Heuristic Scheduling of Grid Workflows Supporting Co-Allocation and Advance Reservation

Joerg Decker, Joerg Schneider
{decker,komm}@cs.tu-berlin.de
Technische Universitaet Berlin, GERMANY

***Abstract*— Applications to be executed in Grid computing environments become more and more complex and usually consist of multiple interdependent tasks. The coordinated execution of such tightly or loosely coupled tasks often requires simultaneous access to different Grid resources. This leads to the problem of resource co-allocation. Efficient and robust scheduling algorithms have to be developed that can cope with the Grid's large-scale distribution, a high number of competing and demanding applications, the inherent resource heterogeneity and the often limited view on resource availability. In this paper, we present two heuristic scheduling algorithms that are based on a well-known list scheduling algorithm and both support co-allocation and advance resource reservation. Our first algorithm preserves the run-time efficiency of Greedy list schedulers while the second approach incorporates more sophisticated search techniques in order to achieve better results with respect to the performance metrics. Both algorithms have been implemented within a Grid simulation framework. An extensive simulation study was conducted to evaluate and compare the performance of both algorithms. It showed the general suitability of our enhanced list scheduling heuristics within heterogeneous Grid environments.**

I. Introduction

Today's *Grid resource management systems* (Grid RMS) provide uniform and transparent access to large pools of heterogeneous resources. Earlier systems allowed users to request and reserve resources manually. The usage models of newer Grid management systems support the automated execution of complex applications consisting of multiple interdependent parts. Hence, Grid users do not have to directly specify their resource demands. Instead, they submit a high-level description of the application. Fig. 1 shows an example of such a so-called *Grid workflow*.

Descriptions of Grid workflows are abstract in the sense that low level details of the concrete resources are hidden. The actual structure of the Grid is transparent to the user and the mapping of the workflow elements to appropriate Grid resources is the task of the Grid RMS. The user's task is to model the workflow *activities* to be carried out, to define dependencies between them and to specify additional information, e.g. temporal constraints such as deadlines that must be considered during workflow execution.

Regarding Grid resource management, the support for service level agreements (SLAs) became a crucial concept. Using SLAs, resource providers and resource users are able to negotiate binding contracts that define certain service levels. For time-critical workflows, e.g. workflows annotated with deadlines, a user might want to insist of SLAs that contain guaranteed execution finish times.

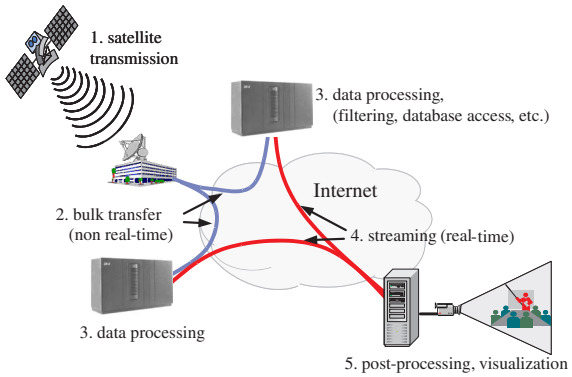


Fig. 1: Example Grid workflow: A complex Grid application with time-dependent tasks.

Advance reservation has been identified as a key technology in order to be able to guarantee that enough resources are available for time-critical workflow execution [1]. For each part of the workflow, the necessary resources will be reserved in advance. Furthermore, to ensure that all input data is available prior the execution of the activities that depend on it, network bandwidth for the transmission has to be reserved.

Some parts of common Grid workflows might have to be executed simultaneously. While advance reservation is a means to book resources for coordinated executions, the Grid RMS has to be able to find common start times and durations for such *co-allocations*.

Another issue that makes the resource reservation complex is that heterogeneous Grids are federations of resources of different types and capabilities, e.g., varying CPU speeds. Hence, the actual execution time of the workflow activities are determined by the capabilities of the target resources. This influences the start times of dependent activities and thus the duration of the whole workflow.

Generally, the problem of finding feasible workflow schedules and reserving appropriate resources while satisfying all constraints is complex. With an increasing number of workflow elements, the already multi-dimensional search space becomes even larger and efficient algorithms are needed for fast admission. The heuristic algorithms we developed in this work are based on the following semantics of the *admission* of user requests: The user gets a positive answer, if and only if a valid schedule was found. The resources will be

reserved according to the schedule. Subsequently, the user will be informed about the guaranteed access. A rejection of the request does not necessarily mean that there is no valid schedule. Though, it is the objective of the Grid RMS to reduce the number of unnecessarily rejected workflows.

We analyzed existing scheduling algorithms for heterogeneous environments and extended the well-known list scheduler HEFT [2] in order to support advance reservations and co-allocation. We developed two algorithms. The first preserves the time efficient greedy structure. The second incorporates more elaborated search techniques to achieve better results w.r.t. our optimization goals.

