Scheduling Strategies for Mapping Application Workflows onto the Grid

Anirban Mandal, Ken Kennedy, Charles Koelbel,
Gabriel Marin, John Mellor-Crummey
Department of Computer Science, Rice University,
Houston, TX 77005, USA
{anirban, ken, chk, mgabi, johnmc}@cs.rice.edu

Bo Liu, Lennart Johnsson
Department of Computer Science,
University of Houston,
Houston, TX 77204, USA
{boliu,johnsson}@cs.uh.edu

Abstract

In this work, we describe new strategies for scheduling and executing Workflow applications on Grid resources using the GrADS [18] infrastructure. Workflow scheduling is based on heuristic scheduling strategies that use application component performance models. The Workflow is executed using a novel strategy to bind and launch the application onto heterogeneous resources. We apply these strategies in the context of executing EMAN, a Bio-imaging workflow application, on the Grid. The results of our experiments show that our strategy of performance model based, in-advance heuristic workflow scheduling results in 1.5 to 2.2 times better makespan than other existing scheduling strategies. This strategy also achieves optimal load balance across the different Grid sites for this application.

1. Introduction

For the past several years, the *Grid Application Development Software (GrADS) Project* has been developing tools to make the construction of applications for the Grid easier, while maintaining a high standard of overall performance. This work has led to the development of a prototype software infrastructure called GrADSoft that runs on top of Globus and facilitates the scheduling, launching, and performance monitoring of tightly coupled Grid applications, particularly those that require MPI communications. At the heart of the GrADS infrastructure is the notion that each Grid program be represented as a *configurable object program* consisting of the application code plus a *mapper*, which would determine how to assign tasks in the application to different components of a collection of grid resources, and a *resource selection performance model*, which could be used to estimate the overall performance of the application when run on a collection of grid resources using a task assignment produced by the mapper. The key to this strategy has been to provide tools for automatic and semi-automatic construction of mappers and performance models. The GrADS team has explored this approach by implementing and executing a number of prototype applications on GrADS software layer running on a special collection of geographically distributed clusters that form the GrADS Testbed.

Although this work has focused on tightly-coupled applications, it became clear to us that the strategy could also be used to improve the development and execution of so-called workflow applications in which the overall workflow is represented by a directed acyclic graph in which each vertex is an application that forms a component of the overall work to be done and each edge represents a data dependency between specific application components. Usually substantive data transfers takes place along edges in the workflow graph.

Currently, there are a number of tools supporting the execution of workflow applications, the most prominent being the Condor DAGMan [7] tool. However, most of these tools schedule the workflow dynamically at each step, so that the overall mapping and schedule is determined by the state of the Grid at the series of points when the application components are ready to execute. A much better outcome can be achieved in some cases by applying in-advance static scheduling to the process to ensure that the key computational steps are executed on the right resources and large-scale data movement is minimized.

In this paper, we present a strategy that extends the GrADS approach of using performance estimators to schedule workflow applications. In addition, we present results of our experiments with the extended GrADSoft infrastructure on a workflow application taken from the domain of medical image reconstruction.

The rest of the paper is structured as follows. Section 2 defines Workflow applications. In section 3, we describe the techniques used for component performance modeling and the heuristic workflow scheduling algorithm using these models. Section 4 describes the novel strategies to execute the workflows. In section 5, we present an experimental

evaluation of our scheduling techniques by comparing it with existing scheduling strategies on workflows from a real biomedical application. Section 6 presents related work. Section 7 concludes the paper and discusses some future work.

2. Workflow Applications

Many important Grid applications fall into the category of Workflow Applications, examples being LIGO [1] pulser search on the Grid, image processing on the Grid [15], Montage [11] etc. Instead of the application being a single large component doing all the tasks, the Workflow Application consists of a collection of several interacting components that need to be executed in a certain partial order for successful execution of the application as a whole. These components have specific control and data dependencies between them. In most cases, the Application Workflow can be represented as a directed acyclic graph (DAG) where each node in the DAG represents an application component and the edges denote control/data dependencies. In the most general case, some of the application components can be parallel components like MPI jobs. The main issues for executing Workflow Applications on the Grid are

- the ability for the components to seamlessly access the required Grid resources
- efficient selection of resources for the components in order to achieve good performance
- satisfying all dependencies and automating the Grid execution of the entire workflow

We intend to address these issues in the following sections. The next section describes our approach to scheduling Workflow application components.

