Algorithms for Job and Resource Discovery for the Meta-Scheduler of the DEISA Grid

Janko Heilgeist, Thomas Soddemann, Harald Richter

***Abstract*—This paper describes a distributed meta-scheduling architecture for the DEISA grid in which site-local proxies migrate jobs between the participating sites with the aim to improve user- and administration-defined goals like job runtimes, hardware utilization, load balancing, etc. The meta-scheduler employs P2P-based algorithms to discover available resources for its local jobs and jobs for its available resources respectively and thereby achieves continued operation even in the case of network or site failures.**

***Index Terms*—Grid computing, meta scheduling, P2P algorithms**

I. Introduction

With the advent of grid computing, a special form of distributed computing, computing resources of many distributed centers are available to their customers which are themselves in some projects distributed all over the world. In order to be able to efficiently offer resources to customers, the necessity grows to balance resource requests across grid infrastructures automatically. Currently, with a few exceptions grid infrastructures only offer limited support for brokering resources. Grid middlewares like UNICORE [1] and the Globus Tool Kit [2] require the user to make the decision on specifying the site which may be offering resources.

DEISA is the Distributed European Infrastructure for Supercomputing Applications. It is formed by eleven of the top European computing centers. Hence, it is much more than just another Grid project. DEISA offers its customers the unique opportunity to access supercomputing resources hardly available elsewhere. While most of the computing resources are used at single sites, there are projects which can make use of resources allocated at multiple sites. Due to nonexistent middleware, currently the resource allocation for such requests is performed manually.

Those deficiencies in automated cross-site resource allocation lead to a few attempts of improving the situation. E.g., the Gridway project [3] tries to address the problem especially in a Globus Tool Kit context. This is at the same time its current weakness. Since not all infrastructures use the Globus Tool Kit, interoperability is not given and due to other architectural concepts e.g. in UNICORE difficult to achieve.

Platform [4] offers a proprietary solution with LSF Multicluster. Here a Grid infrastructure is seen as a Cluster of clusters. Although support is given for a variety of batch systems, their solution is neither open nor adheres to OGF [5] and OASIS [6] standards, which again makes interoperation difficult. With our attempt we try to circumvent such limitations.

This paper describes a distributed meta-scheduling architecture which allows the migration of jobs between the participating resource providers of a grid-like infrastructure with the aim of improved resource utilization, load balancing, and turn over times. An approach will be described which makes use of P2P-based algorithms. The next section describes the basic ideas of our solution in the context of Grid middleware architectures. The third subsection will cover the topic resource location. A conclusion and an outlook will close this paper.

II. Basic Architecture

A meta-scheduler, sometimes also called super-scheduler, has to be distinguished from a site's batch-scheduler. The owner of a resource usually provides one or more queues to which jobs can be submitted by authorized users. The batch-scheduler manages the assignment of these jobs to locally available hardware resources, executes the application belonging to a job and monitors the execution. On the other hand, a meta-scheduler is deployed on a layer above the batch-schedulers and migrates jobs between remote site's local job queues. This migration can be the result of a user's explicit request for a remote resource or a grid-wide load-balancing attempt by the meta-scheduler.

Meta-schedulers can be categorized by their topology as either *centralized* or *distributed* [7]. A *centralized* scheduler is a dedicated entity at one preferred site of the grid which typically has information about all other sites and resources of the grid. A central meta-scheduler assigns jobs either directly to the best resource available or indirectly via the batch queue of that site that hosts the best resource. Indirect job submission relies on the local site scheduler of the respective batch input queue. In this case, the local scheduler must assign the job on behalf of the central scheduler to one of its computers.

In a *distributed* (or *decentralized*) architecture, each site has a local representative of the meta-scheduler which we call proxy. A proxy is responsible for deploying a job in the grid, for communication with other proxies as well as with the middleware of his site. A proxy does not have the overview over all grid resources since it collects scheduling data only locally and from its direct neighbors for performance reasons. Therefore, the distributed approach has to find ways to provide the local proxy with sufficient input for scheduling. This can be achieved by inter-proxy communication, by a grid-wide pool of jobs waiting for execution, or by overlapping smaller pools of jobs of adjacent sites [7], [8].