In the following section we describe the application environment and define the problem. After presenting related work, we introduce the enhanced scheduling algorithms. In section V, we discuss the results of our extensive simulation study based on which we evaluate the performance of our algorithms.

II. Problem Definition

In this chapter, we present the modeling framework which we use to design and develop the scheduling algorithms. It basically consists of the workflow model, the Grid resource model and the model for advance reservation. Furthermore, we introduce the scheduling problem to be solved.

A. Workflow Model

Our approach to modeling workflows is based on graphs. In fact, our model is similar to the traditional task graph models commonly used in the scheduling literature [3], [4], [5]. Essentially, a task graph is a directed acyclic graph (DAG) consisting of interdependent tasks or nodes. In the following, we define our workflow model and introduce the way we describe resource co-allocation for parallel activities.

In analogy to the workflow definitions of the Workflow Management Coalition (WfMC) [6], the elements of our workflow models are also termed *activities*. An activity might be any working package or task that needs resources during workflow execution. In the vast cases, the activities correspond to computational tasks to be executed on compute resources.

Formally, a workflow can be defined as a pair $W = (A, D)$ where A is a set of activities, and $D \subseteq \{(u, v)|u, v \in A\}$ is a set of dependencies between the activities.

1) Activity Weights: Activities and dependencies are weighted. As in the task graph model, activities are assigned a *load* weight. Based on the load weights, the Grid RMS can determine the execution time of an activity. The execution time may depend on the capabilities of the target resource. E.g., a compute activity takes $\frac{load}{resource\ speed}$ time units to finish, where $load$ is an estimate of the amount of instructions of a program. There might be cases in which the execution time does not depend on the resource speed or other resource specific factors. Then, the weight of an activity can be a fixed time value.

The temporal requirement of an activity determines how long a resource is needed. Additionally, there must be an activity weight that expresses *how much* of the resource it claims. This is important in case the resource is space-shared [7]. A Grid resource can be associated with a fixed capacity value and a corresponding capacity value of the activities is used to quantify activity requirements. The speed and the capacity of a resource, e.g., CPU speed and amount of CPUs, and the load and required capacity of an activity, e.g., number of instructions and number of parallel processes, are fixed. Modable activities [8] with a flexible capacity requirement are not considered.

A compute activity for instance might require two execution hosts each of which has four CPUs, which makes a total required capacity value of eight.

2) Dependency Weights: The compute activities carry out their work using one or more input data items. Therefore, data must be transferred from its current location, e.g., central repositories or the hosts where it was produced, to the target host of the applications' activities that rely on it. In our framework, data dependencies are modeled as directed edges between two activities. An activity can produce an arbitrary number of data items and each data item has an estimated size. The number of activities depending on each data item is not limited. It is also possible that an activity depends on more than one data item produced by a single particular activity. Note also that the data dependency model stays the same in case the activities have other synchronous activities as introduced in the next section.

3) Co-Allocation Model: The classical DAG model is extended such that co-allocation of resources for parallel activities can be modeled. For modeling parallel activities, we introduce the notion of synchronous activities. A *synchronous dependency* is represented by a hyperedge and is a connection between any number of activities in the workflow. Formally, sync edges are sets of activities, defined as $Sync \subseteq 2^A$ with 2^A being the power set of all activities. A workflow that contains synchronous activities automatically becomes a hypergraph $W = (A, D, Sync)$.

With the invention of synchronous activities, the Grid user is able to define an additional temporal constraint. All synchronous activities have to be executed simultaneously within the same time window, the *co-allocation window*. For the Grid RMS, this means that resources have to be allocated for all synchronous activities across the same time span.

The synchronous activities communicate during their execution with each other. Therefore, enough network bandwidth has to be allocated. The network requirements of sync activities are represented by *channel dependencies* in our model.

Let $sync$ be true if activity a_i and activity a_j are synchronous:

$$sync(a_i, a_j) := \exists S \in Sync : a_i \in S \wedge a_j \in S$$

Then $C \subseteq \{(u, v)|u, v \in A \wedge sync(u, v)\}$ is the set of channel dependencies between synchronous activities. Using the set of channel dependencies, we can extend our workflow model to $W = (A, D, Sync, C)$.

The weights of the channel dependencies characterize the required communication channel. The type of weights used here depends on how detailed the network resources are modeled. Some models neglect the latency while almost all

incorporate the bandwidth of the network link. In our model the required bandwidth will be used to characterize the channels.

B. Grid Resource Model

Generally, the Grid consists of an arbitrary number of resources. The intention is that the model description is not limited to one resource type. Many types of resources could be part of a Grid, e.g., scientific instruments, storage resources, visualization devices, etc. The most common ones though found in today's Grid infrastructures are *compute resources* R_C and *network resources* R_N. The network resources determine the structure of a Grid since they connect resources of other types, e.g., a network resource might be a communication link between a compute and a storage resource.

