Task Scheduling Strategies for Workflow-based Applications in Grids

Jim Blythe, Sonal Jain, Ewa Deelman,
Yolanda Gil, Karan Vahi
USC Information Sciences Institute
{blythe, sonjain, deelman, gil, vahi}@isi.edu

Anirban Mandal, Ken Kennedy.

Rice University Dept. of Computer Science
{anirban, ken}@cs.rice.edu

Abstract

Grid applications require allocating a large number of heterogeneous tasks to distributed resources. A good allocation is critical for efficient execution. However, many existing grid toolkits use matchmaking strategies that do not consider overall efficiency for the set of tasks to be run. We identify two families of resource allocation algorithms: task-based algorithms, that greedily allocate tasks to resources, and workflow-based algorithms, that search for an efficient allocation for the entire workflow. We compare the behavior of workflow-based algorithms and task-based algorithms, using simulations of workflows drawn from a real application and with varying ratios of computation cost to data transfer cost. We observe that workflow-based approaches have a potential to work better for data-intensive applications even when estimates about future tasks are inaccurate.

1. Introduction

Scientific communities ranging from high-energy physics [3], gravitational-wave physics [4], geophysics [5], astronomy [6], to bioinformatics [7] are embracing grid computing to manage and process large data sets, execute scientific simulations and share both data and computing resources. These scientific, data-intensive applications are no longer being developed as monolithic codes. Instead, standalone application components are combined to process the data in various ways. The applications can now be viewed as complex workflows that consist of various transformations performed on the data. For example, in astronomy, workflows with thousands of tasks are needed to identify galaxy clusters within the Sloan Digital Sky Survey [6]. Because of the large amounts of computation and data involved, these workflows require the power of the grid to execute.

In earlier work [8] we described techniques to generate a workflow based on a desired data product and the requirements of the available components, in terms of both input-output data and resource constraints. Here, we focus on allocation of resources to a workflow whose component tasks are known but not yet allocated. This is an important topic in Grid computing because of its high impact on the efficiency of the workflows, which may generate large amounts of data and occupy valuable resources for days. For example, our experiments below show alternative allocations by competitive algorithms whose runtimes differ by 22 hours with a maximum of 3 days.

We investigate whether, and under what conditions, an allocation algorithm should be influenced by the workflow structure. We evaluate resource allocation algorithms that use two distinct approaches. The first greedily allocates each ready-to-run task to a host based on information only about that task. We refer to this as the *task-based* approach. The second approach searches for an efficient allocation for the whole workflow, and may revise the allocation of a task based on subsequent tasks. We refer to this as the *workflow-based* approach. Apart from these differences, the two algorithms are based on the similar heuristics. We note that in real deployments one would want to find an overall allocation, but would only release portions of the workflow that are ready to run. If the underlying execution environment subsequently changes, the allocation may be redone.

We test the algorithms empirically using workflows that are generated in the context of an astronomy application [9]. We have developed a grid simulator for our experimentation, which allows us to easily vary parameters such as task compute times and file transfer times and also to introduce errors in their estimation to simulate dynamic environments. We test the different approaches in both compute- and data-intensive scenarios.

Many existing resource allocation strategies for Grid applications, *e.g.* [21], concentrate on matchmaking individual tasks to resources and do not attempt to find an efficient overall allocation. Because these algorithms do not examine tasks that come later on in the workflow, their allocation of resources may result in poor overall assignments if they create excessive data movement, particularly in data-intensive applications. Indeed, in our experiments, the workflow-based approach performed similarly to the task-based approach for compute-intensive cases but found more efficient allocations in data-intensive applications. Since the workflow-based approach depends on predictions of future task performance, we performed tests with inaccurate estimates of component runtimes and data transfer times, and found that the workflow-based approach still performs well under these conditions. However, the workflow-based approach is more computationally intensive and our implementation is not scalable for workflows with more than around ten thousand tasks.

In both approaches we initially used the min-min heuristic to identify promising allocations [12]. However, the resulting workflows tended to leave a significant number of resources idle during workflow execution. We defined a new heuristic, weighted min-min, that blends min-min with minimizing the idle time of resources, and show that this heuristic outperforms min-min in our test domain with both allocation approaches.

In the next section we present a formalization of the workflow allocation problem for Grids and describe the two families of algorithms we tested. In section 3 we explain the details of our simulator and the experimental setup used to compare the two approaches, discussing the results in section 4. We describe the weighted min-min heuristic in section 5 and investigate the impact of uncertainty in section 6. Section 7 discusses related work. In the final section we review lessons learnt and future work.

2. Resource allocation strategies

We refer to a directed acyclic graph (DAG) of tasks that provides information only about task and file dependencies as an ***abstract workflow***, and to a DAG that also provides information about the resource to which each task is allocated as a ***concrete workflow***. Figure 1 shows an abstract and corresponding concrete workflow with 4 task nodes, J1 – J4. F1 – F5 are files that are transferred, while R1 and R2 are resources. We use the terms 'task' and 'job' interchangeably.