Although the central approach is easier to design, implement, deploy and maintain, it has several disadvantages.

0-7695-2992-5/07 $25.00 © 2007 IEEE
DOI 10.1109/ADVCOMP.2007.10

Firstly, a central entity is also a single point-of-failure. A broken Internet link, for example, that splits the grid into sub-grids can disconnect sites from the central scheduling. Secondly, a successful attack at the central instance shuts down and possibly compromises the whole grid.

Thirdly, a single scheduler is a performance bottleneck since it has to be informed about every change on every site for its decision to be optimal. This is not a problem for small grids, but scaling the performance becomes harder as the grid grows in sites, users and jobs. Although these issues may be solved by multiple backup servers running the meta-scheduling software, another problem remains: Grids typically span across multiple sites in one country or between countries and local scheduling policies must be taken into account that a central meta-scheduler can not be aware of. For instance, some sites may be reluctant to give away control over their own computing resources, other sites may want to influence the schedule by favoring their user groups or projects. Finally, there may not exist an optimal schedule that accommodates all parties equally. Therefore, the costs of finding such a non-existing schedule must be limited.

The aforementioned problems of a central scheduler are largely non-existent in a distributed meta-scheduler. There is no central point-of-failure, no bottleneck, and splitting the grid in sub-grids will leave sites unaffected, as long as at least adjacent sites can contact each other. The price for the higher reliability is that scheduling will be based on less information and may be thus sub-optimal. Although, full information could be achieved by complete data exchange between all proxies, the resulting $O(n^2)$ costs are prohibitive and make it infeasible. To summarize, we opt for decentralized meta-scheduling.

The distributed architecture we propose improves existing meta-scheduling by employing communication algorithms known from peer-to-peer networks, and by exchanging jobs with other sites. In fig. 1 the basic architecture of the proposed meta-scheduler is depicted for one site. Other sites are analogous.

In our architecture, each site sets up a proxy (A) of the meta-scheduler. This proxy uses the Open Grid Services Architecture (OGSA, [9]) to communicate via the site's middleware (B) with the local batch scheduling system (C). By restricting the proxies' interface to standardized web services, the system can cooperate with any OGSA-compliant middleware such as Globus Toolkit 4 (GT4, [2]) or UNICORE 6 [1]. Unfortunately, OGSA transfers only little information about queue policies of the underlying local batch scheduler to the proxy. Furthermore, not even all local batch schedulers provide these information. Therefore, either an additional bypass is required to transfer scheduling data from C to A, or the local scheduler is reduced to starting and monitoring jobs while scheduling is delegated to the proxy itself.

The proxy has an OGSA-compliant user interface for job submission, monitoring, etc., so that existing tools will still work. Communication between proxies relies upon application-specific web services. In the middle-term, a web portal

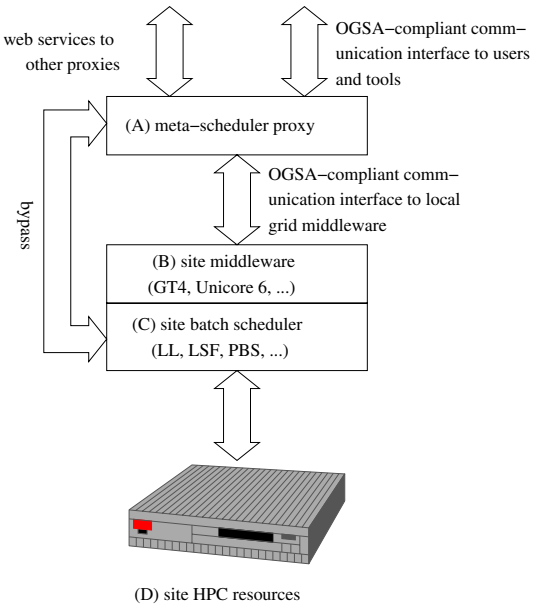

Fig. 1. Architecture of a distributed meta-scheduler at the site level. Abbreviations are explained in: OGSA [9], GT4 [2], LL [10], LSF [11], PBS [12].

will be developed that allows casual grid users to easily use the meta-scheduler.