A Grid can be described by a graph, where the network resources R_N represent the edges between the other resources R

This way, arbitrary networks can be modeled. The links characterize the whole communication path between two resources as point-to-point link.

Different types of resources may have different *attributes*. E.g., compute resources are characterized by the number of CPUs, the CPU speed, whereas network resources are characterized by their bandwidth, respectively their latency.amount of storage capacity.

C. Advance Reservation Model

We use an advance reservation approach for managing the resources. Advance reservations are resource requests made for time periods in the future. During admission, the absolute start time as well as the end time will be fixed. Therefore and in contrast to immediate reservations which are usually made without specifying the duration, advance reservations require the definition of the expected duration or the expected load for a given request. This is necessary to perform reliable admission control, i.e., to determine whether sufficient resources can be guaranteed for the requested period.

Our advance reservation system needs to maintain allocation information in two dimensions, space (resource) and time. We use a slotted time model which means that the time line is divided into fixed-size slots T and all absolute time values are multitudes of the slot length. The advantages of a slotted time model over an arbitrary time model is the easier management and the less external fragmentation.

The time between the request and the actual start time is called *reservation time*. A *book-ahead interval* is specified to limit the reservation time to reasonable small values.

D. Scheduling Problem

We have defined the workflow model and the Grid model. In this section, we use these models to define the scheduling problem.

We basically intend to develop an on-line algorithm which treats the workflows *one by one* and does not reject already admitted workflows. The mapping of one workflow means mapping the workflow elements (activities and dependencies, $A \cup D \cup C$) to Grid resources. The mapping can be defined as

$$\Phi : \{A \cup D \cup C\} \rightarrow \{R \cup R_N\} \times T$$

Note that the mappings of two workflow elements might not be disjoint, meaning that multiple activities for instance can be mapped on the same resource if there is enough capacity available. Generally, the assumption is that one workflow element cannot be mapped on more than one resource, e.g. compute activity a_i is mapped on one compute resource r_j. Furthermore, the resource types, a workflow element can be mapped to, might be restricted.

We have stated that costs for data transfers are taken into account. Though, the data transfers are not explicitly defined, e.g. *transfer 10 Gigabytes byte from host A to host B*. Instead, the data dependencies will lead to data transfers during the scheduling process when it is clear which resources are allocated for activities that have data dependencies in between them. Data dependency d_{01} connecting activity a_0 and a_1 can only be mapped to network resources that connect the target resources of a_0 and a_1. Channel dependencies have to be handled alike.

Our system allows the reservation of resources in *space* and *time*. In the above equations, the spatial dimension of the problem (the question of *where* to map an activity) is represented by the target resource sets R and R_N. For regarding the temporal dimension, each resource allocation must be attributed with a time slot. This means that each scheduled activity has an associated start time $start(a_i)$. The finish time $end(a_i)$ of the activity is estimated by its duration $duration(a_i)$, which could be explicitly given or determined by its load and the resource speed.

Basically, the problem is to find appropriate resources for all activities and schedule them over time.

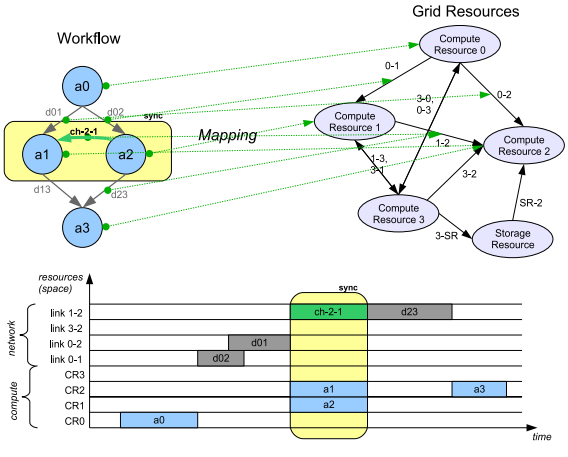


Fig. 2: Illustration of Mapping Workflow Elements to Grid Resources

1) Scheduling Example: Figure 2 illustrates the mapping of a workflow to Grid resources. The simple workflow in the upper left corner consists of four activities and data dependencies between them. Activity $a1$ and $a2$ are synchronous. The corresponding synchronous dependency is drawn as rounded

box. An arrow between $a1$ and $a2$ (channel dependency ch-2-1) indicates that network resources might need to be co-allocated. The horizontal arrows from the left to the right show an example mapping for the workflow elements to Grid resources. We can see that activities are mapped to compute resources and data dependencies are mapped to network resources.

The lower half of the figure shows a bar chart. The mapping is extended to the temporal dimension. The chart also illustrates that the precedence order in the workflow DAG is preserved and resources are co-allocated for synchronous dependencies.