Consider a DAG of jobs $J = \{j_1, j_2, \ldots j_m\}$ and a set of available Grid resources $R=\{r_1, r_2, \ldots r_n\}$. A job j_i has an estimated run time $t(j_i)$ and each resource has an intrinsic speed $s(r_j)$, so that the estimated running time of the job on the resource is $t(j_i)/ s(r_j)$. Resources can be connected to each other and a link between any two resources i, j is assumed to have a bandwidth $b(i, j)$. Each job has a set of associated input and output files. Each file f used or produced by a job has a size $l(f)$, so we assume that transferring the file between resources takes time $l(f)/b(i, j)$. Each resource r_j executes jobs in sequence drawn from a single queue, q_j. When a job reaches the front of the queue it is executed immediately if its input files are available at the resource. If not, the resource remains idle until the files are available.

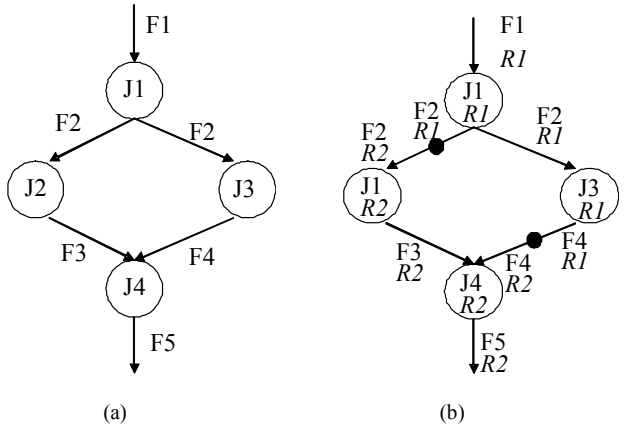

Figure 1. Example Workflows (a) An Abstract Workflow (b) Corresponding Concrete Workflow

A scheduling algorithm seeks a mapping from the jobs in the abstract workflow to the resource queues, $S: J \rightarrow R \times N$, where $S(j_i) = (r_j,x)$ means that job j_i occupies the x^{th} position on the queue of resource r_j. We seek a schedule S that minimizes the ***makespan***, defined as the time from when execution starts until the last job in the workflow is completed. Finding such a schedule is NP-hard, by a reduction from Minimum Multiprocessor Scheduling [10]. One approach for solving the mapping problem optimally is to model it as an Integer Linear Programming problem. However, our mapping creates too many variables to be solved in reasonable time by current methods, so we turn to heuristic solutions to the problem.

We classify resource allocation algorithms into two broad categories based on whether they take into account future tasks in the workflow when allocating a task. We refer to these categories as *task-based* (**TBA**) and *workflow-based* (**WBA**) approaches.

2.1. Task-based approach

Algorithms that only reason about the tasks or jobs that are ready to run at any given instant are classified as *task-based allocation* algorithms. These algorithms make local decisions about which job to send to which resource.

For example, consider the abstract workflow in Figure 1a, where file F1 is available initially and so only job J1 is available for scheduling. Once J1 finishes, jobs J2 to J3 form the set of available jobs. These jobs are selected for scheduling one at a time using a local selection heuristic. We apply the widely-used min-min heuristic from the domain of scheduling parameter sweep applications [11] to study task-based approaches. Min-min runs in polynomial time but produces efficient schedules and has been evaluated for many different situations involving independent/non-communicating tasks in [12]. However, it does not guarantee an optimal mapping.

We define some terms that will be used to describe the algorithms. We assume that the estimation of execution time of a particular job on a particular resource is accurate (in Section 6 we investigate uncertain execution time estimates). These time estimates can be obtained using different kinds of performance modeling techniques, *e.g.* analytical or historical. For every (job, resource) pair we define the following quantities:

- The ***Estimated Execution Time EET(j,r)*** is defined as the time the resource *r* will take to execute the job *j* from the time the job starts executing on the resource.
- The ***Estimated Availability Time EAT(j,r)*** is defined for every resource as the time at which the resource *r* will become free to perform job j (*i.e.* the time at which it will have finished executing all the jobs before j in its queue).
- The ***File Availability Time FAT(j,r)*** is defined as the earliest time by which all the files required by the job *j* are available at the resource *r*.
- The ***Estimated Completion Time ECT(j,r)*** is the time at which job *j* would complete execution at resource *r:*

  $ECT(j, r) = EET(j, r) + max(EAT(j,r), FAT(j, r))$

We now describe the min-min local selection heuristic. For each available job, the resource with the minimum *ECT* value is found. Denote this as a tuple *(j, r, t),* where *j* is the job, *r* is the resource for which the minimum is achieved and *t* is the corresponding *ECT* value. Next, the minimum *ECT* value over all available jobs is found. A job with the minimum *ECT* value is scheduled next. This is repeated until all the jobs have been scheduled. The intuition behind this heuristic is that the makespan increases the least at each iterative step, hopefully resulting in a small makespan for the whole workflow. Figure 2 summarizes the algorithm.

```
1)  while all jobs are not finished do
2)  Find availJobs = jobs with every parent finished;
3)  Schedule(availJobs);
procedure Schedule(availJobs)
4) while all availJobs not scheduled do
5)  foreach job,j  do
6)    foreach  resource,R  do
7)          Calc ECT(j, r);
8)            Find min ECT(j, r) over all R;
9)  Find min(min ECT(j, r) ) over  all J;
10)   Schedule j
11)  Update EAT(r)
```

Figure 2: Min-min task scheduling algorithm.