To improve the local schedule, proxies periodically announce under-utilized site resources to the grid on a time scale of minutes, and simultaneously try to migrate jobs from their input queue to interested neighbors to be computed there. Interested neighbors are either discovered by actively searching in the grid for available resources or by responding to announced resources from other sites.

The decision which jobs to migrate is performed by the site based on local information and policy. Exemplary candidates for job migration are 1.) jobs that would otherwise reduce utilization of the site's resources by fragmenting the batch schedule of the site, 2.) jobs that access data at distant sites during execution, and 3.) jobs that require resources not available locally. The inclusion of the latter allows jobs to be submitted by a user at his home site disregarding resource availability which is the core benefit of grid computing.

To further improve a local schedule, each site carefully selects the jobs it accepts from remote sites. Job acceptance is based on local information and policy and reflects the subsidiarity principle we foster in our architecture.

Although the optimization by migration of jobs will mostly lead to shortened runtimes, this is not always the case. The procedure described is beneficial for computing centers only, however, it can be easily adopted to include the user's interests as well. If each migration decision is based on a weighted combination of criteria advantageous to the site and the user, a balance is attainable. The hard part will be to find a weighting that neither favors the site nor the user exclusively. So, the first step is to automate the process of weighting. However,

personal experience is still required to set the weights for job priority, job length, resource demand, etc. correctly in a final decision.

In this paper we focus on a method for resource discovery in a grid environment. Unfortunately, this is only one step towards a successful migration of a job to another resource. A remote site that replies with a resource offer to a request will usually either 1.) reserve the offered timeslot until acknowledged or cancelled by the requesting site, 2.) reserve the offered timeslot for a specified amount of time, or 3.) provide no reservation at all.

In the first case, a requesting site will have to guarantee a cancellation of unused offers in a timely fashion or all available resources will be blocked soon. In the second case an offer will ultimately time out but an explicit cancellation is still advisable or the utilization of resources will suffer. Finally, in the third case migration attempts will fail, if an offered resource is no longer available when the requester tries to accept the offer.

Extending our supposition of an unstructured grid of locally controlled sites to this problem, we assume that we'll have no control over the type of resource offer returned by a remote meta-scheduler proxy. The implementation will have to support all three types of offers, although the actual algorithm to be used is still being researched.

The following sections will focus on how a meta-scheduler proxy discovers external resources.

III. Locating resources

A. DHT-based P2P search algorithms

Distributed hash-table (DHT) algorithms like CAN [13], Chord [14], Pastry [15], and Tapestry [16] define the state-of-the-art in P2P networks. These algorithms can efficiently map a search request to a target site by hashing a search request and providing a look-up table that maps the hash value to a node-address in the network.

While this idea works very well in P2P networks, where the typical search requests consists of a file name, it is harder to translate the concept to a computing grid. A request for computing resources consists of multiple objectives that need to be satisfied: types of hardware and operating systems, versions of installed software and libraries, etc. In addition to these fixed objectives there are dynamic objectives where an exact match is not desirable as a job's requirements can be specified by a range of values: number of CPUs, size of memory or hard-disk space, etc.

These *multidimensional requests* and *ranged requests* are not directly supported by the aforementioned DHT-algorithms. In the recent years, a lot of work has been put into translating the DHT-idea into the grid-environment. New algorithms have been developed like MAAN [17], Dgrid [18], Pub/Sub-1 [19] and Pub/Sub-2 [20], Meghdoot [21], XenoSearch-II [22], and many others. An excellent overview and taxonomy of grid-enabled DHT-based algorithms is given in [23].

Many of the new algorithms use CAN, Chord, or Pastry for routing of the search requests. These methods employ additional techniques to map the multidimensional and ranged requests to a type of request supported by the routing layer. E.g. the mapping for multidimensional requests is achieved by using multiple routing layers, reducing multidimensional to 1-dimensional requests with the help of space filling curves, dividing the multidimensional space with tree-based methods into regions which are then assigned to individual nodes, or applying adapted hashing functions. On the other hand, ranged queries are supported by, e.g. using locality-preserving hashing functions, separate requests for each value in a range or each predefined sub-range the request intersects with, etc.

