) reserve the offered timeslot until acknowledged or canceled by the receiving site, 2.) reserve the offered timeslot for a limited time, or 3.) don't provide any guarantees on the period of availability at all. Each of these scenarios requires a receiving site to react accordingly, by either canceling unused offers in a timely fashion, acknowledging offers within their restricted lifetime, or re-issuing a request if previous offers are withdrawn.

In an unstructured grid of independent peers, all these scenarios may occur and have to be supported by a meta-scheduler proxy. For this reason, our decision making algorithm ranks the offers in order of decreasing preference. It iterates through this list and tries to acknowledge the offers successively. The first offer that is available will be accepted and the remaining offers explicitly canceled. Thus, the algorithm will always return the best possible resource for the job regardless of the types of offers received.

III. RESOURCE DISCOVERY WITH P2P-ALGORITHMS

Over the last decade, peer-to-peer (P2P) networks have changed the way a search algorithm's usability is evaluated. With sizes of hundreds of thousands of simultaneous peers, they have rendered most traditional algorithms obsolete. Instead, a whole new class of *distributed hashtable (DHT) based* search algorithms has been created such as CAN [14], Chord [15], Pastry [16], and Tapestry [17]. The combination of a particular network overlay structure with an efficient routing algorithm makes them especially suited to deal with large distributed networks.

Unfortunately, their approach of hashing a search request is not particularly useful in a grid environment. Here, most search requests are multi-criteria queries, e.g., "10 nodes with 32 CPUs each for 5 hours", and composed of ranged criteria, e.g., "10-15 nodes". Hashing such a request results in loss of valuable information. Recent research tackles this problem by mapping these extended queries to the routing layers of, e.g., CAN or Chord. We describe three examples of such algorithms in Section VI. An excellent overview and a taxonomy for grid-enabled DHT-based algorithms is given in [18].

DHT-based algorithms derive their scalability, speed, and fault-tolerance from the fact that queries are distributed evenly amongst the participants of a network. This spread is a direct result of the mathematical properties of the used cryptographic hashing functions such as SHA-1. However, in a grid environment the necessary difference between query values is not always guaranteed. It is typical to encounter limited types of different hardware, common software stacks, and restrictions by resource providers on the requestable numbers of nodes and runtimes of jobs. All of these constraints reduce the possible values an attribute can adopt. As a result, the routing of queries focuses on a few grid nodes and a DHT-based algorithm's benefits are

seriously diminished. Considering the cost and complexity and their fairly insubstantial benefits for grids of the targeted size, we decided against using DHT-based algorithms.

A. Forwarding-based Algorithms

Instead, we employ traditional *forwarding-based* search algorithms. As their name suggests, these algorithms work by forwarding a request from one peer to another peer. Starting at an initial site, a request recursively spreads through the grid until some stop criterion is met. Each receiving peer tries to satisfy the request locally or, else, forwards it to its direct neighbors. Resource offers take the reverse path of a request back to the initial site.

In most instances, different algorithms vary in the way a request is routed, a stop criterion is chosen, or additional information is collected to augment the routing. Two exemplary algorithms that are representative of the class of forwarding-based algorithms are the Breadth-First-Search (BFS) and k -RandomWalks.

Breadth-First-Search: The BFS is the simplest of the forwarding-based algorithms without all the bells and whistles other variants attach to it. A request is endowed with an integral “time-to-live” (TTL) that is chosen by the initial site. Each peer that receives a request decrements the request’s TTL by one. If it hasn’t reached zero yet, the peer forwards a copy of the request to all of its neighbors indifferently. Afterwards, the peer tries to generate offers matching the request and returns them to the initial site.

Basic details will usually differ between implementations of this algorithm. In general, peers will check whether they have already seen a request before handling and forwarding it a second time. In this spirit, they will not forward a request to the neighbor they originally received it from either. Further, an implementation will usually see to it that each peer aggregates all offers from its neighbors. By bundling the replies at each peer, the number of return messages is kept down. But even with these enhancements, the number of requests grows exponentially with the depth of a search. Also, because the requests continue to be forwarded even if a satisfying offer was discovered, there is no chance to abort a search early.

Still, the BFS exhaustively examines the grid up to the search depth. If there is a matching resource on any of the peers it can reach in TTL hops, then the BFS is guaranteed to discover it. In an unstructured distributed grid, the BFS is the only algorithm with this kind of promise. Nevertheless, the algorithm is too expensive to be employed on its own.

k-RandomWalks: The k -RandomWalks algorithm is a variant of BFS that reduces the number of messages disseminated in the grid. The initial peer issues its request to a maximum of k neighbors. Each subsequent peer will forward the request only to a single random neighbor of its own. Therefore, the number of messages is bound by $k \times \text{TTL}$, and any of the two parameters can be adjusted independently.

The BFS’ major drawback — its huge cost in terms of messages — is somewhat reduced by k -RandomWalks. By modifying the parameters k and TTL the character of the algorithm can be changed. Increasing or decreasing TTL directly influences the distance that can lie between a requesting site and the potential offers it receives. Increasing or decreasing k controls the thoroughness with which the grid is searched for results.

However, regardless of the values of the parameters the algorithm will never yield the quality of results produced by the BFS. Instead, it will often find no results even though a matching resource is nearby. For this reason, the k -RandomWalks is rarely employed by itself, too.

B. Situation-dependent algorithm selection

The balance between proper results and incurred costs is crucial when employing forwarding-based algorithms. Naturally, there are more advanced algorithms in this class than the cited examples BFS and k -RandomWalks. Variants such as, e.g., Adaptive Probabilistic Search (APS, [19]) and Distributed Resource Location Protocol (DRLP, [20]) learn from previously performed searches. They collect data about which neighbor returned promising results and adjust their routing correspondingly. Nevertheless, the costs associated with these variants can not be justified by their results in general.

Instead of designing yet another forwarding-based algorithm, we chose to examine the existing methods more closely. It emerged that most algorithms are not to be rejected out of hand. Rather, it is the situation that determines the suitability of a method. No algorithm is always perfectly applicable. But given the right circumstances and an intelligently selected technique, the results can be acceptable. Therefore, it is advantageous to investigate the situations in which a search algorithm is to be invoked.

Coming back to our metascheduler, we discovered two main cases in which a P2P algorithm is necessary: I.) *resource requests*, that is, active inquiries into available resources matching specific requirements, and II.) *resource announcements*, that is, the dissemination of vacant resources that other peers passively listen for. We further identified two sub cases each: I.a.) *necessary* and I.b.) *optional* requests and, on the other hand, announcements of II.a.) *immediate* and II.b.) *future* resource vacancy. Of these four cases, each makes other demands on an appropriate search algorithm.