2.2 Workflow-based Approach

Algorithms that reason about the whole workflow rather than the set of available jobs are classified as *workflow-based allocation* algorithms (WBAs). In this approach all the jobs in the workflow are mapped a priori to resources in order to minimize the makespan of the whole workflow. As mentioned in the introduction mapping the entire workflow does not imply scheduling all the jobs ahead of time. In fact if changes in the environment occur, remapping may be necessary. Mapping the entire workflow avoids potential myopia in the scheduler, as is the case with task-based approaches, which only consider available jobs. In this section we present a local search algorithm for workflow allocation based on generalized GRASP procedure (Greedy randomized adaptive search) [13], which has been shown to be effective for job-shop scheduling [14].

In this approach a number of iterations are made to find the best possible mapping of jobs to resources for a given workflow. The main difference is that WBA creates and compares many alternative whole workflow schedules before the final schedule is chosen, while TBA compares partial schedules among the available tasks as the workflow is executed.

On each iteration, an initial allocation is constructed in a greedy phase. In principle a number of local modifications may be considered by swapping pairs of tasks, but this is not implemented in the current system. The initial allocation algorithm computes the tasks whose parents have already been scheduled on each pass, and considers every possible resource for each such task. For each (task, resource) pair, the algorithm computes the increase to the current

makespan of the workflow if the task is allocated to this resource. Let *I-min* be the lowest increase found and *I-max* be the largest. The algorithm picks one pair at random from those whose increase *I* is less than *I-min* + α (*I-max* – *I-min*) for some width parameter α, 0 ≤ α ≤ 1, and continues until all tasks are allocated. The width parameter α determines how much variation is allowed each time a candidate workflow allocation is constructed. When α = 0, each iteration of the algorithm behaves like the task-based min-min solution. When α = 1, each iteration is random. In some domains, a non-zero α is essential to find globally optimal allocations, while in others the variation due to several component allocations having equally good heuristic scores is enough to find the optimal. The algorithm for our workflow-based approach is shown in figure 3, omitting the subroutine to swap task allocations.

```
1)  repeat until time limit is reached:
2)   concreteWF = CreateMapping(workflow);
3)   if concreteWF has lower makespan than bestConcreteWF
4)     bestConcreteWF = concreteWF

procedure CreateMapping(workflow)
5)  while all jobs in workflow are not mapped do
6)     Find availJobs = unmapped jobs with every parent mapped;
7)     Map(availJobs);
// 8)   SwapAllocationsofPairsofTasks()

procedure Map(availJobs)
9)  while all availJobs not mapped do
10)   foreach job,j  do
11)     foreach resource,R  do
12)          calc ECT(j, r);
13)   I-min  =  min makespan increase over all j and r;
14)   I-max  =  max makespan increase over all j and r;
15)  availPairs = all pairs (j', R')
16)     s.t makespan increase <= I-min  + α *(I-max – I-min);
17)   (j*, R*) = random choice from availPairs;
18)   map(j*, R*)
19)   Update EAT(j*,R*)
```

Figure 3: Workflow-based algorithm

3. Experimental Setup

We investigated the performance of the two approaches using a grid simulator built over the popular network simulator NS-2 [17]. We use NS to estimate the average bandwidths between the resources and as a discrete event simulator. We briefly describe the simulator and our experimental setup.

The grid simulator models resources, bandwidths of links connecting different resources, jobs and files being transferred as separate objects. This provides a fairly simple testbed for comparing the performances of scheduling algorithms for grids, in which the underlying grid infrastructure can be changed very easily. For example, we can easily change the number of resources, their individual computational power, the bandwidths of the links connecting them, etc. Resources are modeled as *sites* containing one or more *hosts*, where jobs are executed, and *storage* objects, where related input/output data, program files are stored. A *site* acts as an entry point for jobs to be executed and also manages the fetching of files from *storage* units. After a job is received, the site submits the job on a compute resource within its pool that has minimum job queue length. The site is created on top of NS's node as an application agent capable of transferring packets between them. The *hosts* are modeled with *first-come-first-serve* queues.