III. Related Work

Advance reservation is an important allocation strategy that provides simple means for reliable planning and co-allocation of heterogeneous resources. Besides flexible support for co-allocations advance reservations also have other advantages such as an increased admission probability when reserving sufficiently early and reliable planning for users and operators. In contrast to the independent usage of several different resources, where also queuing approaches are conceivable, advance reservations have a particular advantage when time-dependent co-allocation is necessary, as shown in Fig. 1 and 2. Advance reservation support has been proposed for several management systems for distributed and parallel computing [1], [9], [10]. In [1], advance reservations have been identified as being essential for a number of higher level services such as the support of SLA.

Complex applications requiring multiple resources are becoming a major application of the Grid [11]. Such Grid workflows increase the complexity of the allocation process and efficient scheduling and allocation schemes have to be developed.

A. DAG Scheduling Algorithms

As we have stated above, the workflows are described in form of weighted DAGs. The problem of scheduling DAGs on a set of resources, e.g., processors, cluster nodes, has been studied extensively by many research groups. In its general form, the scheduling problem is NP-complete [12]. Besides some special cases, e.g. for restricted graphs [13], [14], the problem cannot be solved in polynomial time and hence many heuristic algorithms have been developed.

Earlier algorithms focused on homogeneous environments, such as dedicated parallel computers that have hundreds of equally fast CPUs [15], [16]. Later research efforts aimed at finding algorithms for heterogeneous computing environments [17], [18].

If communication costs are taken into account, the network topology becomes important. Some algorithms assume a fully-connected network, others allow arbitrary topologies of links having an arbitrary bandwidth. Another significant question is whether network contention is regarded [19], [20].

B. List Scheduling Heuristics

Most DAG scheduling heuristics are based on classical *list scheduling* techniques [21], which basically are Greedy Best-First methods without Backtracking capabilities. In general, the three steps of static list scheduling algorithms [22] are

1) *activity priorities computation*, whereas the activity priorities should characterize the overall behavior of an activity, e.g. the hierarchical order, its dependencies, whether it is communication or computational intensive [23].
2) *activity selection* according to priorities
3) *resource selection*. Selecting the ”best” resource for the previously selected activity. Continue with step 2 until all activities are scheduled.

The activities to be scheduled are stored in a sorted list. The list is either static if it is constructed once or dynamic if the priorities of the activities may change during scheduling.

List scheduling heuristics vary in the way, the priorities are assigned to the activities. Two commonly used activity priorities are the so-called *t-levels* and *b-levels*, which are both calculated recursively. T-level calculation starts at the top, b-levels are generated starting from the bottom level of the graph.

The t-level of an activity a_i is the length of the longest path from a activity that has no predecessor (*entry node*) to a_i, a_i not taken into account. The length of a path is determined by all node and edge weights, i.e., execution and communication costs, along the path. Since all weights are summed up recursively, the t-level value corresponds qualitatively to the earliest starting time of activity a_i. The b-level of an activity a_i is the length of the longest path from a_i to an exit node, a_i's weights included. In [16], the authors state that scheduling in ascending order of t-levels tends to account for the topological order of the DAG while scheduling in descending order of b-levels tends to prioritize critical path activities.

An issue during rank calculation is that the costs can often only be estimated. E.g., in a heterogeneous environment the time it takes to execute a computation strongly depends on the speed of the resource it is eventually scheduled on. Here, the rank calculation is mostly based on estimated values, e.g., average CPU speeds, worst case bandwidth values, etc. Though, the way the so-called *rank functions* estimate the costs has a strong impact on whether the optimization goals will met [24].

The HEFT (Heterogeneous Earliest-Finish-Time) algorithm is a well-known list scheduling heuristic with good performance [2]. It is frequently used as a benchmark for new algorithms. The objective of the HEFT algorithm is to find a mapping of activities to resources that minimizes the schedule length (*makespan*). The rank is the b-level adapted for the heterogeneous environment. The execution time of an activity is calculated using the average speed of all resources the activity can generally use and the communication costs are approximated with the average bandwidth of all links in the network. Furthermore, the activity selected by the rank is mapped on the resource that leads to the *earliest finish time* (EFT) for this activity. The EFT is calculated based on the finish times of the already scheduled predecessors and the

actual communication and execution cost for the selected resource.

IV. Workflow Scheduling Algorithms

The algorithms presented here are based on the HEFT algorithm introduced in the previous section. The extension is required because of the extended application model employed. One main difference to the classical task graph model used in the original HEFT algorithm is the introduction of co-allocation modeled as *synchronous dependencies* and the ability to specify (data) channel requirements between two activities.

A. HEFT's Incompleteness

The HEFT algorithm was not explicitly designed for an advance reservation environment. Though, since it is an insertion-based heuristic that greedily searches for available Grid resources while picking the one that leads to the earliest finish time, it can be applied in an advance reservation environment without modifications. However, it becomes clear that such a Greedy search technique may not find a feasible schedule for the workflow.