3. Scheduling Workflow Applications

The design space for Grid Schedulers in general is very rich. First, it depends on what objective function the user wants to minimize or maximize – examples being minimizing overall job completion time, minimizing communication time and volume, and maximizing resource utilization or throughput. Second, it depends on how the job requirements, job performance models, and Grid resource models are specified and used. The scheduler must also carefully choose between different implementations of user authentication, allocation, and reservation. Other choices include scheduling application components for single or multiple users and whether rescheduling or re-planning is required. In our Workflow scheduling work, we have chosen the following design parameters. Our objective function is to minimize overall job completion time or the makespan of the application. The makespan of a Workflow application is defined as the time at which the last component in the workflow finishes execution. We develop Grid resource model through calls to different Grid services - Monitoring and Discovery Service (MDS) [12] and Network Weather Service (NWS) [31]. We derive application component performance models using methods described in a later section. The scheduling problem we are trying to solve can be stated as follows.

3.1 Problem Statement

Given a DAG of the Workflow representation of the application, let the set of available application components from the DAG be denoted by $C = \{c_1, c_2, ...c_m\}$ and the set of available Grid resources be $G = \{r_1, r_2, ...r_n\}$. The scheduling problem is to output a mapping from elements of C onto elements of G, or in other words, to output a mapping of which component runs on which Grid resource in what order.

3.2 Approach

We take a two-stage approach to solve the problem. In the first stage, for each component, we rank the resources and assign specific rank values to each resource on which the component can be mapped. Rank values reflect the expected performance of a particular component on a particular resource. We will explain how to assign these rank values in detail in the next section. As a result of this step, we will have associated rank values for a set of resources for a particular component. In the second stage, we take the rank values for each component and build up a Performance Matrix. We then use certain known heuristics to obtain a mapping of components to resources. These steps are explained in detail below.

3.2.1 Calculation of rank values

Calculating rank values involves matching a specific component to a set of resources and then ranking the resources. For this purpose, we assign rank values for each possible mapping of the component to an available resource. According to our ranking convention, resources with a lower rank are a better match for the component. The rank value for each resource is assigned in the following manner.

- At first it is checked whether the Grid resource meets certain hard requirements [like required OS, required memory, storage, required minimum CPU speed etc.] for the component. If the resource doesn't meet the hard requirements, it is given a rank value of infinity. The resource properties are derived from calls to MDS services.

- Next, the rank values for the eligible resources are evaluated. Rank value is a weighted linear combination of expected execution time on the resource, r_j for the component, c_i denoted by $eCost(c_i, r_j)$ and expected cost of data movement denoted by $dCost(c_i, r_j)$.

  $$rank(c_i, r_j) = w_1 \times eCost(c_i, r_j) + w_2 \times dCost(c_i, r_j) \quad (1)$$

  The weights can be customized to give more importance to one over the other. $eCost(c_i, r_j)$ is derived from Application component performance model and is described in the next section. $dCost(c_i, r_j)$ is calculated as follows. Let $map(c_i)$ denote the resource on which c_i has been mapped to, $vol(c_i)$ denote the volume of data produced by c_i, $Parent(c_i)$ denote the set of parent components for c_i and $Tx(r_p, r_q)$ denote the unit data transfer time from resource r_p to resource r_q. Then $dCost(c_i, r_j)$ is defined as

  $$dCost(c_i, r_j) = \sum_{p \in Parent(c_i)} vol(p) \times Tx(map(p), r_j) \quad (2)$$

  $Tx(r_p, r_q)$ is estimated from latency/bandwidth information from the NWS. Note that when the rank for the current set of available components is being calculated, the mapping for the parents of the current components will be already known. For a single component, the resource having the lowest rank may be returned as the best match.

$eCost(c_i, r_j)$ values are calculated using application component performance modeling as follows.

3.2.2 Component Performance Modeling

This section describes how to estimate the execution time of an application component on a Grid resource using application component performance modeling. The Workflow scheduler uses performance models to determine the runtime resources needed by an application, and to compute a mapping for different components that minimizes the application makespan. To estimate the execution cost of an application on arbitrary grid configurations, we analyze an application's behavior by modeling its characteristics in isolation of any architectural details. We then estimate the application's execution cost on a target platform described by its available hardware resources (e.g. number and type of execution units, cache size and memory access latency etc.).