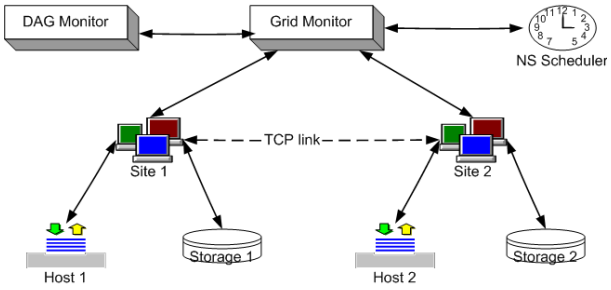

Figure 4: Architecture of Grid Simulator

The simulator can handle more than one workflow simultaneously, thereby emulating a Grids' capacity of handling jobs from more than one client. The *DAG Monitor* module manages information specific to different workflows. It receives a workflow in the form of a DAG. Depending upon the preferences set different scheduling strategies are used to map jobs to the sites.

The *Grid Monitor (GM)* is the central module of the simulator, coordinating the activities of the other modules. It receives jobs from the DAG Monitor and passes them to the sites/resources where the job is to be executed. It interacts with the *NS scheduler* for setting events such as transfers, job start and finish times, etc. and handles the events when they are due. The GM keeps track of all the sites created and the files they contain and the jobs that are scheduled on them.

We used a simple network of 6 fully connected sites for all experiments in this paper, with each site having a single host and storage unit. The hosts can have the same or different computational speeds resulting in a homogenous or heterogeneous case respectively. A set of initially available files at each site is specified for each workflow. It is assumed by the schedulers that the initial files are available from at least one site. For simplicity, we do not report results

with packet level file transfers in NS, which take large amount of time for the size of workflows we deal with. Thus, in our model, we assume that any number of files can be transferred in parallel without affecting the bandwidths of the connecting links and computational powers of corresponding sites. We estimated bandwidths by simulating a large amount of data transfers on the topology used on NS.

4. Experimental comparison of the approaches

We conducted several experiments in order to compare the two scheduling approaches using workflows drawn from the Montage astronomy application [9]. These workflows have a basic structure shown in Figure 5, but are varied in the number of jobs at each horizontal level. The job compute times at a level were drawn from a distribution for given mean and a variance of 10%. The mean was varied for each level. All files produced by jobs at same level were modeled to have same size. We conducted all the experiments on the grid simulator and with the host topology described in the previous section.

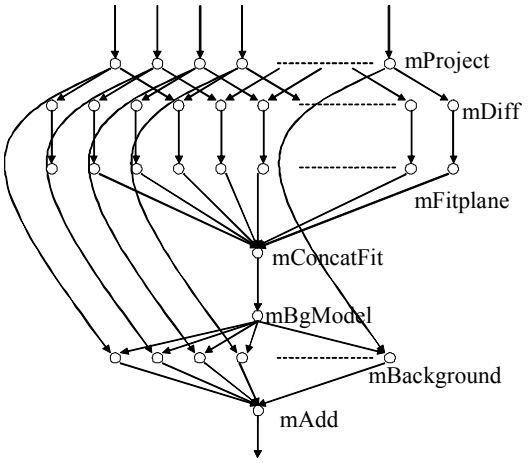

Figure 5: Montage workflow schema

In the simulator, we varied the relative time to perform computational jobs versus file transfers to explore different conditions for scheduling. For the same workflows we multiply job compute times with a factor called compute factor (CF) and file sizes with data multiplying factor (DF) in order to obtain different workflows. We refer to workflows whose file transfer times dominate job compute times as ***data-intensive*** (where CF is relatively small compared with DF). Workflows whose job compute times dominate file transfer times are termed as ***compute-intensive*** (where CF is relatively large compared with DF). We used two different resource scenarios, one in which all resources have the same computational power (*homogeneous*) and one with different computational powers (*heterogeneous*). We ran the workflow-based algorithm with a time limit of 200 seconds. Although in WBA α can be varied to find best possible solution in the specified time limit, in our experiments we fixed the value of α to 0.005 as we observed this yielded the best solution in most cases.

Table 1 compares the makespan of the allocation found using the task-based approach with that found using the workflow-based approach and a random allocation for a workflow with 1185 job workflow for (a) homogeneous and (b) heterogeneous resource scenarios. The upper three rows in the tables, with constant compute factor (CF) 0.1, represent *data-intensive* cases and the lower three rows with constant data multiplying factor (DF) of 100 represent *compute-intensive* cases.

As can be seen from table 1, both the workflow and task-based approaches outperform a random allocation algorithm. WBA produces schedules with significantly lower makespan for the data intensive cases compared to TBA. There is no significant difference for compute intensive cases. We also compared the performance of these algorithms for smaller workflows from the Montage domain with 57 and 739 jobs with similar results.