Situations that lead to an unsuccessful scheduling process and the rejection of the incoming workflow may arise:

- no feasible resources satisfying the activity requirements are available for the activity to be scheduled,
- no free slots are available before the workflow deadline, respectively within the book-ahead interval
- co-allocation: no resource can be found within the given time window, the co-allocation window.

In the original HEFT algorithm, the *book-ahead interval* is assumed to be infinity and no deadlines are specified for the task graphs. This implies that temporal constraints do not lead to an unsuccessful scheduling procedure, even though it is a Greedy algorithm with no Backtracking abilities.

The HEFT resource model is heterogeneous in terms of the resource speeds. Though, no different types of resources can be specified. Hence, each task's requirement is limited to a certain amount of processing power expressed as amount of instructions. This means that, during the scheduling process, there are always feasible resources available, at least in the far future. In our case, situations may well arise in which activity requirements cannot be satisfied.

B. The HEFTSync Algorithm

As an extension of the HEFT algorithm we propose the *HEFTSync* algorithm, which additionally supports co-allocation.

1) Rank Calculation: The recursive rank calculation of HEFTSync is done slightly different compared to the rank calculation of HEFT. The extended approach originates from the definition of synchronous activities and dependencies.

Let $Sync$ be the set of synchronous dependencies of workflow $W = (A, D, Sync, C)$, $S \in Sync$ being a synchronous dependency, i.e., one set of synchronous activities (cp. section II-A.3). The resource requirements of all activities of a synchronous dependency are treated as one requirement set. For each of these requirements, resources are allocated simultaneously (*co-allocated*) stretching across the same time interval, the co-allocation window. This means that $\forall a_i \in S : start(a_i) = start(S) \wedge end(a_i) = end(S)$. The activity of S that consumes the most time determines the duration of the co-allocation window, $duration(S) = end(S) - start(S)$. This implies that all activities of the synchronous dependency have the same priority and are all taken into account when calculating recursively. The set of successors of activity a_i, $succ(a_i)$, is extended such that it includes not only the immediate successors but also all successors of all other activities in the synchronous dependency.

$$succSync(a_i) = succ(a_i) \cup \{a_k | a_k \in succ(a_j) \wedge sync(a_i, a_j)\}$$

Furthermore, an activity without direct predecessors is not regarded as an entry node if it is member of a synchronous dependency with activities that do have predecessors.

2) Resource Selection: As a static list scheduler, the HEFTSync algorithm goes through the sorted list of ranked activities and tries to schedule them selecting appropriate Grid resources. Since b-level rank calculation is used, each activity is scheduled before all activities that depend on it.

The determination of the earliest finish time (EFT) of an activity is adapted to the advance reservation environment. First, for each Grid resource of a matching type, the earliest start time is calculated based on the finish time of the predecessors of the activity, the data dependencies and the actual available bandwidth on the links leading to this resource. The Grid information system or the local resource manager is then queried for the earliest possible reservation after the earliest start time. The actual start time and thus also the finish time depends not only on other activities of this workflow assigned to this Grid resource but is also influenced by other Grid workflows and locally submitted jobs. The activity will be assigned to the Grid resource which offers the earliest finish time.

3) Co-Allocation: Synchronous dependencies have to be handled differently to guarantee that all activities have the same start and finish times.

In order to calculate the earliest start time, the maximum of all finish times of the activities' predecessors has to be calculated. The worst case communication costs for any data transfers to any of the activities in the synchronous dependency is estimated based on data transfer size and the minimum bandwidth. The sum of the maximum finish times and the worst case communication costs mark the first time slot to search for a *co-allocation* window.

The activities will be processed subsequently in the decreasing order of their load. The first activity is processed like a single activity: The Grid resource giving the lowest EFT is determined and the start and finish time of this allocation mark the initial co-allocation window. For the following activities, an allocation with the same start and finish time is searched. If multiple Grid resources are available, the one which would lead to the earliest EFT for this activity is selected.

If, for a given activity, the available Grid resources are too slow, the current co-allocation window limit might be

exceeded. In this case, the window is adjusted increasing the duration and all reservations made for other activities in the synchronous dependency S are canceled and rescheduled using the new co-allocation window.

After all activities have been scheduled, the co-allocation window is fixed. Eventually, the algorithm tries to allocate all necessary communication channels between the activities. If successful, the scheduler tackles the rest of the workflow. If not, all reservations are canceled and the whole workflow is rejected.

C. HEFTSyncBT: Backtracking Extension

The evaluation of the *HEFTSync* algorithm showed (see Sec. V) that most workflows were rejected because no co-allocation could be found which covered all activities of a synchronous dependency or because there was not enough bandwidth available for the data channels. Therefore, we propose an algorithm that combines the efficient greedy algorithm for single activities and a backtracking-based approach to resource co-allocation for synchronous activities.

Not only the earliest feasible allocation slot for each activity that is part of the synchronous dependency but all possible allocation ranges are taken into account. Furthermore, the bandwidth requirement of the channel dependencies is handled together with the other activities.