To characterize an application's single node performance, we consider both the number of floating point operations executed as well as its memory access pattern. We do not aim to predict an exact execution time, but rather give an estimated execution time to the scheduler so that it can determine the hardware configuration that is most effective for the application components from the set of nodes available on the Grid. To understand the amount of computation performed by an application for a particular program input, we use hardware performance counters to collect floating-point operation counts from several executions of the program with different, small-size input problems. We then apply least square curve-fitting on the collected data to predict for an actual input data-set.

To understand an application's memory access pattern, we collect histograms of memory reuse distance (MRD) - the number of unique memory blocks accessed between a pair of references to the same block - observed by each load and store instruction [23]. Characterizing memory access behavior for programs in this way has two major advantages. First, data reuse distance is independent of cache configuration or architecture details. Second, reuse distance is a measure of data reuse, which is the main determinant in cache performance.

We collect reuse distance information separately for each reference in the program, for several, small-size input problems. We use the memory reuse distance data to model the behavior of each memory instruction, and to predict the fraction of hits and misses for a given problem size and cache configuration. Our modeling strategy dynamically finds groups of accesses that have a similar growth function for the reuse distance, and models each such group by two polynomials: one models how the number of accesses in that group changes with problem size, and one models how the average reuse distance of those accesses changes with problem size. To determine the cache miss count for a different problem size and cache configuration, we evaluate the MRD models for each reference at the specified problem size, and count the number of references with reuse distance greater than the target cache size.

As mentioned above, our goal is to indicate better machines for execution rather than predict the exact execution time for the application. We use the following simplified model.

$$EstExecTime(psize) = \frac{A + B + C + D}{CpuClock(arch)} \quad (3)$$

$$A = k_0 \times \frac{totalFp(psize)}{FpPipelineNum(arch)} \times FpRptRt(arch) \quad (4)$$

$$B = k_1 \times L1MissCnt(psize) \times L1MissPnlty(arch) \quad (5)$$

$$C = k_2 \times L2MissCnt(psize) \times L2MissPnlty(arch) \quad (6)$$

$$D = k_3 \times L3MissCnt(psize) \times L3MissPnlty(arch) \quad (7)$$

In the equations, $\{k_0, k_1, k_2, k_3\}$ are constants, $psize$ is the problem size and $arch$ is the target architecture. $FpRptRt(arch)$ is the repeat rate of the floating point pipeline. It is the number of cycles that occur between the issue of one instruction and the issue of the next instruction

to the same execution unit. MissPnlty, the penalty for a miss in an arbitrary level of the memory hierarchy, is the difference between the access time to the next memory level and the access time to the current memory level.

$$L(j)MissPnlty(arch) = P - Q \quad (8)$$

$$P = L(j+1)Latency(arch) \quad (9)$$

$$Q = L(j)Latency(arch) \quad (10)$$

The rank values can now be determined using the performance models and equation (1). We build a matrix using these rank values and call it the Performance Matrix, M, where the entry p_{ij} denotes the rank value of executing the $i-th$ component on the $j-th$ resource. Once we have this matrix, we can solve for the final mapping of components to resources.

3.3 Solving for final mapping

The mapping problem is an NP-Complete problem since Minimum Multiprocessor Scheduling problem is NP-Complete [13] and can be reduced to our mapping problem. An optimal approach for solving the mapping problem is to model the problem as an Integer Linear Programming (ILP) problem and solve the corresponding ILP. But it turns out that, in our case, with the ILP modeling we came up with, the number of variables in the corresponding ILP is too large for the available solvers. So we resorted to known heuristics to solve the mapping problem.

3.3.1 Heuristic Approach

We have chosen to apply three heuristics from the domain of scheduling parameter sweep applications [5, 30]. These heuristic approaches to finding a mapping run in polynomial time but don't guarantee an optimal mapping. The three heuristics we chose are

- **Min-min heuristic**: For each component, the resource having the minimum estimated completion time (ECT) is found. Denote this as a tuple (C, R, T), where C is the component, R is the resource for which the minimum is achieved and T is the corresponding ECT. In the next step, the minimum ECT value over all such tuples is found. The component having the minimum ECT value is chosen to be scheduled next. This is done iteratively until all the components have been mapped. The intuition behind this heuristic is that the makespan increases the least at each iterative step with the hope that the final makespan will be as small as possible.