Experiment Description		Random	Task Based Allocation	Workflow Based Allocation	WBA/ Random	TBA/ WBA
CF: 0.1 DF: 100	*(a)*	12834	3100	2112	6.08	1.47
	(b)	45504	7872	3863	11.78	2.04
CF: 0.1 DF: 1000	*(a)*	97242	23413	15921	6.11	1.47
	(b)	101952	32384	21267	4.79	1.52
CF: 0.1 DF: 10000	*(a)*	942814	231569	154634	6.10	1.50
	(b)	920314	239038	159825	5.76	1.50
CF: 1.0 DF: 100	*(a)*	18098	15354	14030	1.29	1.09
	(b)	440070	20260	20070	21.93	1.01
CF: 10.0 DF: 100	*(a)*	153307	134713	133959	1.14	1.01
	(b)	4395721	182076	180038	24.42	1.01
CF: 100.0 DF: 100	*(a)*	1496738	1337900	1337619	1.12	1.00
	(b)	44022665	1791580	1786649	24.64	1.00

Table 1: Average makespan in seconds by strategy for 1185-job workflows.

One of the most important reasons for the difference in performance of the two algorithms in this domain is the ability to *pre-position* the data using WBA. Transfer of a large file can begin immediately after it is created because the destination is known before the file is created. In TBA, transfer cannot begin until the consuming job is scheduled, which is only done after the last parent job has been scheduled. In the Montage workflows, the files transferred

between the mProject and mBackground jobs are typically large, and can be transferred simultaneously with the execution of the mDiff and mFitplane jobs in WBA.

We would also expect WBA to be more effective in general data-intensive scenarios, when costly transfers of large files might be avoided by producing them where they are to be used, or at locations with a high bandwidth connection. On the other hand TBA, making purely local decisions about each job, tends to distribute jobs evenly among available resources. In data-intensive scenarios, this increases the chance that a large file will need to be transferred. Allocating several jobs on the same resource whose outputs are later combined often leads to a more efficient workflow even though the computation time for the jobs will be longer.

WBA and TBA both perform much better than random allocations in compute-intensive cases with heterogeneous resources, but the difference is far less marked with homogeneous resources. This is because the general structure of a good solution is different in the two cases. When one or two resources are faster than others, a good schedule must allocate key jobs to those machines, and this has a low probability of occurring in a random allocation. When resources are homogeneous, a good allocation must spread jobs evenly between them, and a random allocation is likely to have this characteristic.

TBA's algorithm, and the Map() subroutine of WBA have quadratic time complexity in the size of the workflow. WBA took an average of 0.95 seconds per iteration for 1185-job workflows, 3.5 seconds for 2047 jobs and 8 seconds for 3029 jobs. Using weighted min-min, WBA typically finds a solution in a few tens of iterations that is close to the best solution it finds in hundreds of iterations, however it is unlikely to be effective in its current form for workflows with tens of thousands of jobs.

5. A New Local Selection Heuristic

While studying the performance of the two approaches, we observed that resources are often left idle, waiting for the input files for the job to be executed to arrive. Since the local decision-making policy is to minimize the estimated completion time, we observed that this could lead to higher idle times. We therefore developed a new heuristic to optimize the resource utilization which explicitly reasons about the *idle time* of the resources, called the weighted min-min heuristic.

Idle time for a resource is the time when one or more jobs wait in its queue for their files to be transferred and no job can be executed. For large file sizes and low bandwidth scenarios, this idle time causes poor resource utilization. For example consider figure 6. For job *j* to be mapped on to resource *R1* the required file F1 must be transferred from R2 to R1. If F1 requires a considerable amount of time to transfer (as in data intensive cases), the resulting idle time IT(j,R1) for job j on resource R1 is significant. Figure 7 shows an example where a job j1 is mapped on resource R1 since it has a lower estimated completion time than another job j2, but j1 leads to a larger idle

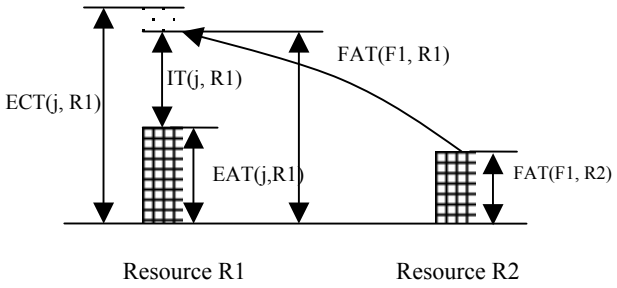

Figure 6: Explanation of Idle Time

time on resource R1. We observed such cases often in our experiments.