To reduce the search effort, synchronous activities are handled again in decreasing order of their load. A search tree is built up and many partial solutions are expanded, pruned and dismissed. The first level contains all feasible reservations for the first activity. Initial feasible ranges lay between the latest finish time of the activities part of the predecessor set of the synchronous dependency and the workflow deadline.

Each subsequent synchronous activity produces new partial solutions based on the Grid resources available during the co-allocation window. A partial solution is dismissed as soon as there are no expansions for the current synchronous activity. Then, the second best solution will be further investigated whereas the "best" means the solution that leads to the earliest finish time.

The allocation of the channel dependencies are treated as equally important elements of the synchronous dependency. Though, they are dynamically added to the set of elements to be mapped right after both communicating synchronous activities have been allocated. If no network resources can be found within the current co-allocation window, the search tree can be pruned at this point.

V. Evaluation

The proposed algorithms have been studied using extensive simulations. In the next section, we introduce the simulation environment employed. In subsequent sections, we introduce performance metrics and discuss the results.

A. Simulation Environment

A discrete event simulation of a Grid management system similar to our VRM architecture [1] was used. The simulations were made assuming an infrastructure of seven cluster and parallel computers with homogeneous node equipment, i.e., each job is capable of running on any of the machines involved. Though, the infrastructure was heterogeneous in term of processing speed. The speed ratings of the computer ranged uniformly between 500 and 1100. All resources were fully connected with equal network links.

The simulations serve the purpose of showing the overall performance of the algorithms. Since according to [25] the actual distributions of job sizes, job durations etc. do not impact the general quality of the evaluation results, even when simple models are used, the simulations were made using a simple synthetic workflow generation model. Each workflow was assumed to be reserved in advance with the book-ahead time being exponentially distributed. The workflows were created randomly with following uniformly distributed parameters: number of activities, number of dependencies, number of synchronous activities, number of channels, and shape factor. The shape factor influences the workflow width: the amount of concurrent and synchronous activities on the same level.

B. Request Rejection Ratio and Grid Utilization

As discussed before, the scheduling process may be unsuccessful. Since heuristic algorithms are used, workflow requests might also be unnecessarily rejected. To measure such effects we used the *Request Rejection Ratio*:

$$Request\ Rejection\ Ratio := \frac{|\overline{\mathcal{W}}|}{|\mathcal{W}|}$$

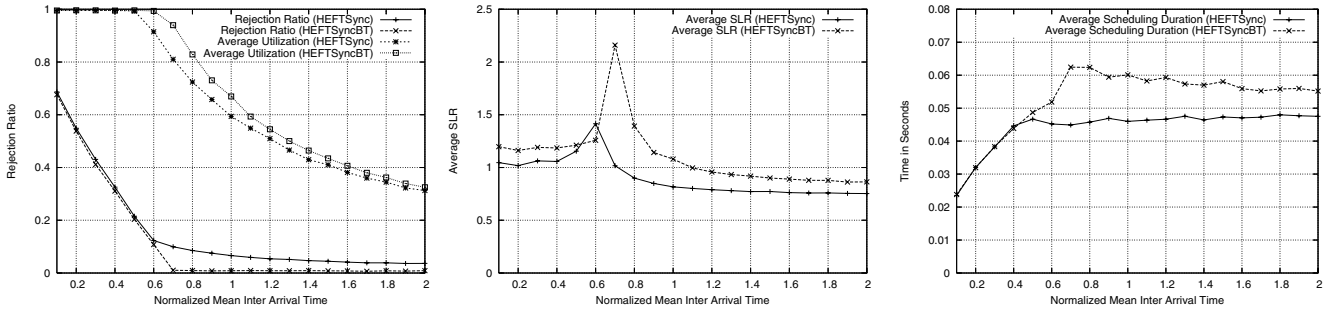
$\overline{\mathcal{W}}$ denotes the set of rejected workflow reservation requests and $\mathcal{W}$ denotes the set of all incoming requests.

Another important statistical value is the *average utilization* of the system. Since network resources are always reserved in the context of compute resource reservations, the average utilization value is computed only for the compute resources.

We varied the mean inter-arrival times in order to simulate different load situations. In Fig. 3a the request rejection ratio as well as the utilization is shown for different mean inter-arrival times. If workflows arrive at a high frequency due to short inter-arrival times, both algorithms utilize almost all resources and a lot of requests have to be rejected.

Basically three phases can be observed in Fig. 3a. For inter-arrival times smaller than 0.5 time slots, the simulated Grid is nearly fully loaded. Each additional incoming workflow is rejected as the linear progression of the rejection ratio shows. In the following phase, the utilization drops because less workflows arrive. Though, not all workflows can be accepted. This phase lasts until about 0.7 time slots. For higher inter-arrival times, the rejection ratio is nearly zero in case of HEFTSyncBT. Using the HEFTSync, there are still a small number of rejected workflows during this lower loaded third phase.