The key difference between the main cases lies in the way a particular message needs to be distributed. In I.) the system seeks a resource that matches detailed requirements. Thus, it is beneficial to employ algorithms that, e.g., learn from previously performed searches and can route a request specifically to sites where such a resource is known to exist. A similar design could theoretically be used with resource announcements in II.) to find jobs that can take advantage of a vacant resource. However, jobs are transient

compared to HPC resources. Although the probability to find a compatible job is higher on sites with a similar resource, other sites are not to be neglected.

The difference between a situation's sub-classes lies in the urgency, by which a metascheduler requires its results. In the cases I.a.) and II.a.) a higher priority is placed on the search than in the corresponding counterparts I.b.) and II.b.). Due to the higher priority, it is especially important to find actual results and preferably in a speedy manner. Extra cost incurred by an expensive search algorithm is acceptable due to the urgency of the request. For instance, it is more important to obtain jobs for an immediately available resource than for a resource available in the future. In the latter case, a sequence of cheap searches with a reduced chance of success may still yield a result in time.

Each class of situations requires another type of algorithm. It will always lead to an inferior performance if a single technique is chosen. Therefore, we opted to implement various forwarding-based algorithms and dynamically select the particular method to apply in a situation. Compared to DHT-based algorithms, the forwarding-based algorithms are mostly easy to implement and make no demands on a specific network overlay structure. With a basic framework in place, the simplicity of implementing these techniques allows us to support several methods simultaneously.

Each time a P2P search algorithm is about to be invoked, the circumstances are analyzed and the situation is categorized into one of the four classes I.a, I.b, II.a, and II.b. Then the appropriate algorithm is determined and the search initiated. Which algorithm is deemed appropriate can be configured by an administrator of a site. While it is necessary that every proxy supports all employed algorithms, each proxy can be differently configured. However, not every combination will actually lead to improvements. Due to the limited amount of space, we refer the interested reader to [21] for feasible configurations.

IV. SOFTWARE DESIGN

A huge part of the development time of any complex software system is spent dealing with secondary issues such as handling threads, writing web service stubs and skeletons, and managing database access. Therefore, we decided to build the metascheduler as an enterprise application archive (EAR) that is deployable in any application server conforming to the JavaEE 5 standard [22]. The environment provided by a JavaEE application server supports the programmer by taking over many of the more arduous tasks.

Moreover, an administrator benefits from an easy installation, configuration, and maintenance of the final application. Still, minor adjustments are generally required before an EAR can be deployed in a container since the JavaEE specification grants compliant implementations some leeway. Our development team uses the Apache Geronimo [23] application server because it offers a free, open, and complete

implementation of the JavaEE 5 standard. In addition, we plan to actively support Red Hat's JBoss [24] and SUN's Glassfish [25] in the fourth quarter of 2009, too.

The metascheduler is structured into three main modules: 1.) resource discovery, 2.) decision making, and 3.) resource management. Whereas the first two constitute the core modules of the scheduler, the latter embodies the interface to grid middleware or batch scheduler. Additional auxiliary components provide the glue binding the three main modules together. They are secondary to the logical modular design however, and we will neglect them for now.

The *resource discovery* module is chiefly responsible for the discovery of remote resources that match certain specified requirements. This task includes the active search for resources as well as the dissemination of vacant resources to other sites. The module incorporates the various forwarding-based P2P algorithms referred to in Section III-A and exposes them as web services to the remote peers. In cooperation with the decision making module, it implements the situation-based selection of one of these algorithms.

In the *decision making* module, we have concentrated the logic that steers the metascheduler. A variant of the Analytic Hierarchy Process (AHP) [26, 27] is used throughout the system to choose between alternative solutions. Its hierarchical design allows us to consolidate the opinions of several parties into an overall decision. The selection of the target site of a migration can therefore incorporate different viewpoints such as the interests of job owner, resource provider, and grid community. In contrast to the standard AHP, the utilities of an alternative are determined dynamically and with minimal human interaction.

Finally, the *resource management* module serves as an interface between the metascheduler and the lower layers of a site's scheduling stack. It provides a set of abstract methods that allow the reservation and management of timeslots of the underlying hardware resource. Different grid middleware will generally require different implementations of this module. But with an OGSA-compatible default implementation, the metascheduler is adequately equipped to handle a majority of the existing installations on a grid.

A. Resource Discovery

The implementation of a P2P algorithm's web services follows the "contract first" design approach. First, the abstract web services are unambiguously described in the Web Service Description Language (WSDL, [28]). Then, JAX-WS-compliant (Java API for XML Web Services, [29]) tools are used to generate an interface from this description. The interface contains the definitions of the web service methods. Implementing these methods and deploying the code into an application server results in a usable web service.

Each P2P search algorithm supported by a metascheduler proxy is made available as a separate web service under its own URL. Thus, a proxy is characterized by a bundle of

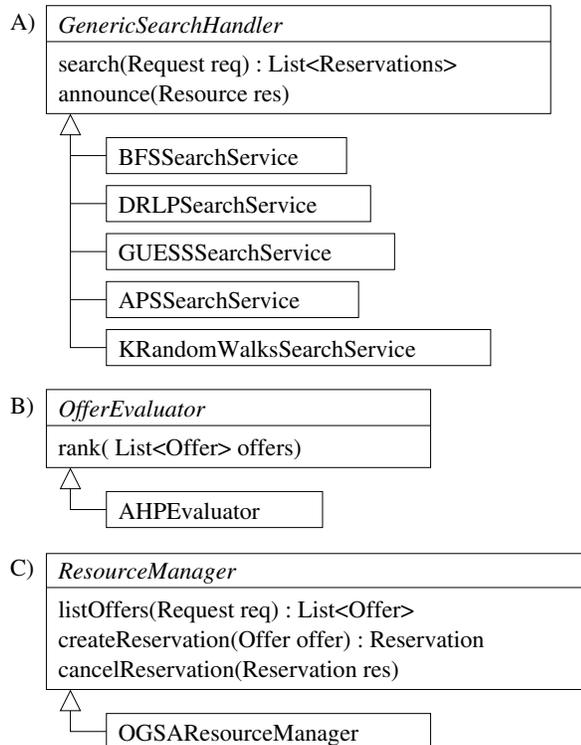


Figure 2: Software modules of the distributed metascheduler.

different service URLs. It is therefore reasonable to provide a specialized “get-to-know-service” that serves as a central point of entry for remote peers. It can be seen as some kind of directory that a remote peer can query to identify the additional services offered by a proxy. We call this service the *gatekeeper service*. Note, that this is not a central directory service but a local web service provided by a proxy to supply information about itself in agreement with the distributed nature of the design.