To avoid this resource underutilization we introduce a new local selection heuristic for job scheduling. We first formalize the definition of idle time as follows. Idle Time *IT(r)* for a resource *r* is the total time spent by all the jobs scheduled at *r* waiting for their respective files to arrive. Idle Time *IT(j,r)* for a job *j* and resource *r* is the estimated total idle times for all the resources if *j* is scheduled at *r*, which is the sum of idle times of all the resources plus the idle time caused if job *j* is executed on resource *r.*

$$IT(j, r) = IT(r) + \max(0, (FAT(j, r) - EAT(j, r)))$$

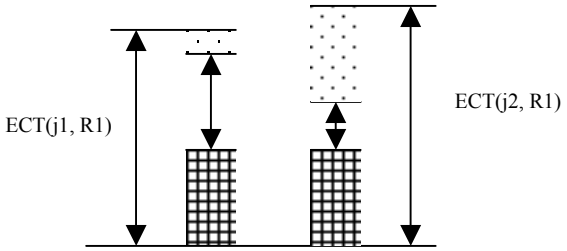

Figure 7: Significance of Idle Time

Weighted Min-Min Heuristic: For each available job *j*, and feasible resource *r*, the weighted sum *(WT)* of *ECT* and *IT* is computed as follows:

$$WT(j, r) = \gamma\ IT(j, r) + (1-\gamma)ECT(j, r),$$

For some γ with $0 \leq \gamma \leq 1$. We find a job and resource pair with minimum weighted sum, and such a job is scheduled next. This is repeated until all the jobs have been mapped.

We use the same workflows and underlying grid topology on the simulator as for the earlier experiments to test the effect of the new heuristic. Table 2 compares both task and workflow approaches for different local selection heuristics for 1185 job workflows. We compare the min-min heuristic with the weighted min-min heuristic. We observe that weighted min-min heuristic results in an average speed up factor of 1.1 for the task-based approach and 1.27 for the workflow-based approach. Reasoning about idle time makes a significant difference for data intensive cases. For compute intensive cases, the two heuristics perform nearly the same for both approaches. This is not surprising as the idle time plays a significant role only when file transfer times are comparable with or larger than the job compute times. The weight (γ) can be changed over different iterations in WBA, allowing it to find the best weight value for a particular workflow. For TBA, we observed that γ =0.5 gave best solution in most of the experiments and used this weight for all the results reported here. In addition, WBA typically converges in around 5 iterations in the Montage domain using weighted min-min – far fewer than with regular min-min. In data-intensive cases, the weighted workflow-based approach leads to an average speed-up factor of more than 2 over the weighted task-based approach.

Experiment Description		TBA Min-min (i)	TBA wtd min-min (ii)	Speed up (i)/(ii)	WBA Min-min (iii)	WBA wtd min-min (iv)	Speed up (iii)/(iv)	Wtd Speed up (ii)/(iv)
CF: 0.1 DF: 100	*a*	3100	3076	1.01	2112	2113	1.00	1.46
	b	7872	9297	0.85	3863	3129	1.23	2.97
CF: 0.1 DF: 1000	*a*	23413	15890	1.47	15921	8798	1.81	1.81
	b	32384	32271	1.00	21267	15113	1.41	2.14
CF: 0.1 DF: 10000	*a*	231569	154575	1.50	154634	79031	1.96	1.96
	b	239038	172900	1.38	159825	89210	1.79	1.94
CF: 1.0 DF: 100	*a*	15354	15152	1.01	14030	14037	1.00	1.08
	b	20260	20723	0.98	20070	20106	1.00	1.03
CF: 10.0 DF: 100	*a*	134713	134664	1.00	133959	134034	1.00	1.00
	b	182076	182041	1.00	180038	179928	1.00	1.01
CF: 100.0 DF: 100	*a*	1337900	1340170	1.00	1337619	1338148	1.00	1.00
	b	1791580	1792240	1.00	1786649	1783579	1.00	1.00

Table 2: Average makespan in seconds for different heuristics for 1185-job workflows.

6. The Impact of Inaccurate Execution Estimates

In practice, grid environments are highly dynamic. A job may have a very different runtime on a resource than the estimated time used for mapping. File transfer times may also vary widely. Workflow-based algorithms may be more vulnerable to inaccurate estimates of runtimes because they rely on forecasts for the performance of future tasks, while task-based approaches only reason about the next tasks to be run.

We therefore compared the performances of the two approaches under both compute-time and transfer-time uncertainty. We simulate uncertain job compute time by creating a distribution for a given percentage of uncertainty and the expected job compute time for all the jobs in the workflow. Algorithms for both the approaches use the estimated job compute time but the actual compute times are randomly picked from the distribution. To model file transfer time inaccuracies we use the expected and actual file sizes for all the files. Again the two approaches rely on the estimated file sizes but simulator picks up the actual file size randomly from the distributions generated for all files in the workflow.

Figure 8 presents performance of both the approaches using the weighted min-min heuristic in a case of uncertain compute times for a compute intensive case (CF 100.0 and DF: 100). We found that the effect of job compute time uncertainties is most pronounced in the compute intensive case (and for the same reason we chose a data intensive case for transfer time uncertainties). We vary the uncertainty level from 0 to 400 %. 400% means that the simulator picks the actual compute time randomly from the range: [1, (*rt* + 8 * *rt*)] where *rt* is the estimated runtime. The range has width equal to twice the amount of possible divergence and is shifted so as to always be positive. Our results are averaged over 25 random trials. The performance of TBA is more affected by uncertainty in compute times than WBA, although both the approaches perform poorly as the uncertainty increases.