B. Forwarding-based P2P search algorithms

Before the advent of the DHT-based methods, many algorithms were based on the idea of each node forwarding a search request to its neighbors in the network. These algorithms usually suffered either from poor results or excessive network overhead: the quality of the search results depended directly on the number of nodes contacted in the grid and a reduction of the network traffic by selective forwarding of the requests lead to a reduced number of hits.

Nevertheless, given the complexity introduced into a system by grid-enabled DHT-based algorithms their usage seems like breaking a butterfly on the wheel in smaller grid-environments. With less than a few hundred sites in a grid the performance of forwarding-based search algorithms can at least measure up to the more advanced methods. The increase in network traffic that the older algorithms induce can be reduced by a situation-dependent selection from a portfolio of different algorithms.

Therefore, we restrict ourselves to non-DHT algorithms. Since each of these algorithms is not efficiently enough as stand-alone method, we combine them to a hybrid approach.

1) Breadth First Search (BFS): The initial site forwards its request to all of its known neighbors, attaching a fixed *time-to-live* (TTL) to the transmission. Each neighbor processes the request, decrements its TTL by one, and forwards it to all of its own neighbors in turn, and so on, until the request's TTL has reached zero. This behavior is also called *flooding* and the TTL is often called the *depth* of the BFS. A site's response takes the reversed path of the original request back to the source site.

The BFS is guaranteed to discover a match if the number of hops between requesting and responding site is less than the depth of the search. Unfortunately this result comes at a high price: the number of messages transmitted in the network increases exponentially with the TTL of a search request. The Gnutella P2P network initially used a BFS with a TTL of 7 [24], which resulted in serious scalability issues when the number of users increased.

As a consequence many variation of the basic algorithm were developed to overcome the BFS's shortcomings. E.g. Modified BFS [25], Intelligent BFS [25], and Directed BFS [24] try to reduce the number of messages send in the network by selecting only a subset of neighbors to which to forward a request. The authors of Iterative Deepening [24] propose to

start with a small TTL and successively increase the depth until result are discovered.

2) Random Walks: The Random Walks, or k-Random Walks, algorithm is yet another variant of the BFS, in which the first node transmits its request (including an assigned TTL) to k randomly selected neighbors. Each recipient processes the request, decrements the TTL, and forwards the message to a single, random neighbor of its own, until the TTL reaches zero. The k independent requests traveling through the network are called *walkers*. Responses return to the initial site on the reversed path of the request message.

In contrast to the BFS algorithm, there is no guarantee that Random Walks will find results, even if a match does exist within n hops from the searching site (where n is the TTL of the request). On the other hand the number of transferred messages is only $O(nk)$. Depending on the field of application it has to be decided, whether the decreased number of messages is worth the reduced chance of finding a result.

3) Adaptive Probabilistic Search (APS): The APS [26] is a modified version of the Random Walks algorithm. Each site maintains an index which indicates the relative probability for each neighbor to be chosen as forwarding target of an incoming request based on the searched object. If a site possesses a requested resource, it will terminate the walker and will return a response to the requesting site afterwards. Otherwise the site randomly selects a single neighbor and forwards the request to it. The random selection is subject to the probabilities for the particular search topic or uniformly distributed in the case of a topic never seen before.

Then, the index is updated either in an optimistic or pessimistic way by incrementing or decrementing the relative probability of the selected neighbor for the requested object. An optimistic way means that the algorithm does expect to find a match, the pessimistic way assumes the contrary. This policy is chosen by the initial requesting site and constant for each individual walker. If a walker terminates successfully (unsuccessfully) and its policy was optimistic (pessimistic), then the result can be returned directly to the site, which started the request in the beginning; if its policy was pessimistic (optimistic), then the response has to traverse its search path in reverse order and correct the probabilities at each site accordingly. In its pure version the APS never changes its policy at runtime.

A variation of the algorithm called *swapping-APS* (s-APS) watches the ratios of successful vs. unsuccessful searches for each topic and dynamically switches between the policies to reduce the number of necessary corrections after a walker terminates. Another variant called *weighted-APS* (w-APS) takes the distance between the site and a discovered object into consideration when adjusting the probabilities. This favors nearby search results over results further away.

4) Distributed Resource Location Protocol (DRLP): The DRLP [27] augments the original BFS by a learning feature, which can drastically reduce the number of transmitted messages for searches for an object discovered previously. Each site in the grid maintains an index, which indicates presumed locations of a searched object and is updated regularly with the information extracted from forwarded response messages.