The information supplied by a gatekeeper currently contains a user-friendly name and a list of all locally available web services. The name is used exclusively as a convenience for users, while proxies identify themselves by their gatekeeper URLs instead. It can be specified by an administrator to label a managed resource, e.g., “IBM Power6 system at RZG”. In the list of services, each available web service is identified by its fully-qualified name (FQN) and mapped to its respective local URL. We established the provided information as scarce and restricted to the absolute minimum. It is easily extendable to include, e.g., a free text description of a resource, usage policies, service level agreements, detailed costs, etc.

Search algorithms are defined via a generic abstract web service displayed in part A) of Figure 2. An actual algorithm needs to implement only two methods: *search* and *announce*. The *search* method takes a resource request as an argument and returns a list of matching

reservations. Where these reservations actually come from, e.g., multiple reservations from a queried proxy itself or additional reservations from remote sites, depends on the particular algorithm considered. Naturally, reservations will have to contain the appropriate pieces of information that a querying site requires to positively identify the source of an offer.

As of now, a resource request incorporates a requirements definition in the format of the Job Submission Description Language (JSDL, [30]) plus a data block each for search algorithm and decision making algorithm. These blocks are used to map a received request to the appropriate algorithm implementation and, additionally, used by the particular algorithms to attach custom data to a request. Regarding the search algorithms, such a piece of custom data can, e.g., include timeout or original source of a request, routing directives, etc.

The *announce* method takes a structure similar to the request as an argument but has no return value. An argument’s JSDL component is used to describe the offered resource including the address of the offering site. A return value is not required, because a site that wishes to apply for the resource, is required to submit its request in a separate step. Thus, an announcement can be handled as a “fire and forget” message. By removing the reply message for this type of search, an announcement’s costs is kept down.

B. Decision Making

The module in charge of *decision making* is a simplified implementation of the Analytic Hierarchy Process (AHP) [26, 27, 31] displayed in part B) of Figure 2. It considers various facets of an item to sort a given set of similar items in order of preference. The module is applied in several stages of the metascheduling process to rank available timeslots, received offers, migration candidates, etc. By making the algorithm a replaceable module, the different sites could, in theory, employ different decision making modules without interfering with each other.

The AHP is a hierarchical multi-criteria decision making algorithm originally developed for the domain of economics. Its decision is based on a tree of criteria that have been chosen as relevant to the decision at hand. Figure 3 shows an exemplary AHP tree with two levels that combines several criteria relevant to a migration decision. Inner nodes of a tree define meta-criteria that are fully described by their child nodes. Leaf nodes represent criteria where the utility of an option can be explicitly determined. Each criterion in the tree has a weight assigned by a human. These weights determine the importance of a criterion with regard to its corresponding parent node.

The ranking of alternatives starts at the leaves of a tree. Here, each alternative item is evaluated and has its utility computed. In the original AHP, an item’s utility is determined based on pairwise comparisons provided by a human

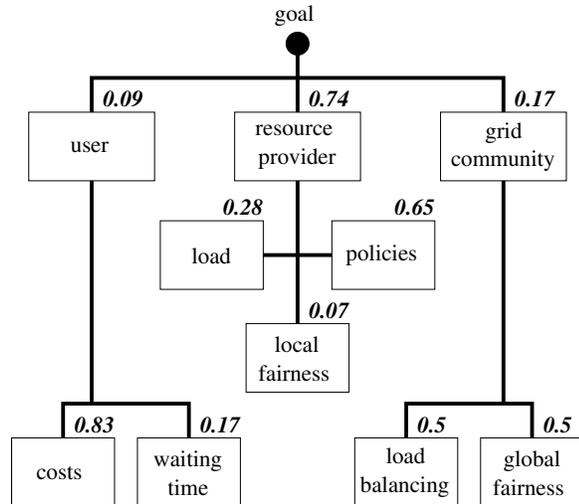


Figure 3: Exemplary AHP hierarchy with multiple parties. The weights of each node's children add up to 1.0.

decision maker. For this purpose, each pair of items has to be judged and the preference for an item has to be expressed on a scale of nine values. These values represent notions from “equally preferred” to “extremely preferred” defined by Saaty [26]. A pairwise comparison matrix $A = (a_{ij})_{1 \leq i, j \leq n}$ is the result of this rating process. Finally, the normalized principal Eigenvector of A is determined and its k th entry is used as the utility of the k th alternative.

Utilities with regard to inner nodes of a tree are computed recursively. At every node, the utility of an alternative is determined as the weighted sum of corresponding values at the child nodes. That is, for each inner node the utilities at its immediate child nodes are multiplied by the corresponding child's weight. These products are then added and result in an alternative's utility with respect to the parent node. As a by-product of using normalized utilities at the leaves and using normalized weights, the process also produces normalized utilities at each inner node. The obtained values are then used at the root node to rank alternatives from best to worst.

Usually, consent in an AHP context is reached by personal interaction: stakeholders come together to debate pairwise comparisons. In a grid, involved parties are geographically distributed, yet situations where a decision has to be reached are abundant. Because debate and direct interaction are infeasible, we exploit the hierarchical structure of a criteria tree to account for distinct opinions. Each stakeholder is allowed to build its own tree from a common set of criteria and assign its own weights. These separate trees are brought together under a common root to construct the final tree.

At this point, the persistent normalization protects an automated decision making process from malicious users. As utilities at each subtree's root are normalized, the expressed

opinions are initially of equal strength. In the next step, every subtree is connected to the root of the final hierarchy with its own weight. These *root weights* exclusively control the share of participation given to a particular stakeholder. In a grid environment, it is reasonable to assume that resource providers will reserve the right to define these values to themselves. As root weights can be defined on a per resource basis, different providers will not interfere with each other. Additionally, a provider can encourage participation — and even advertise the possibility thereof — without loss of control.

In our metascheduler, three stakeholders participate in a decision: 1.) the owner of a job, 2.) the provider of a resource, and 3.) the grid community. The lifetime of a subtree will vary according to the party that defined it. A user's tree is optionally created as part of the job submission process and permanently attached to a job description. It exists only as long as a job remains in the grid. A provider's tree is specific to a particular resource and part of a proxy's configuration. It will generally be constant for extended periods of time and will be used in every local decision making process. Finally, a grid community's tree is defined at the grid level and assumed to be part of the initial agreement to found a grid. It is configured at the proxy level, too, but identical across all participating sites. Typically, this tree will be the most static of the three subtrees. The root weights are configured in the same XML document that sets provider and grid community subtrees.

Furthermore, we extended the original AHP to dynamically compute utilities at the leaves of a tree. Such an extension is necessary, before AHP can be employed in a grid environment. Here, alternatives are defined by measurements or predictions such as waiting time (in minutes), cost (in dollars), and utilization (in percent). Again, AHP is clearly focused on human interpretation of the alternatives. Saaty gives several examples in [27], where a human decision maker's preference may differ significantly from the values suggested by a standard scale. We tried to accommodate this view and simultaneously make the algorithm feasible for a grid environment. Hence, we made the computation of utilities configurable based on an underlying criterion.