- **Max-min heuristic**: The first step is exactly same as in the min-min heuristic. In the second step the maximum ECT value over all the tuples found is chosen and the corresponding component is mapped instead of choosing the minimum. The intuition behind this heuristic is that by giving preference to longer jobs, there is a hope that the shorter jobs can be overlapped with the longer job on other resources.

- **Sufferage heuristic**: In this heuristic, both the minimum and second best minimum ECT values are found for each component in the first step. The difference between these two values is defined as the sufferage value. In the second step, the component having the maximum sufferage value is chosen to be scheduled next. The intuition behind this heuristic is that jobs are prioritized on relative affinities. The job having a high sufferage value suggests that if it is not assigned to the resource for which it has minimum ECT, it may have an adverse effect on the makespan because the next best ECT value is far from the minimum ECT value. A high sufferage value job is chosen to be scheduled next in order to minimize the penalty of not assigning it to its best resource.

We run all three heuristics and choose the mapping that delivers the minimum makespan. In the pseudo-code for the overall Workflow scheduling presented below, ECT(j,R) is the estimated completion time of a particular component on a particular resource. EAT(R) is the expected time at which the resource, R will be next available [probably after the previous component finishes on the same resource]. The overall algorithm described in Algorithm 1 works as follows. For each heuristic, until all components in the workflow are mapped, the current set of available components are found. The rank matrix is then obtained for the set of available components. Then, Algorithm 2 is invoked with the current rank matrix, available components and heuristic as arguments. Algorithm 2 implements the core scheduling heuristics. Depending on the current heuristic, it returns a mapping for the current set of available components. Algorithm 1 updates the mappings and current makespan. When the outer loop of Algorithm 1 finishes execution, we have the mappings and makespans corresponding to the three heuristics. We choose the mapping that gives the minimum makespan among the three and output that as the final mapping.

4. Workflow Execution

In this section we address the issue of automating the Grid execution of the entire Workflow. We have used and extended the GrADSoft infrastructure to handle launching of Workflow style applications. The application along with the performance model is handed over to the Workflow Engine, which in unison with the Workflow Scheduler schedules the application components onto available resources.

```
foreach heuristic do
    while all components not mapped do
        Find availComponents; // satisfy dependencies
        Calculate the rank matrix;
        findBestSchedule(availComponents, heuristic);
    endwhile
endforeach
Select mapping with minimum makespan among three;
Output selected mapping;
```

Algorithm 1. Workflow Scheduling

```
while all availComponents not mapped do
    foreach Component, j do
        foreach Resource, R do
            ECT(j,R)=rank(j,R)+EAT(R);
        endforeach
        Find minECT(j,R) over all R;
        Find 2nd-minECT(j,R) over all R;
    endforeach
    Calculate min(minECT(j,R)) over all j; //min-min
    Calculate max(minECT(j,R)) over all j; //max-min
    Calculate min(2nd-minECT(j,R)-minECT(j,R))
    over all j; //sufferage
    Store mapping;
    Update EAT(R) and makespan;
endwhile
```

Algorithm 2. findBestSchedule

The Workflow Scheduler consults the GrADS information system for MDS/NWS information and uses the available performance models. Once the set of resources are chosen, the GrADS Application Manager is invoked. The GrADS Application Manager is the central controller for the GrADS execution system. It interacts closely with the GrADS Binder to launch the Workflow Application components onto actual Grid resources.

The GrADS Binder provides the final modifications to the application before it is launched on the grid. The Binder is a distributed component and executes on all resources on which the application is launched. It is responsible for compiling the GrADS application, optionally optimizing it for the target machine, and launching the program on the grid.

The Binder executes on each resource that is chosen by the workflow scheduler. At each resource, the Binder must be aware of the locations of software resources. These software resources consist not only of application specific libraries and header files, but also of Binder code and libraries on the resource. One approach to solve this would be to specify software installation locations for every resource. However, we found this approach to be overly restrictive. Hence, the Binder uses the GrADS Information Service (GIS) to locate necessary software on the scheduled resource. On execution, the Binder first queries GIS for the location of the local Binder code. It then launches the local copy of the Binder which further queries GIS for the locations of application-specific libraries. Having obtained resource-specific information, the Binder analyzes and instruments the code by inserting AutoPilot code senors [26] to enable runtime monitoring.