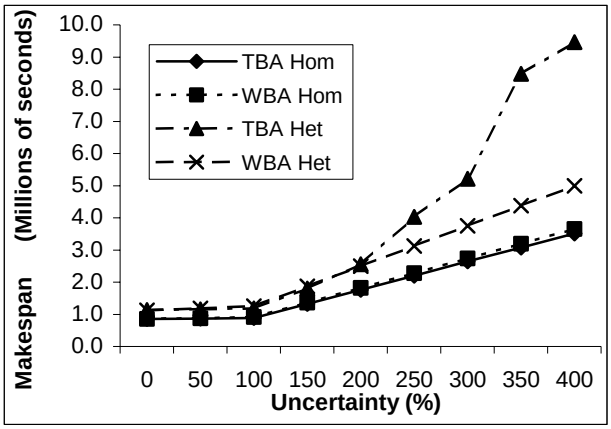

Figure 8: Average makespan in seconds for uncertain compute times, 739 job workflows.

Figure 9 compares TBA and WBA for uncertain transfer times for a data intensive case (CF: 0.1 DF:

10000) for a 739-job workflow. Here the difference between the performances of TBA and WBA was more pronounced. Performance of TBA degrades rapidly with increasing uncertainty in comparison to WBA. We conjecture that higher levels of uncertainty, in our model, increase the mean transfer time, making the problems more data-intensive.

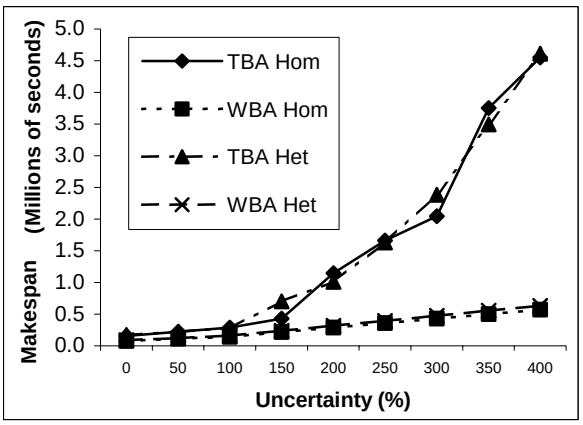

Figure 9: Average makespan in seconds for uncertain transfer times, 739 job workflows.

7. Related work

Scheduling application components onto multiprocessors is a hard problem that has been studied extensively in the literature. In most cases the problem is NP-complete, since the minimum multiprocessor scheduling problem is NP-complete [10]. Therefore, most of the literature deals with finding good heuristic solutions. There is a body of work on multiprocessor scheduling of independent application components. [12] gives an overview of different heuristics including min-min, max-min and sufferage. [11] extends some of these heuristics and presents a new heuristic to consider data movement, which is important in the Grid context. Simple heuristics for dynamic matching and scheduling of independent jobs onto heterogeneous resources are presented in [1, 2]. [18] applies a modified min-min heuristic to schedule independent components on the Grid.

Our work deals with scheduling tasks whose dependencies with each other form a DAG, so these heuristics can't be directly applied. However, we use min-min during the overall scheduling process. [19] gives a survey on different heuristic scheduling techniques for scheduling application DAGs onto homogeneous platforms. These heuristics also can't be applied directly to grids because they are highly heterogeneous. [20] considers DAG scheduling for heterogeneous platforms. Most of the heuristics are generalizations of the list-scheduling based heuristics for homogeneous platforms, which have several drawbacks in grids. First, they do not consider the global effect of the current scheduling decision. Second, they do not group tasks for scheduling and third, the dramatic heterogeneity of grids makes the average values they use for edge and node weights questionable.

Our work has some similarities with the Levelized Min Time heuristic [20] in the sense that we also consider the nodes level by level. However we use more sophisticated heuristics at each level and randomize decisions to reduce the probability of being caught in local minima. The work in [16] on a hybrid scheduling heuristic for DAG scheduling onto heterogeneous systems is most closely related to ours. Unlike their work, we consider resource speeds and bandwidths, to reflect the heterogeneity of the Grid. Current Grid workflow management systems use simple approaches to scheduling such as first-come-first-served with matchmaking [21], or random allocations or round-robin [8].

The GridLab group [22] describe genetic algorithms, simulated annealing and tabu search for workflow scheduling. However we are not aware of an experimental evaluation of their techniques. These techniques have also been applied to job-shop scheduling problems, and tested under uncertainty [23]. The GRASP algorithm, which forms the basis of our workflow-based algorithm, was also used for job-shop scheduling by Binato et al. [14].

8. Conclusions and Future work

We distinguished the task-based and workflow-based approaches to resource allocation for tasks in workflows, and compared them via simulation on a class of workflows based on an astronomy application. The performance of the two approaches is similar for compute-intensive cases, but the workflow-based approaches perform better for data-intensive cases, where they take advantage of the ability to begin transferring large data sets earlier and make decisions based on global measures of performance.