We provide two evaluation methods in the stock installation of a metascheduler: a direct mapping module and a hyperbolic tangent [32] based module. The direct mapping module allows a provider to map ranges of input values to specific utilities. It supports open ranges, ranges bounded above or below, and exact values. The resulting utility function is obviously non-continuous. It might be tempting to use this method for continuous criteria to express, e.g., that only waiting times of up to 5 hours are acceptable. However, this is generally a bad idea, because it leads to situations where the priorities do not reflect a stakeholder's intent. A job with a predicted waiting time of 4:59h is in most cases only marginally more preferred than a job with

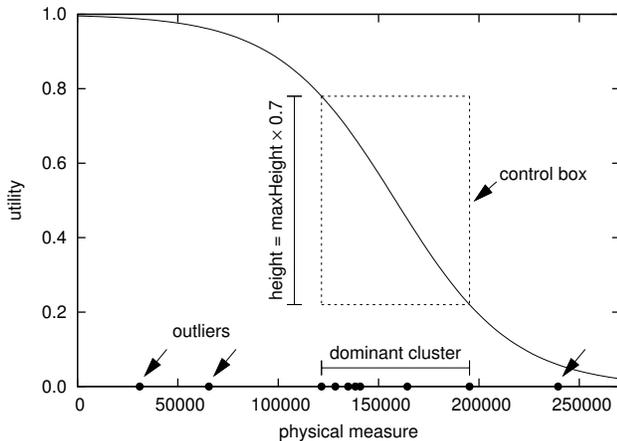


Figure 4: Continuous utility function for minimization criteria based on the hyperbolic tangent. The control box is determined automatically from the distribution of the measures.

5:01h waiting time. Thus, this method should only be used for discretely valued criteria such as user IDs, group IDs, queue names, etc.

In contrast, the evaluation method based on a hyperbolic tangent is designed for continuous criteria. It reflects the idea, that distinct utilities should be computed for the range of input values where the majority of the alternatives are located. Above average quality should yield higher utilities but eventually be bounded. An analogous reasoning is true for below average quality. We chose the hyperbolic tangent as the underlying utility function because it converges to 0 (resp. 1) for small (resp. large) input values. The notion of a *control box* is used to scale and shift the hyperbolic tangent with the aim of focusing its transition from 0 to 1 on average alternatives (see Figure 4).

Upon receiving a set of alternatives, the evaluation module first starts to determine the dominant cluster with regard to a given criterion. It iteratively aggregates the closest input values until a cluster exceeds 51% of all values. Horizontal position and width of a control box are set to reflect the position and width of this dominant cluster. Vertically, the box is centered around the midpoint 0.5 of the available utilities. The height of a control box is at most 0.8 and scaled with the ratio of elements in the dominant cluster. Determining a control box is independent of whether the current criterion is about maximizing or minimizing measures.

Finally, the hyperbolic tangent is scaled and shifted to match a control box, i.e., the curve runs through the lower left and upper right corners of a control box for maximizing criteria. Accordingly, it runs through the upper left and lower right corners for minimizing criteria. The resulting function is used to map input values to raw utility values, which are

then normalized to bring them in a form suitable for use in AHP.

Direct mapping and hyperbolic tangent based evaluation represent two methods designed for non-continuous and continuous criteria, respectively. In Section V, we present a full example that uses both mapping techniques. However, the metascheduler is easily extendable with additional evaluation methods.

C. Resource Manager

The interface between metascheduler and locally installed grid middleware or batch scheduler is represented by the resource manager module displayed in part C) of Figure 2. It provides the means to inquire for a set of offers matching a request's JSDL description and, optionally, to fix such an offer into a reservation. The definitions of offer and reservation are deliberately kept as broad as possible to account for different systems, which a metascheduler is required to interact with.

Conceptually, an offer represents a timeslot on some resource that is endowed with additional metadata. The contents of the structure must allow the resource manager to identify the offered resource and assign a job correctly to it. The attached pieces of metadata constitute a mapping of arbitrary criteria to corresponding measures taken or predicted by a resource manager. An offer satisfies the pre-conditions put forth by the decision making module. Thus, a set of offers can be ranked in order of preference if some weighting of the criteria is supplied externally.

As mentioned previously, the metascheduler currently supports the criteria queue size, average waiting time, and waiting time. These points of comparison were selected because every existing batch scheduler should be able to supply them easily. However, the metascheduler is not restricted to any fixed set but can be configured to understand arbitrary criteria. Thus, the set can be freely extended as long as a resource manager knows how to obtain the related values.

A reservation is just a wrapper around an offer that optionally assigns an owner to it. Whether this assignment is for a limited period of time, until the reservation is explicitly canceled, or not authoritative at all is up to the particular resource manager or its site. The main reason for a separation of offer and reservation was to distinguish between the mere availability of a timeslot and its dedication to a particular party.

V. DECISION MAKING — AN EXAMPLE

Two unique features distinguish the described metascheduler from existing solutions. First, its concurrent use of multiple forwarding-based search algorithms that selects the best algorithm for a given situation. Second, its generic decision making algorithm that supports multiple stakeholders for each scheduling decision.

The same basic principles of decision making will eventually be used in both search module and generic DM module. Forwarding-based search algorithms are then plain implementations of existing techniques that generally require no customization. For this reason, the following example will focus on the new concepts in decision making. We will show a selection process that is common to scheduling: which resource from a given set of alternatives will be used to execute a compute job. In the process, we will pay particular regard to the observance of individual interests in a joint decision.

A user has used the metascheduler to submit a job that requires 9,216 CPU-hours to run to completion. It is assumed, that the job is moldable and will scale perfectly to any number of CPUs, i.e., it can run in 3 days on 128 CPUs or in 18 hours on 512 CPUs. Such an assumption is rather naive and more realistic models for the speedup of parallel programs exist. For instance, Cirne and Berman [33] presented a generator that models sets of moldable jobs according to Downey [34]. However, an explanation of these concepts is beyond the scope of this example.

As part of the submission process, the user selected three criteria that he wishes the metascheduler to optimize. His goals are to minimize the *cost* of computing his job, to minimize the *waiting time* until his computation will start, and to minimize the overall *completion time* until the results are returned to him. Then, he used the method established by AHP to judge the three criteria with regard to their relative importance on a verbal scale. The user stated that he preferred lowering the cost strongly over lowering the waiting time and, moderately over lowering the completion time. Likewise, he preferred lowering the completion time moderately over lowering the waiting time. From these judgments, normalized weights were derived for cost, waiting time, and completion time as, respectively,

$$w_{\text{cost}} = 0.637, \quad w_{\text{wait}} = 0.105, \quad w_{\text{comp}} = 0.258.$$

Note that weights derived through the AHP will always add up to 1.0.