The final function of the Binder is to enable the actual launch of the application. The computational resources in the testbed varies from site to site. Currently, there are Pentium IA-32 machines, Opteron machines and IA-64 based Itanium machines on the GrADS testbed. Consequently, the GrADS Binder has to deal with heterogeneity in the target machines. The design of the Binder was carefully chosen to accommodate differences in machine characteristics. By using a high-level representation of the program and configuring and compiling it only at the target machine, the code is not specialized for a particular architecture when the Binder receives it. As a result, the Binder can successfully cope with heterogeneous architectures by configuring, compiling and linking the application components at the target resources and then launching them. Preserving high-level program information provides opportunities for architecture-specific optimizations.

5. Experimental Evaluation

In this section, we evaluate our heuristic workflow scheduling strategy by comparing it with other workflow scheduling strategies. We describe briefly salient features of an example workflow application, the other scheduling strategies and the Grid testbed on which the application is scheduled and executed. We then present the results of the experiments.

5.1 A Workflow Application Test Case - EMAN

We apply the scheduling strategies in the context of launching EMAN [Electron Micrograph ANalysis] [21], a Bio-Imaging workflow application onto the Grid. EMAN has been developed at the Baylor College of Medicine and primarily deals with 3D reconstruction of single particles from electron micrographs. Figure-1 gives an overview of the EMAN application.

A preliminary 3D model is constructed from electron micrographs. The preliminary 3D model is iteratively refined to obtain the final 3D model of the particle. Human intervention and expertise is needed to come up with a preliminary 3D model. The refinement from a preliminary model to the final 3D model is fully automated. This refinement

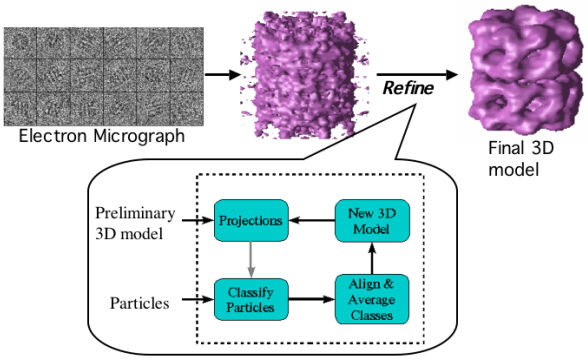

Figure 1. EMAN Overview

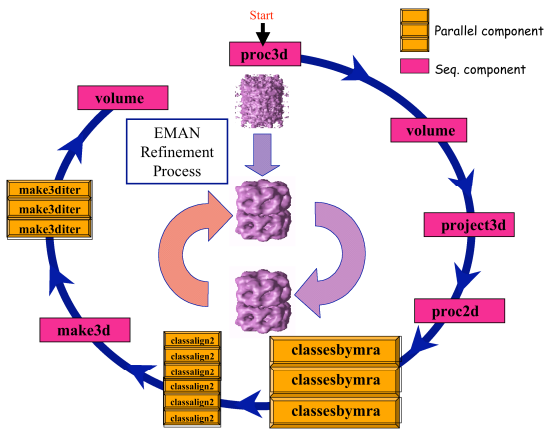

Figure 2. EMAN Refinement Workflow

step is the most computationally intensive step and benefits from harnessing the power of the Grid. In this work we have scheduled and executed the refinement workflow for EMAN. The different components in the EMAN refinement Workflow is described in Figure-2. It is essentially a linear Workflow with some of the components being parallel parameter sweep steps. 'proc3d', 'project3d', 'proc2d' etc. are relatively inexpensive sequential components that are executed toward the beginning of the workflow. The next component, 'classesbymra', is the most compute intensive parameter sweep step and takes up more than 90% of the total time to do the refinement. Efficient scheduling of the 'classesbymra' instances is very important to achieve a good makespan. While 'classesbymra' can be distributed across multiple clusters, the next parallel component in the workflow, 'classalign2', can only be run on a single cluster.