During the first phase, both algorithms perform quite equal. In subsequent phases, HEFTSync shows acceptable low rejections ratios but it is always significantly worse than the enhanced HEFTSyncBT.



(a) Rejection ratio and utilization (b) Average schedule length ratio (c) Average scheduling duration

Fig. 3: Performance of both algorithms depending on the incoming load.

C. Schedule Length Ratio

In the vast amount of articles in the scheduling literature, the schedule length or *makespan* is used as performance metric and the objective of the proposed algorithms is to minimize the makespan.

Since the workflows that are used for performance evaluation are all generated randomly, the absolute makespan greatly varies from workflow to workflow. Hence, in order to get meaningful results, we normalized the makespan to an estimated average-case schedule length. In [2], this concept was introduced as the *schedule length ratio* (SLR). The difference between the original definition and ours is that, for the estimation, we use the average speed of the Grid resources and not the lowest. Furthermore, we integrate average communication costs. The SLR of a given successfully scheduled workflow is defined as

$$SLR := \frac{makespan}{\sum_{critical\ path} average\ case\ exec.\ and\ comm.\ costs}$$

The curves in Fig. 3b show that the intensive search of the backtracking version leads to longer schedules, on average. The SLRs are definitely higher in overloaded and highly loaded systems than in lower loaded systems. Disregarding the peaks for the moment, the fluctuations are not extreme. All values are within 75% and 125% of the estimated schedule length, a fact that justifies the estimation method employed. Though, there are two remarkable peaks in the previously identified second phase. In this phase many workflows could only be accepted by significantly increasing their makespan. In higher load scenarios (lower inter-arrival time), these workflows are rejected because they miss the deadline.

Again, both algorithms have the same characteristic, but the HEFTSyncBT has a significant better performance.

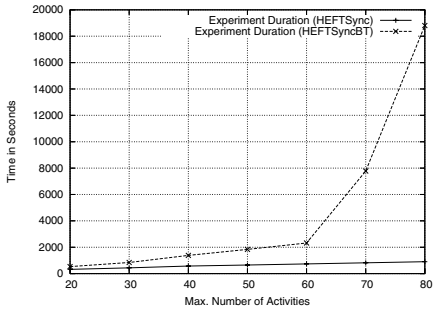


Fig. 4: The run-time of both algorithms for different workflow sizes.

D. Algorithm Run-time

As outlined, HEFTSyncBT is *effective* in finding resources for complex workflows. The remaining question is how *efficient HEFTSyncBT* is.

First we analyze the performance of both algorithms for different load situations. A rough estimation of the algorithm performance is the *average run-time* of a successful scheduling of a single workflow. The Fig. 3c shows that, on average, *HEFTSyncBT* needs more time to schedule the workflows but has the same characteristic.

Another important aspect is the scalability of a scheduling algorithm. Kwok and Ahmad have conducted a comprehensive study in which they compare many task graph scheduling heuristics [16]. In their article, they state that most algorithms are evaluated using small problem sizes, and it is not very clear how the algorithms scale with the problem size. The authors see a need for a performance measure being an indicative parameter of an algorithm's *scalability* as well as of the trade-off between its solution quality and running time.

When we developed the backtracking heuristic, we were aware of the exponential *worst-case* performance of backtracking algorithms. What becomes clear is the fact that the *effective* algorithm HEFTSyncBT might not be as *efficient* as required. The plot in Fig. 4 illustrates the durations of different experiments with respect to different workflow sizes (up to 60 activities on average). In the experiment, 5000 randomized workflows were generated and scheduled. The times are only estimations since the overall duration of the whole experiment is measured and not only the time needed for scheduling. But the exponential complexity of the backtracking variant is highly visible, even though backtracking is only used for co-allocations.

VI. Conclusion and Outlook

We developed a framework for modeling Grid Workflows and Grid resources. Our workflow model is based on the

classical task graph model used in the scheduling literature. We introduced the concept of *synchronous activities* to be able to model parallel activities. Communication requirements can be specified between such synchronous activities in the form of channel dependencies.

Basically, our workflow description is *abstract* meaning that precedence constraints of activities and data dependencies can be defined. The *concrete* Grid resources that are allocated by the Grid Management System are hidden from the user. For data transfers and inter-task communication, network resources need to be reserved. This requires effective and efficient mapping and scheduling algorithms that can cope with the NP-complete combinatorial problem given. Additionally, the workflow execution is restricted by user-given temporal constraints. This imposes the additional issue that during scheduling, workflow deadlines must be met.

After reviewing multiple scheduling approaches, we decided to use list scheduling heuristics as a basis for our algorithms. We extended and modified the well-known *HEFT* [2] algorithm according to our requirements, the support of advance reservation and coping with deadlines and co-allocation.