A received search request is first matched against locally available resources. A successful match terminates the search and a response is sent along the reversed path of the request; otherwise it is checked, whether the local index contains any entries for sites, that might possess the searched object. If such a hint exists, the request is forwarded to the indicated site; if the look-up in the local index fails, then every neighbor of the current site receives a copy of the request as long as the TTL has not reached zero.

To update the indices with new information and to establish the indices in the first place, each site analyses the response messages that it receives. These messages correlate to requests previously forwarded. It extracts the topic of the search and the site, where a result was finally discovered, and adds this information into the local index. Further searches for the same topic will now be directly forwarded to the extracted site.

Additionally, the authors of DRLP suggest, that a site that no longer holds a resource, should notify sites from which it receives requests, so that they may remove the corresponding entry from their index. This allows for "unlearning" of hints, that are no longer valid.

5) Gnutella UDP Ext. for Scalable Searches (GUESS): GUESS [28] is an algorithm completely different from the procedures described before, as it follows a non-forwarding search concept. Each site maintains two different lists, where it collects information about other sites in the grid: a fixed-size *link-cache* and a *query-cache* of (in theory) unbounded size.

The link-cache is used as the "neighborhood" of a site, which it updates constantly by sending *ping* messages to each listed peer in turn. On receipt of such a ping message, the other site replies with a *pong* message, that includes a subset of its own link-cache. These new peers are simply added to the link-cache of the pinging site or used to replace outdated entries. This mechanism keeps the network connected, which is especially important in a P2P environment, where users often enter the network and then disconnect again after a short period of time. Due to the frequent changes in the topology, the link-cache's size should be kept small in this environment, to guarantee regular pings to each listed peer.

The query-cache serves as a list of peers, that are contacted during a search for resources. Initially empty, it is seeded with entries from the searching site's link-cache as initial values. During a search, the algorithm sequentially selects each entry in the list and sends the corresponding peer the request. Every reply, whether positive or negative, is accompanied by another excerpt from the responding peer's link-cache. This new list of sites is merged into the searching site's query-cache.

Upon receiving a positive reply, the searching site can decide whether it immediately stops querying further peers or contacts more peers to accumulate a set of resources to choose from. Similarly, in the case of a negative reply the searching site can at every point abort the search and give up, without any excess messages sent. This is the major advantage of the GUESS algorithm, that the number of messages in the grid is

reduced to the minimum a site considers necessary.

The effectiveness of the algorithm is mainly dependent on the order in which the searching site iterates through its query-cache, the method by which a site selects the subset of its link-cache to return to the searching site, and the threshold of positive or negative replies before a search is concluded or aborted. We feel that the potential for misuse, e.g. the flooding of the grid by a malicious site, is negligible in a scientific environment, where all parties are interested in a working grid.

C. Situation-dependent algorithm selection

In this section, we'll classify different situations in a grid that trigger a search for resources or jobs and we'll explain which aspects to consider before choosing one distinct search algorithm in one of these situations.

1) Communication situations: As previously described, there are two different cases in which a search in the grid is triggered: 1.) to find available resources matching a job's requirements, and 2.) to discover jobs to execute on an idle resource. A search of type 1 is usually started by the site, whose job queue contains the job for which alternative resources are sought. The issued request message will contain sufficient information about the job, that any receiving site's meta-scheduling proxy can come to an informed decision, whether it is beneficial to make an offer or not. The result of the search is a set of resource offers from remote sites. This set is used by the original site to determine, if a migration of the job is worthwhile.

Searches of type 2, on the other hand, are started by sites, that own under-utilized resources. An information message containing a description of the available hardware is distributed in the grid; any site with jobs in its job queue, that could benefit from the offered resources, can reply to the original site and negotiate a migration of jobs. A search of type 2 differs from a type-1 search in that the searching site will not necessarily receive a feedback. Receiving sites are not required to acknowledge their receipt of a resource-offer which means that there there is no answer-phase in the second type of search.