However, the time taken by the workflow-based algorithms grows more rapidly than for the task-based approaches, making them less appropriate for workflows containing several thousand tasks. Although the class of workflows tested is too small to draw conclusions for a wide range of workflows, they are realistic and representative of problems encountered on the grid. Our results indicate that task-based approaches may be the better suited to compute-

intensive domains by virtue of their speed, but workflow-based approaches appear better suited to data-intensive domains, when they are practical.

We also demonstrate a new heuristic that improves the resource utilization by taking the resource idle time into account. The heuristic converges to a good solution quickly for our tested workflows, making workflow-based approaches suitable for larger workflows. We are currently investigating ways to scale the workflow-based approaches to handle even larger workflows, based on both modifications to the algorithm and aggregating groups of tasks in the original workflow to create a small sub-problem. We are also studying the performance of the two approaches and the new heuristic on a wider range of workflow schemas.

Acknowledgments

We are grateful for conversations with Carl Kesselman and Rizos Sakellariou. Jaskaran Singh initially implemented the simulator.

This work was supported by NSF under grant ITR AST0122449 (NVO), grant No. ACI 0103759 (GrADS), Cooperative Agreement No. CCR-0331654 (VGrADS) and by an internal grant from the Information Sciences Institute. Montage is supported by the NASA Earth Sciences Technology Office Computing Technologies program, (ESTO-CT) under Cooperative Agreement Notice NCC 5-6261.

References

[1] Braun T D, *et al*, A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. *IEEE Workshop on Advances in Parallel and Distributed Systems,* West Lafayette, IN, Oct. 1998, pp. 330-335.

[2] Maheswaran M, *et al*, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. In the *8th IEEE Heterogeneous Computing Workshop (HCW '99),* San Juan, Puerto Rico, Apr. 1999, pp.30-44.

[3] GriPhyN, www.griphyn.org

[4] E. Deelman, *et al*., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," *Intl Symp on High Performance Distributed Computing*, 2002.

[5] SCEC CM Environment, www.scec.org/cme/.

[6] J. Annis, Y. Zhao, *et al*., "Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey," *Technical Report GriPhyN*-2002-05, 2002.

[7] NPACI, Telescience, https://gridport.npaci.edu/Telescience/.

[8] Deelman, E. Blythe, J. *et al.*, Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, vol.1, 2003 pp. 25-39.

[9] Berriman el al. Montage: a Grid Enabled Image Mosaic Service for the National Virtual Observatory, ADASS 13, 2003

[10] Garey, M. and Johnson, D., Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, New York, 1979

[11] Casanova,. H. et al., Heuristics for scheduling parameter sweep applications in Grid environments. In *Proc. 9th Heterogeneous computing Workshop (HCW'2000),* Cancun, Mexico, 2000, pp.349-363.

[12] Tracy D, *et al*, A Comparison of eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *J. Parallel and Distributed Computing,* 61, pp.810-837, 2001

[13] Resende, M. and Ribeiro, C., Greedy Randomized Adaptive Search Procedures, State-of-the-art Handbook in Metaheuristics, Glover and Kochenberger, eds., Kluwer Academic Publishers, 2002

[14] Binato, S. *et al*., A GRASP for job shop scheduling. In Ribeiro and Hansen, eds, *Essays and surveys on metaheuristics*, pp 59-79. Kluwer Acad. Publishers, 2001.

[15] Freund R et al, Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *Proc. 7th IEEE Heterogeneous Computing Workshop (HCW '98),* Orlando, FL, USA, Mar. 1998, pp.184-199.

[16] Sakellariou. R and Zhao, H., A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. In *Proc 13th Heterogeneous Computing Workshop (HCW '04), Santa Fe, New Mexico, USA,* 2004.

[17] Network Simulator, http://www.isi.edu/nsnam/ns

[18] He, X., Sun, X, and von Laszewski, G., QoS guided min-min heuristic for grid task scheduling. *Journal of Computer Science and Technology*, 18, 2003, pp.442-451.

[19] Yu-Kwong Kwok and Ishfaq Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors. In *ACM Computing Surveys*, vol. 31,1999 pp. 406-471.

[20] Topcuoglu, H., Hariri, S. and Wu, M., Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems,* 13, 2002, pp. 260-274.

[21] DAGMan meta-scheduler for Condor http://www.cs.wisc.edu/condor/dagman

[22] Mika, M. *et al*. *A Metaheuristic Approach to Scheduling Workflow Jobs on a Grid*, in Grid Resource Management: State of the Art and Future Trends, Kluwer Academic Publishers, 2003

[23] Beck, C. and Wilson, N. Job shop scheduling with probabilistic durations, *European Conference on Artificial Intelligence (ECAI '04)*, 2004

[24] Cooper, K. *et al*. New Grid Scheduling and Rescheduling Methods in the GrADS Project, *18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 10 (NSF NGS Workshop)*, 2004