The two proposed extended algorithms *HEFTSync* and *HEFTSyncBT* are able to allocate and reserve resources for complex Grid Workflows. During the performance analysis of our first approach (*HEFTSync*), we saw that a pure Greedy approach to scheduling synchronous activities still leads to satisfying results. Its scalability is definitely better than more complex search techniques. Though, we have seen that we get worse results with respect to our major performance metric, the requests rejection ratio.

The second algorithm (*HEFTSyncBT*) uses Backtracking in order to find resources for co-allocation. The major drawback of Backtracking algorithms is their worst-case performance leading to exponential complexities. Though, compared to the Greedy approach, the Backtracking version of our algorithm significantly improves the ratio of accepted workflows.

We have seen that the two algorithms have their weaknesses and strengths. Which algorithm eventually suits better depends on the structure and capacity of the Grid as well as on the average and worst-case complexity of the Grid Workflows.

Additional work might be needed to improve the HEFTSync algorithm to cope better with overloaded systems, e.g. it might be useful not to stop searching for free parallel slots after the first unsuccessful try but to continue the search along the time axis. In order to optimize the performance of HEFTSyncBT with respect to its scalability, it might be promising to earlier detect and reject worst-case requests, e.g. workflows having many synchronous activities that exceed the system capacity.

REFERENCES

[1] L.-O. Burchard, M. Hovestadt, O. Kao, A. Keller, and B. Linnert, "The Virtual Resource Manager: An Architecture for SLA-aware Resource Management," in *4th Intl. IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid), Chicago, USA*, 2004.

[2] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, 2002.

[3] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, 1988.

[4] O. Beaumont, V. Boudet, and Y. Robert, "A realistic model and an efficient heuristic for scheduling with heterogeneous processors," in *Proceedings International Parallel and Distributed Processing Symposium., IPDPS 2002*, 2002.

[5] A. S. Grimshaw, "Easy-to-use object-oriented parallel processing with mentat," *IEEE Computer*, vol. 26, no. 5, 1993.

[6] W. M. Coalition, "Workflow management coalition terminology & glossary, document number wfmc-tc-1011, document status - issue 3.0," Workflow Management Coalition, Tech. Rep., 1999.

[7] L. Wang, W. Cai, B.-S. Lee, S. See, and W. Jie, "Resource co-allocation for parallel tasks in computational grids," in *Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*, 2003.

[8] S. Srinivasan, V. Subramani, R. Kettimuthu, P. Holenarsipur, and P. Sadayappan, "Effective Selection of Partition Sizes for Moldable Scheduling of Parallel Jobs," *Proceedings of the 9th International Conference on High Performance Computing*, 2002.

[9] Foster, I., C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation," in *7th International Workshop on Quality of Service (IWQoS), London, UK*, 1999.

[10] Snell, D., M. Clement, D. Jackson, and C. Gregory, "The Performance Impact of Advance Reservation Meta-scheduling," in *6th Workshop on Job Scheduling Strategies for Parallel Processing, Cancun, Mexiko*, ser. Lecture Notes in Computer Science (LNCS), vol. 1911. Springer, 2000.

[11] Next Generation GRIDs Expert Group, "Future for European Grids: GRIDs and Service Oriented Knowledge Utilities – Vision and Research Directions 2010 and Beyond," 2006.

[12] D. Fernández-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, Nov. 1989.

[13] E. Coffman and R. Graham, "Optimal scheduling for two-processor systems," *Acta Informatica*, vol. 1, 1972.

[14] E. Fernandez and B. Bussel, "Bounds on the number of processors and time for multiprocessor optimal schedules," *IEEE Transactions on Computers*, vol. C-22, 1973.

[15] A. Gerasoulis and T. Yang, "A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 16, no. 4, 1992.

[16] Y.-K. Kwok and I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms," *Journal of Parallel and Distributed Computing*, vol. 59, no. 3, 1999.

[17] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C.-L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, vol. 26, no. 6, 1993.

[18] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems," in *Symposium on Reliable Distributed Systems*, 1998.

[19] O. Sinnen and L. Sousa, "List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures," *Parallel Computing*, vol. 30, no. 1, 2004.

[20] G. Sih and E. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogenous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. PDS-4, no. 2, Feb. 1993.

[21] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Communications of the ACM*, vol. 17, no. 12, 1974.

[22] A. Radulescu and A. J. C. van Gemund, "Fast and effective task scheduling in heterogeneous systems," in *Proceedings of the 9th Heterogeneous Computing Workshop (HCW)*, 2000.

[23] M. A. Iverson and F. Özgüner, "Dynamic, competitive scheduling of multiple DAGs in a distributed heterogeneous environment," in *Proceedings of the 1998 Seventh Heterogeneous Computing Workshop (HCW)*, 1998.

[24] H. Zhao and R. Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," in *Euro-Par*, 2003.

[25] V. Lo, J. Mache, and K. Windisch, "A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling," in *4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, USA*, ser. Lecture Notes in Computer Science (LNCS), vol. 1459. Springer, 1998.