The reasoning behind this difference is, that in the case of a type-1 search even a negative answer can provide information to the search algorithm. If a site returns no offer because it owns no resources as required by the job, then it can be safely excluded from future searches for this kind of hardware. The fact, that a site possesses matching resources but its hardware is currently fully utilized, can be used to trigger a short cool-down phase before further requests are directed to it. On the other hand, a negative reply is almost useless for type-2 searches since the jobs in a site's job queue are dynamic and a new job with previously unseen requirements can be submitted at any time.

Due to this difference, we call type-2 searches *resource announcements*, too, as they don't represent a "search" in the usual meaning of the word.

Type-1 and type-2 searches can be further subdivided by considering the urgency of a request. We classify type-1 searches as either *necessary* or *optional*. A necessary search seeks resources for a job, that has currently no assigned hardware, usually because its requirements can not be fulfilled at the site of its submission. Other reasons include unplanned downtimes of previously assigned resources or the cancelling of a job's reservation by a site's administration. Without a positive search result (followed by a successful job migration to one of the discovered sites) such a job will not be able to execute.

A search is called optional, on the other hand, if the corresponding job already has its assigned set of resources and the search request is started merely to improve the job's performance. The investment of network bandwidth and computational resources is still justified as users expect the access to better hardware and reduced runtimes from the grid. Nevertheless, a positive outcome is of a lower priority than it is for necessary searches. In the absence of unexpected downtimes, failures or administrative intervention, such a job will have its chance to run eventually and any optimization is voluntary.

The classification of resource announcements is based on the point in time, at which a resource will become available. We speak of *immediate* and *future* availability of resources, whereas the denomination "immediate" can include hardware that becomes idle "very soon". The exact length of the time interval that is considered to be "immediate" is situation-dependent as described in the following paragraphs.

Hardware usually becomes available immediately, if an executing job stops earlier than expected and before its assigned timeslot runs out. This can happen, for example, due to an overestimation of the runtime by the user, termination because of errors, or cancellation of the job by its owner or an administrator. As well, a job that is cancelled shortly before it is supposed to be started by the batch scheduler can result in short-term under-utilized resources. If the allocation of jobs from the site's job queue via backfilling is impossible or undesirable, than a search request for matching jobs can be issued to the grid.

Future resource availability denotes the remaining cases, where idle capacities on a site's hardware can be predicted in a timely manner. This is usually the case, e.g. if a computing center reserves fixed time-slots for usage by the grid community, an advanced reservation is cancelled early, or a batch scheduler anticipates timeslots that it can not allocate from its job queue.

The difference between both classes of resource announcements lies in the urgency, by which the meta-scheduler requires the job submissions from interested sites. An immediate resource availability requires fast results, preferably from nearby neighbors in the grid to avoid long migration delays. Every second wasted in an extensive search, reduces the time remaining for a candidate job's computations, as the following job is probably already scheduled.

The rather fuzzy definition of "immediate" as "situation-dependent" stems from the ambiguous urgency of an announcement, which depends e.g. on the size of the available

time-slot and the requests of the jobs typically found in the grid environment. Even a resource predicted to be idle in a few hours might require an urgent decision, if most of the jobs rely on large datasets, that have to be transferred between sites during a migration. On the other hand, hardware available now, but for a rather long interval, in a grid with mostly small jobs allows for a less urgent and more thorough search.

2) Selection of algorithms: Forwarding search algorithms have been shown to work in a grid environment before [29], but the individual advantages and disadvantages of each algorithm persist, whether they are applied to a computing grid or a P2P network: guaranteed search results but exponential increase in messages with growing search depth (BFT), linear increase in messages but little chances for success (RandomWalks), good chances for success but additional management overhead (s-APS, DRLP), etc. Every algorithm has its areas of application, but will be a suboptimal choice in other situations.

As a solution to this problem, we suggest to support multiple search algorithms, that will be chosen according to the situation, in which the search is triggered. To this end, we employ the classification into necessary and optional searches resp. announcements for immediate and future availability of resources. Future research will focus on an algorithm, that automatically sorts situations into these four classes by weighting different criteria, but for the moment we will just avoid this problem and assume the sorting already happened.

a) Necessary searches: In these situations the number and quality of discovered results directly influence the chances of a job to be executed. Therefore, it seems reasonable to employ algorithms, that produce many results of good quality, even if this means to invest additional time, bandwidth and resources. For a typical job this kind of search is triggered only once upon submission, if no extraordinary events result in the job's loss of its allocation later on.