The metascheduler issued a search on behalf of the user's job to acquire offers for a matching slot on an HPC resource. Eventually, the search returned five offers from three different resources. For the sake of simplicity, we are working with several assumptions about these resources and offers. First, the discovered resources were filtered such that they fulfill all necessary requirements with regard to software, disk space, memory, etc. Second, the user is authorized to use any of the three HPC resources. And third, all resources belong to the same resource provider. This third assumption is most restrictive, but allows us to keep the number of parties participating in the decision down to the user and his provider.

Like the user, the resource provider selected two criteria that the metascheduler is supposed to optimize. The first

goal is to maximize the average load of the resources, and, therefore, to favor resources with a low *utilization* during scheduling. Then, explicit *queue priorities* were manually defined for queues and job classes. On resource *A*, medium-sized jobs that used at most 64 CPUs are favored over large jobs that used up to 512 CPUs in parallel. This decision was made to increase the number of users served simultaneously and, therefore, maximize the perceived fairness. Also, guests on resource *C* are penalized by a lower priority with regard to regular users for political reasons. The second goal is to stay true to these priorities and consider them in the scheduling process. Finally, both criteria were judged by an administrator who stated that better utilization was moderately preferred over adherence to the queue policies. The resulting normalized weights are

$$w_{\text{util}} = 0.750, \quad w_{\text{prio}} = 0.250$$

for *utilization* and *queue priorities*, respectively.

Furthermore, the root weights that govern the share of participation were established in cooperation with a user representative. Instead of using the AHP, the values were obtained by external means and set to

$$w_{\text{user}} = 0.16, \quad w_{\text{prov}} = 0.84,$$

for *users* and *provider*. In other words, a share of 16% in the overall decision is granted to users.

Table I shows the relevant characteristics of the three resources that the metascheduler found. For each resource the cost in dollars per CPU-hour was chosen consistent with the values calculated by Walker [35]. All remaining properties were defined based on our experience with HPC resources. Table II lists five offers and their characteristics with regard to the decision making criteria. An offer's cost is determined by multiplying the job size of 9,216 CPU-hours with the corresponding resource's cost. Waiting times were picked randomly to provide some variance in the quality of each offer. Finally, completion time is derived from the waiting time and the job's runtime using the maximum number of CPUs allowed for each queue. Utilization and queue priority associated with an offer can be obtained from Tab. I.

Now, the next step is to evaluate all offers with regard to all criteria specified by the participating decision makers. For cost, waiting time, completion time, and utilization, the style of automatic prioritization illustrated in Section IV-B is used. Queue priorities, however, already represent priorities themselves. They are an example for a direct mapping technique where discrete values — in this case the queue names — are mapped to static utility values. On resource *C*, however, the mapping can be implemented purely numerical by detecting user or guest status based on the user ID. Still, the resulting utilities are not normalized yet. Therefore, they are divided by the sum of all five queue priorities to fit into the generic hierarchical prioritization scheme.

| Resource | Cost (\$/CPU/hour) | Utilization | Queues | Queue Priority | Policy |
|----------|-----------------------|-------------|--------|----------------|---------------|
| A | 0.07 | 0.95 | jumbo | 1.0 | max. 512 CPUs |
| | | | medium | 1.2 | max. 64 CPUs |
| B | 0.045 | 0.90 | | 1.0 | max. 128 CPUs |
| C | 0.10 | 0.70 | users | 1.0 | max. 64 CPUs |
| | | | guests | 0.6 | max. 64 CPUs |

Table I: Characteristics of the available HPC resources.

| Offer-ID | Resource | Queue | Cost (\$) | Waiting time (hours) | Completion time (hours) |
|----------|----------|--------|--------------|-------------------------|----------------------------|
| offer-1 | A | jumbo | 645.12 | 36 | 54 |
| offer-2 | A | medium | 645.12 | 16 | 160 |
| offer-3 | B | | 414.72 | 20 | 92 |
| offer-4 | C | users | 921.60 | 4 | 148 |
| offer-5 | C | guests | 921.60 | 8 | 152 |

Table II: Characteristics of the resource offers.

The four utility mappings with automatically derived hyperbolic tangents are shown in Figure 5. It can be seen, that the algorithm adjusts itself to distribution and magnitude of the underlying measures regardless of the actual criterion. For instance, most of the waiting times lay in the range between four and 20 hours with a negative outlier at 36 hours. Here, the algorithm maps the average offers to (unnormalized) utilities between 0.18 and 0.82, while it singles out “offer-1” and assigns an utility that marks it as unacceptable. On the other hand, three out of five completion times fall between 148 and 160 hours with two positive outliers at 54 and 92 hours. Again, the mapping assigns values in the middle spectrum of utility to the cluster, but it awards “offer-1” and “offer-3” with utilities close to the maximum. Also note, how the algorithm accommodates the height of the control box to the share of options in the dominant cluster. Figure 6 shows the normalized utilities for all offers and criteria in comparison.

In a final aggregation step, the singular utilities are combined for each set of elements in the criterion hierarchy until the root is reached. To illustrate, let’s consider “offer-1”, which is characterized by five independent utilities:

$$u_{\text{cost}}^{(1)} = 0.198, \quad u_{\text{wait}}^{(1)} = 0.005, \quad u_{\text{comp}}^{(1)} = 0.279, \\ u_{\text{util}}^{(1)} = 0.080, \quad u_{\text{prio}}^{(1)} = 0.208.$$

A weighted sum of these values is calculated to determine the utilities from the point of view of the user and the resource provider, respectively. The user’s utility for “offer-1” is, therefore,

$$u_{\text{user}}^{(1)} = w_{\text{cost}} u_{\text{cost}}^{(1)} + w_{\text{wait}} u_{\text{wait}}^{(1)} + w_{\text{comp}} u_{\text{comp}}^{(1)} = 0.198$$

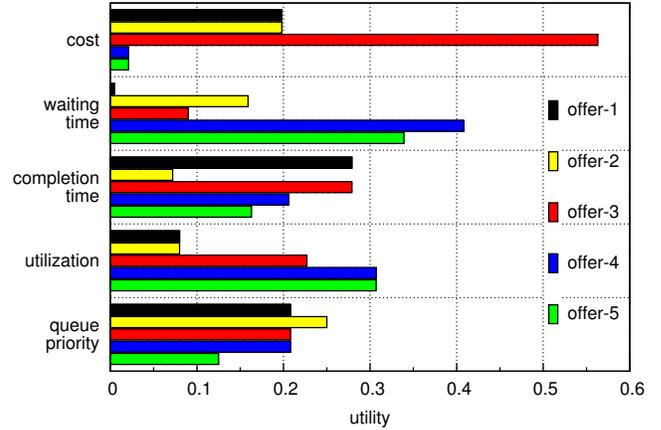


Figure 6: Utilities of each resource offer with regard to the leaf criteria.

and the provider’s utility

$$u_{\text{prov}}^{(1)} = w_{\text{util}} u_{\text{util}}^{(1)} + w_{\text{prio}} u_{\text{prio}}^{(1)} = 0.112.$$

Finally, these utilities are combined with the root weights to obtain the final overall utility of the first offer.