We have scheduled EMAN Refinement Workflow components using the Workflow Scheduler and executed the Workflow on distributed, heterogeneous platforms using the GrADS Binder and the rest of the GrADS execution system.

5.2 Design of Experiments

We wanted to investigate the importance of performance models and heuristics in scheduling workflows. Hence, we experimented with four scheduling strategies to isolate the effects of each of these factors on the quality of schedule [hence quality of application makespan] obtained. We picked the random scheduling as a baseline case. This is in essence what is used in the Condor DAGMan tool, where the available components are dynamically mapped to resources without any consideration of the resulting overall makespan. In order to investigate the value of (1) use of performance models and (2) use of heuristics for taking the scheduling decisions when using a plan-ahead whole workflow scheduling, the following scheduling four strategies were compared.

- RN: Random Scheduling with no performance models. In this scheme the application components are mapped randomly. Each available component is mapped to a resource that is randomly picked from the universe of resources.

- RA: Weighted random scheduling with accurate performance models - RA. In this scheme, the number of instances mapped to a randomly picked resource depends on the accurate performance model of the instance on that resource. Proportionally more instances are mapped to 'better' resources.

- HC: Heuristic Scheduling with crude performance models based on CPU power of the resources. In this scheme, we use the scheduling heuristics described in this paper, but with only crude performance models to determine the rank values. Rank value of a component on a resource is just the relative value [ratio] of the CPU speed of the resource to the CPU speed of the resource having the best MHz rating.

- HA: Heuristic Scheduling with accurate performance models generated semi-automatically. This is the Workflow scheduling scheme described so far in the paper.

We compare the makespan of the 'classesbymra' step for each scheduling strategy. We also evaluate the load balancing capabilities of the HA strategy.

5.3 Testbed and Data-set for Experiments

The testbed for the experiments consisted of distributed, heterogeneous resources from multiple participating institutions. There were 64 dual processor 900MHz Itanium IA-64 nodes at the Rice University Terascale Cluster [RTC]. There were 16 2009MHz Opteron nodes at the University of Houston Opteron cluster [medusa] and 60 dual processor 1300MHz Itanium IA-64 nodes at the University of Houston Itanium cluster [acrl]. 7 Pentium IA-32 nodes from the torc cluster at the University of Tennessee, Knoxville were also used for some experiments. Note that the testbed was heterogeneous in terms of architecture, CPU speeds, memory and storage.

Two EMAN refinement data-sets were used for the experiments. One was a moderately large data-set corresponding to a virus particle called 'rdv'. The input data-set was 2GB and was replicated across all the clusters. The other data-set was relatively small corresponding to the 'groel' particle. The input data-set was 200MB and was replicated across all clusters.

5.4 Results

Table-1 and Figure-3 hows the results of comparisons of the 4 scheduling strategies for the rdv data-set run across the Itanium RTC and Opteron medusa clusters. The table shows for each scheduling strategy, the number of 'classesbymra' instances mapped to each cluster [i(R) for RTC and i(M) for medusa], the number of nodes picked in each cluster by the scheduling strategy [n(R) for RTC and n(M) for medusa], the execution time in minutes at each cluster [t(R) for RTC and t(M) for medusa] and the overall makespan [MS]. The results show that the makespan obtained using the HA strategy is better than the one obtained using the RN strategy by a factor of 2.21 and is better than the ones obtained using the HC and RA strategies by a factor of 1.5. This shows that good makespan is attributed to both the factors - the heuristics used and accurate relative performance models. With either of them, we get a makespan improvement of a factor of 1.5, while using both gives a makespan improvement of a factor of 2.21.

Table-2 shows that the HA strategy achieves good load balance when 'classesbymra' is launched across the three clusters - RTC, medusa and acrl. Depending on the performance models and the heuristics, the scheduler mapped 29 instances to the 43 available RTC nodes, 42 instances to the 14 available medusa Opteron nodes and 39 instances to the 39 Itanium acrl nodes. t(R), t(M) and t(A) are the execution times for i(R), i(M) and i(A) instances at RTC, medusa and acrl clusters respectively. The difference in execution times between any two clusters is less than the granularity of the fastest execution time of a single instance of 'classesbymra'