The number of necessary searches can be further minimized, if a meta-scheduler proxy tries to dedicate local resources for a job immediately upon submission. This obviously implies, that the requirements can be fulfilled at the job's home site. Without special needs like advanced reservations, deadlines, etc., that prevent simple backfilling, this should be possible in many cases, although the reserved timeslot may be little attractive from a user's viewpoint.

Algorithms that seem suitable for application in this situation are, for example, s-APS and DRLP. Both methods perform rather randomly in the initial phase, but become increasingly effective with time, as the per-node knowledge base is filled with more data. We suggest to use data about existing hardware at a site as the routing information for these algorithms. Since hardware rarely changes, this obviates the need for frequent updates in the distributed data base. By directly routing search requests to sites, where the hardware does exist, the algorithms can increase the chance, that a suitable migration target is found. As the information spreads through the grid, searches will become shorter and shorter, allowing these algorithms to be used for other situations as well.

b) Optional searches: Despite its name, the class of optional searches is the dominant type of search request in the grid. Assuming that at least one resource can be found for every job upon submission, all further requests to improve the performance of a job are voluntary and belong into this class. Therefore, an algorithm should be used, that is less costly than the algorithms employed for necessary search requests.

The RandomWalks algorithm, augmented by a variable number of walkers, that are adjusted according to the available bandwidth, is one candidate for this category. Its fixed number of messages, that can be regulated via the TTL and the number of walkers, allow for a predictable cost for the grid. Although results can not be guaranteed with this method, this is not a problem for optional searches.

Another option is the GUESS algorithm, following the same argument, that the cost for the grid is predictable. The sequential contacting of the sites in the query-cache can be aborted at any time, which allows for exact steering of the number of requests sent. The method for maintaining the link-cache can be interesting to keep the grid connected, independent of an application of the GUESS algorithm.

Finally, if either s-APS or DRLP are used for necessary searches the same algorithm can be applied to optional searches as well. These methods execute very efficiently after the per-node knowledge bases have stabilized. The cost of the database maintenance will occur anyway, so that there is little disadvantage in using them for optional searches as well.

c) Immediate resource availability: It is especially important to find jobs fast for resources available immediately, as every second invested into finding the perfectly matching job not only wastes resources, but also reduces the computing time available to any candidate job. Announcing unutilized hardware using the breadth-first search guarantees to inform all sites within the the number of hops specified by the used TTL. Since only interested sites will reply with a job submission, the response phase of the BFS is reduced to the minimum. The locality of the search request will provide the additional advantage, that discovered jobs exist on sites in the vicinity of the searching site. If virtual neighborhood corresponds to nearness in reality, the migration of a job's data will benefit from short transmission routes.

Alternatively, the DRLP algorithm can be used. The information collected by learning algorithms like DRLP or APS about existing resources at a site can be used to efficiently route resource announcements. It can be assumed, that all jobs with specific hardware needs will reside in the job queues of sites where this hardware exists, since otherwise their home site will issue necessary searches until a suitable site is discovered and a migration succeeds. Therefore, by routing announcements to sites with identical resources the chance of a positive hit is increased.

d) Future resource availability: Compared to the previous case of resource announcements, the need for speedy results is less pressing, if the availability of a resource can be predicted in advance. It is advisable to distribute the infor-

mation about the unutilized hardware to many sites, so that hopefully many candidate jobs will be submitted in response. With a larger set of jobs from which to choose, the chance to find a good match increases significantly.

While the distribution of announcements with the breadth-first search guarantees a wide exposure, the cost is still prohibitively large for high TTLs. But the fact, that the time available for a search is known to the meta-scheduler, provides the unique opportunity to maximize the results from algorithms like RandomWalks, APS and GUESS, whose runtime can be predicted, too. If learning algorithms were already used for one of the other cases, then this information provides further benefit for APS and GUESS. We opt for APS, as there are no immediate replies when distributing resource announcements, which reduces the usefulness of GUESS's sequential behavior.