Intermediate utilities, i.e., utilities with regard to user, provider and the overall goal, are shown in Figure 7. It is noticeable, that the overall utilities are closely related to the provider utilities. The reason for this similarity is, of course, the large share of the root weights granted to the resource provider. Nevertheless, the user’s strong preference for the third offer is not ignored. Despite his minor power, the third option is judged as marginally more preferable than the fourth option in the final prioritization. Using the

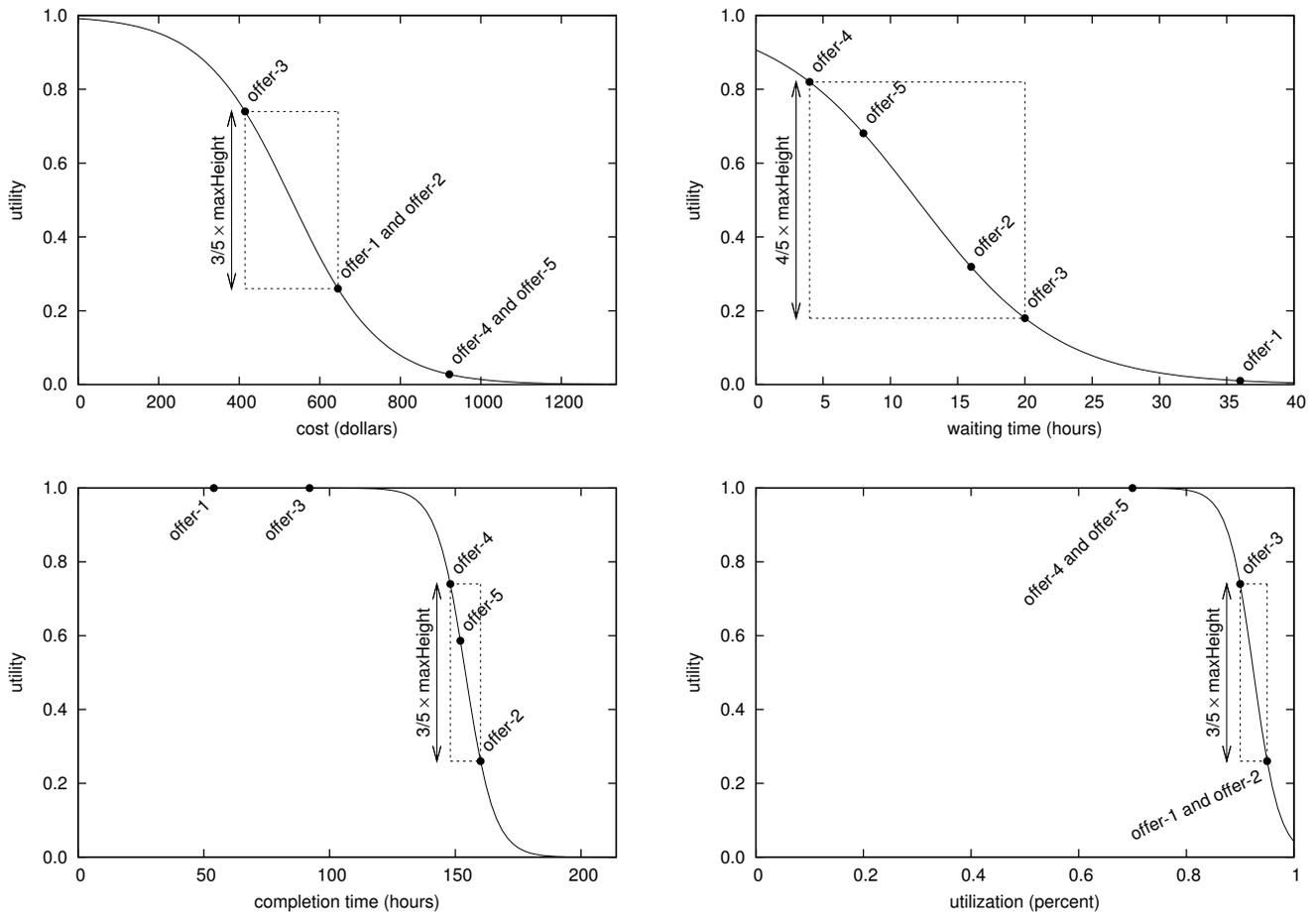


Figure 5: Automatic utility mappings derived for the criteria cost, waiting time, completion time, and utilization.

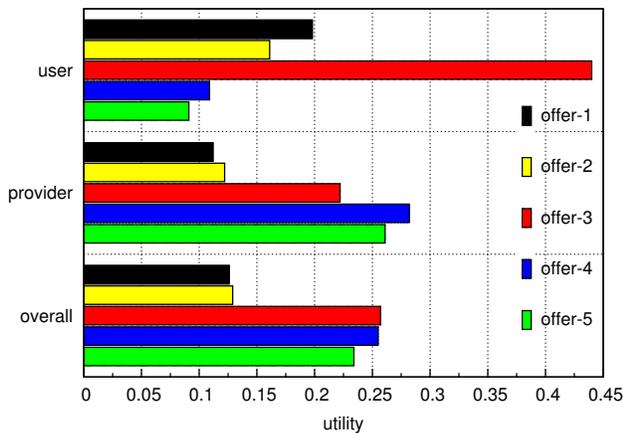


Figure 7: Per-party and overall utilities of each resource offer.

overall utilities as a guideline, the metascheduler would try

to acknowledge the offers in decreasing order of preference

$$\text{offer-3} > \text{offer-4} > \text{offer-5} > \text{offer-2} > \text{offer-1}.$$

The first offer that would actually be accepted by both sides of the bargain is assigned to the job; remaining offers are subsequently canceled.

VI. RELATED WORK

Current grid scheduling solutions focus on centralized resource discovery approaches. UNICORE 6 [11] is a grid middleware that is widely deployed, e.g., in the DEISA 2 grid [7] and in parts of the German D-Grid [36]. It performs resource discovery via a single registry called *Common Information Provider* (CIS). The CIS collects static and dynamic information on a grid's resources, which is provided by *CIS Information Providers* (CIP) monitoring each resource. A CIP publishes an Atom-feed that is periodically polled by the CIS by means of a web service protocol. The migration of job steps in a workflow is performed by so-called *Service Orchestrators*. They employ different

brokering strategies, which in turn use the information from the CIS to reach their decisions.

A similar approach is taken by the second major grid middleware Globus Toolkit 4 [9, 10]. The resource management is contained in the *Monitoring and Discovery Service* (MDS), which consists of *Aggregator Framework*, *Index*, and *Trigger*. Index is the counter-part to UNICORE's CIS. There are one or more index servers per virtual organization (VO) that store all the information about a VO's resources. The Aggregator Framework is used locally to monitor a resource by periodically spawning shell scripts. Obtained data is then pushed into the central index.

Finally, the Trigger component can be configured to perform actions whenever a predefined event occurs. While the Aggregator Framework is the primary way of updating an index, Trigger can also be used to poll information into the registry. Globus Toolkit 4 has no support for automatic migration of jobs by itself. It requires the help of additional software such as Gridway metascheduler [2] to provide this functionality. The Gridway information manager accesses the MDS index via its web service interface to discover remote resources. Based on this data it reaches its scheduling decisions. Another centralized scheduling approach that resides on top of GT4 is described in [37].

Distributed designs for resource discovery have previously been produced in theory. Here, the focus has been on the transfer of grid principles to the domain of P2P networks. Examples for such algorithms are MAAN [38], Squid [39], and QuadTree [40]. We sketch these three methods because they represent interesting and unique approaches to the problem of mapping ranged criteria and multi-criteria queries to the underlying DHT-based substrates. Many more grid-enabled DHT-based algorithms are described in [18].

Multi-Attribute Addressable Network (MAAN) uses two separate procedures to augment the underlying Chord algorithm. First, multi-criteria requests are resolved by maintaining distinct hashing functions and querying separately for each involved attribute. Second, MAAN employs a *locality preserving hashing function*, that is, a function that maintains the order relation between numerical values for their hashed values. Each query for a ranged criterion is then represented by the actual sought-after value and the minimum and maximum allowed values. MAAN forwards a request from the node responsible for the minimum value via its successors in the Chord ring to the node corresponding to the maximum value. Whatever results have been found at this point are returned to the querying node.

The Squid [39] algorithm is based on a Chord routing layer, too. It interprets the d criteria or attributes relevant to a grid-style query as spanning a d -dimensional space. This space is mapped to a 1-dimensional space by means of *space filling curves* (SFC). The resulting scalars are conventionally hashed and mapped to grid nodes using the Chord routing layer. Each multi-criteria query corresponds to a point or,

in case of ranged criteria, a sub-region of the d -dimensional space. It is mapped by the SFC to distinct clusters of scalars. Each cluster corresponds to individual nodes in the grid, which have to be queried separately. The results obtained by several of these traditional DHT-queries are then merged to receive the final results of a grid-enabled DHT-query.

Another method that gets by with a single Chord DHT-layer is the QuadTree algorithm [40]. It recursively subdivides the d -dimensional attribute space with the help of quad-trees. Each block is identified by its centroid and mapped to a Chord node. A ranged multi-criteria query intersects with one or more blocks. To execute a query, those relevant blocks are determined and the corresponding grid nodes are contacted. The combined results of these distinct queries form the final response. The performance of the QuadTree algorithm is further improved by a cache. Here, each node stores the addresses of its immediate children in the tree to reduce the number of lookups performed.

With regard to decision making, the scheduling heuristic of Maui Cluster Scheduler [41] bears certain similarity to the hierarchic approach we propose in this article. However, Maui's hierarchy is static and weights are the only mechanism by which an administrator can modify the criteria. A direct modification of the tree is infeasible, since aggregation of the individual per-criteria priorities is not generic. Instead, criteria derive priorities from their child criteria in unique ways, e.g., applying minima or maxima along the way. Furthermore, the definition of weights is restricted to the administration of a resource, and, in contrast to our design, users can not participate in a scheduling decision. Maui, though, supports fairshare criteria that favor a fair distribution of the resource amongst the users. Thus, user interests are considered in the scheduling at least indirectly.

VII. CONCLUSION AND FUTURE WORK

We have presented our implementation of a hybrid algorithm for a distributed metascheduler that efficiently links available resources and matching jobs. It supports the exchange of jobs between resources and, thereby, achieves improved resource utilization and shorter turn over times. The metascheduler is currently being implemented and will be deployed as part of the DEISA2 grid in fall of 2009.

The design of the metascheduler fully supports the JavaEE specification's security framework. Accordingly, the metascheduler can be extended to integrate existing security infrastructure including virtual organizations, Community Authorization Service [42], and Shibboleth [43]. Towards middleware or local batch schedulers it acts transparently, hence, letting these layers provide their own security arrangements.

Finally, we provided an example for a decision making process that is common to scheduling. The example showed how the prioritization algorithm uses clustering to reward or penalize options of particular merit or demerit, respectively.

Furthermore, it illustrated how the opinion of a job's owner can be regarded in a decision without loss of control for the resource provider.

During the design and implementation of the scheduler, we gained intense experience with input queues and local resource management systems (LRMS) of HPC sites. In cooperation with the authors of [37], we have isolated four intrinsic problems that have to be solved to make metascheduling a fully working feature in the future.

First, the grid scheduler generally has only the role of a "power user" from the perspective of the LRMS and has to compete with other users, e.g., other grid schedulers in the same grid. As a consequence, an optimal schedule is not possible for any individual metascheduler.

Second, grid schedulers have no control over a site's policy, and, therefore, over the prioritization of the jobs waiting in queues. Thus, no control exists over the respective local resource management system for the metascheduler.

Third, every site uses a custom configuration of queues, and processors can be shared among queues or dedicated exclusively. Usually, the information from the LRMS does neither allow one to reliably determine the number of free processors nor does it allow one to determine the total number of processors. The only statistics commonly available are the number of running and waiting jobs. Some sites do not even provide this information because of nondisclosure. To conclude, local schedulers currently do not provide sufficient information for a good schedule.

Finally, resources are highly utilized and waiting times at clusters can last up to hours. Therefore, queue waiting time considerably exceeds the actual execution time for small jobs. Since small jobs constitute the majority of all jobs, input queue waiting time is the dominant factor. However, input queue waiting time and input queue length have shown to be not continuously differentiable functions over time. Instead, they can vary within minutes by a factor of thousand or more. They behave more like fractals than continuous functions. This makes predictions of future queue waiting times and queue lengths a delicate task. However, scheduling always relies on such predictions. Future work will focus on finding solutions to these obstacles.