IV. Conclusions

We have described an hybrid algorithm for the distributed meta-scheduler of the DEISA grid that efficiently discovers jobs to migrate and resources to be used. It is based on dynamically selecting the most appropriate method out of a predefined set of algorithms known from peer-to-peer networking. The resulting exchange of jobs between sites leads to improved resource utilization and shorter turn over times. The hybrid algorithm is currently implemented as part of the DEISA grid scheduler.

Future work will focus on an efficient yet simple algorithm for the multi-criteria decision which job to migrate and which resource to use. The goal is to take the interests of users and grid community into account, while still allowing the site administration to retain control over their hardware and enforce local policies.

References

[1] (2007, May) Uniform Interface to Computing Resources (UNICORE). [Online]. Available: http://www.unicore.eu

[2] (2007, May) Globus Toolkit. The Globus Alliance. [Online]. Available: http://www.globus.org

[3] (2007, May) Gridway. The Globus Alliance. [Online]. Available: http://www.gridway.org

[4] (2007, May) Platform computing. [Online]. Available: http://www.platform.com

[5] (2007, July) Open Grid Forum (OGF). [Online]. Available: http://www.ogf.org

[6] (2007, July) Organization for the Advancement of Structured Information Standards (OASIS). [Online]. Available: http://www.oasis-open.org

[7] 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-Verlag, 2000, pp. 191–202.

[8] 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.

[9] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich, *The Open Grid Services Architecture*, Open Grid Forum Grid Final Document, Rev. 1.5, July 2006. [Online]. Available: http://www.ogf.org/documents/GFD.80.pdf

[10] *IBM Load Leveler: User's Guide*, IBM Corporation, Kingston, NY, September 1993.

[11] S. Zhou, "LSF: Load sharing in large-scale heterogenous distributed systems," in *Proc. Workshop on Cluster Computing*, 1992.

[12] R. Henderson and D. Tweten, "Portable Batch System: External reference specification," NASA Ames Research Center, Tech. Rep., 1996.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2001, pp. 161–172.

[14] 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: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM Press, 2001, pp. 149–160.

[15] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. London, UK: Springer-Verlag, 2001, pp. 329–350.

[16] 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.

[17] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," *Fourth International Workshop on Grid Computing*, p. 184, 2003.

[18] Y. M. Teo, V. March, and X. Wang, "A DHT-based grid resource indexing and discovery scheme," in *Singapore-MIT Alliance Annual Symposium*, 2005.

[19] D. Tam, R. Azimi, and H. Jacobsen, "Building content-based publish/subscribe systems with distributed hash tables," in *Proc. of the 1st Intl. Workshop on Databases, Information Systems, and P2P Computing (DBISP2P)*, 2003.

[20] P. Triantafillou and I. Aekaterinidis, "Content-based publish/subscribe systems over structured P2P networks," in *International Workshop on Distributed Event Based Systems*, 2004.

[21] A. Gupta, O. Sahin, D. Agrawal, and A. El Abbadi, "Meghdoot: content-based publish/subscribe over P2P networks," in *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 254–273.

[22] H. Shan, L. Oliker, and R. Biswas, "Job superscheduler architecture and performance in computational grid environments," in *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 44.

[23] R. Ranja, A. Harwood, and R. Buyya, "Peer-to-peer based resource discovery in global grids: A tutorial," *IEEE Communication Surveys and Tutorials*, 2007, to appear.

[24] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *ICDCS '02: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*. Washington, DC, USA: IEEE Computer Society, 2002, p. 5.

[25] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*. New York, NY, USA: ACM Press, 2002, pp. 300–307.

[26] D. Tsoumakos and N. Roussopoulos, "Adaptive probabilistic search for peer-to-peer networks," in *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 102.

[27] D. Menascé and L. Kanchanapalli, "Probabilistic scalable p2p resource location services," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 2, pp. 48–58, 2002.

[28] B. Yang, P. Vinograd, and H. Garcia-Molina, "Evaluating GUESS and non-forwarding peer-to-peer search," in *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 209–218.

[29] A. Iamnitchi and I. Foster, "On fully decentralized resource discovery in grid environments," in *GRID '01: Proceedings of the Second International Workshop on Grid Computing*, ser. Lecture Notes In Computer Science, vol. 2242. Springer-Verlag, 2001, pp. 51–62.