REFERENCES

- [1] J. Heilgeist, T. Soddemann, and H. Richter, "Design and implementation of a distributed metascheduler," in *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP '09. Third International Conference on*. IEEE Computer Society, Oct. 2009, pp. 63–72.
- [2] "Gridway," <http://www.gridway.org>, The Globus Alliance, [accessed: 2010-06-19].
- [3] "Platform Computing," <http://www.platform.com>, [accessed: 2010-06-19].
- [4] S. Zhou, "LSF: Load sharing in large-scale heterogeneous distributed systems," in *Proc. Workshop on Cluster Computing*, 1992.
- [5] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour, "Evaluation of job-scheduling strategies for grid computing," in *GRID '00: Proc. 1st IEEE/ACM Intl. Workshop on Grid Computing*. Springer-Verlag, 2000, pp. 191–202.
- [6] Q. Wang, X. Gui, S. Zheng, and Y. Liu, "De-centralized job scheduling on computational grids using distributed backfilling: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 18, no. 14, pp. 1829–1838, 2006.
- [7] "Distributed European Infrastructure for Supercomputing Applications (DEISA 2)," <http://www.deisa.eu>, [accessed: 2010-06-19].
- [8] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw *et al.*, "The Open Grid Services Architecture (OGSA)," <http://www.ogf.org/documents/GFD.80.pdf>, Open Grid Forum, July 2006.
- [9] I. Foster, "Globus toolkit version 4: Software for service-oriented systems." in *IFIP Intl. Conf. on Network and Parallel Computing*, ser. Lecture Notes in Computer Science, H. Jin, D. Reed, and W. Jiang, Eds., vol. 3779. Springer-Verlag, 2005, pp. 2–13.
- [10] "Globus Toolkit," <http://www.globus.org>, The Globus Alliance, [accessed: 2010-06-19].
- [11] "UNICORE," <http://www.unicore.eu>, Jülich Supercomputing Centre, [accessed: 2010-06-19].
- [12] *IBM Load Leveler: User's Guide*, IBM Corp., Sept. 1993.
- [13] R. Henderson and D. Tweten, "Portable Batch System: External reference specification," NASA Ames Research Center, Tech. Rep., 1996.
- [14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proc. 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2001, pp. 161–172.
- [15] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM '01: Proc. 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM Press, 2001, pp. 149–160.
- [16] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proc. IFIP/ACM Intl. Conf. on Distributed Systems Platforms*. Springer, 2001, pp. 329–350.
- [17] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE J. Sel. Area Comm.*, vol. 22, no. 1, pp. 41–53, 2004.

- [18] R. Ranja, A. Harwood, and R. Buyya, "Peer-to-peer based resource discovery in global grids: A tutorial," *IEEE Commun. Surveys Tuts*, vol. 10, no. 2, pp. 6–33, 2008.
- [19] D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search for peer-to-peer networks," in *P2P '03: Proc. of the 3rd Intl. Conf. on Peer-to-Peer Computing*. IEEE Computer Society, 2003, p. 102.
- [20] D. Menascé and L. Kanchanapalli, "Probabilistic scalable P2P resource location services," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 2, pp. 48–58, 2002.
- [21] 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*. IEEE Computer Society, Nov. 2007, pp. 60–66.
- [22] "Jsr-000244: Java EE 5.0 specification," <http://jcp.org/en/jsr/detail?id=244>, Sun Microsystems, Inc., [accessed: 2010-06-19].
- [23] "Apache Geronimo," <http://geronimo.apache.org>, The Apache Software Foundation, [accessed: 2010-06-19].
- [24] "JBoss Enterprise Middleware," <http://www.jboss.org>, Red Hat Middleware, LLC., [accessed: 2010-06-19].
- [25] "Sun GlassFish," <https://glassfish.dev.java.net>, Sun Microsystems, Inc., [accessed: 2010-06-19].
- [26] T. Saaty, *Multicriteria Decison Making: The Analytic Hierarchy Process*, 1988, revised and published by the author; Original version published by McGraw-Hill, New York, 1980.
- [27] —, "How to make a decision: The Analytic Hierarchy Process," *Eur. J. Oper. Res.*, vol. 48, no. 1, pp. 9–26, 1990.
- [28] "Web Service Description Language (WSDL)," <http://www.w3.org/TR/wsdl>, World Wide Web Consortium (W3C), [accessed: 2010-06-19].
- [29] "The Java API for XML-based Web Services (JAX-WS)," <http://jcp.org/en/jsr/detail?id=224>, Sun Microsystems, Inc., [accessed: 2010-06-19].
- [30] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job Submission Description Language (JSDL)," <http://www.ogf.org/documents/GFD.136.pdf>, Open Grid Forum, July 2008.
- [31] T. Saaty, *Fundamentals of the Analytic Hierarchy Process*. RWS Publications, 2000.
- [32] M. Abramowitz and I. A. Stegun, Eds., *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing*. Dover, 1972, ch. 4.5 Hyperbolic Functions, pp. 83–86.
- [33] W. Cirne and F. Berman, "A model for moldable supercomputer jobs," in *IPDPS '01: Proceedings of the 15th International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2001, p. 10059b.
- [34] A. B. Downey, "A model for speedup of parallel programs," Computer Science Division, University of California, Berkeley, Technical Report UCB/CSD-97-933, Jan. 1997.
- [35] E. Walker, "The real cost of a CPU hour," *Computer*, vol. 42, no. 4, pp. 35–41, Apr. 2009.
- [36] "D-Grid Initiative," <http://www.d-grid.de>, [accessed: 2010-06-19].
- [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. Schwarzmann, 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] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," *4th Intl. Workshop on Grid Computing*, p. 184, 2003.
- [39] C. Schmidt and M. Parashar, "Flexible information discovery in decentralized distributed systems," in *HPDC '03: Proc. of the 12th IEEE Intl. Symp. on High Performance Distributed Computing*. IEEE Computer Society, 2003, p. 226.
- [40] E. Tanin, A. Harwood, and H. Samet, "Using a distributed quadtree index in peer-to-peer networks," *The VLDB Journal*, vol. 16, no. 2, pp. 165–178, 2007.
- [41] D. B. Jackson, Q. Snell, and M. J. Clement, *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Berlin/Heidelberg, Germany: Springer, 2001, vol. 2221/2001, ch. Core algorithms of the Maui scheduler, pp. 87–102.
- [42] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A community authorization service for group collaboration," in *Proc. of the 3rd Intl. Workshop on Policies for Distributed Systems and Networks*, 2002, pp. 50–59.
- [43] "Shibboleth," <http://shibboleth.internet2.edu>, Internet2 Middleware Initiative, [accessed: 2010-06-19].