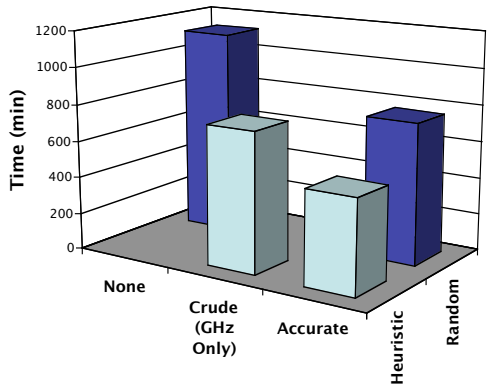

Figure 3. Comparison of different scheduling strategies

	i(R) (inst)	**i(M)** (inst)	**n(R)** (node)	**n(M)** (node)	**t(R)** (min)	**t(M)** (min)	**MS** (min)
HA	50	60	50	13	386	505	505
HC	58	52	50	13	757	410	757
RN	89	21	43	9	1121	298	1121
RA	57	53	34	10	762	530	762

Table 1. Comparison of makespans for different scheduling strategies with rdv data

i(R) (inst)	i(M) (inst)	i(A) (inst)	t(R) (min)	t(M) (min)	t(A) (min)	MS (min)
29	42	39	383	410	308	410

Table 2. Load balancing of the 'classesbymra' instances using the HA strategy

on the best resource. This implies that load balancing of the instances had been optimal for this case.

Also, the relative performance models of an instance on the sites matched closely to the relative actual execution times on the sites. The ratio of performance model values for an instance on a RTC node and a medusa node was 3.41 while the ratio of actual execution times for an instance on a RTC node and a medusa node was 3.82. The same set of values for an acrl node and a medusa node are 2.36 and 3.01 respectively.

We have obtained similar results when we ran the same experiment across 7 nodes at the UTK torc cluster and 6 nodes at the Itanium cluster at University of Houston. The scheduler mapped 68 instances to the 6 nodes on the Itanium cluster and 42 instances to the 7 nodes on the torc cluster. The execution time at the Itanium cluster was 84 hours 30 minutes and at the torc cluster was 81 hours 41 minutes, which resulted in a makespan of 84 hours and 30 minutes. Load balancing was optimal in this case too. The results are similar for the smaller 'groel' data-set. Other experiments with the smaller data-set have shown that inaccurate performance models and loading on the resources can have adverse impact on the makespans. The experiments for the results in Table-1 used only two clusters because hardware problems in the acrl cluster rendered it unusable during the course of these experiments.

We are currently working on applying the scheduling algorithm for mapping workflows from the Montage application [11]. Preliminary results from this application show that our scheduling strategy obtains upto 13 times better makespan than random strategy on heterogeneous platforms.

6. 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 [13]. Therefore, most of the literature deals with finding good heuristic solutions. There is a body of work on multiprocessor scheduling of independent application components. [30] gives an overview of different heuristics including min-min, max-min and sufferage. [5] 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 [22, 3]. [16] 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 directed acyclic graph (DAG) so these heuristics can't be directly applied. However, we use min-min, max-min and sufferage during the overall scheduling process.

Our work is also related to work in the area of approximation algorithms for different scheduling problems. The closest areas are minimum job-shop scheduling [28] and minimum precedence constrained scheduling [20]. The minimum job-shop scheduling problem is different from our problem because of a number of reasons. There are no precedence constraints between the operations belonging to different jobs. Also, the jobs take fixed amount of time on each processor and communication delays are not considered. These assumptions are not true for the problem at hand. In the case of the minimum precedence constrained scheduling, the tasks are of unit length and there are no communication delays. These assumptions are not true for our problem. There are approximation algorithms for variations like unit communication costs [25] and non-unit length tasks on 'uniformly related' processors [6]. Since our problem has stronger constraints like non-uniform task and communication costs, these algorithms can't be applied. The workflow scheduling problem can be precisely formulated as a multi-mode resource-constrained project scheduling problem with schedule dependent setup times [24]. We are not aware of any approximation algorithm for the same.

[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 grids are highly heterogeneous. [29] 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 [29] in the sense that we also consider the nodes level by level. The work in [2] on workflow-based approaches to plan whole workflows uses more sophisticated heuristics at each level and randomizes the decisions at each level to reduce the probability of being caught in local minima. It also compares static, whole workflow based scheduling with dynamic, task-based scheduling. The work

in [27] 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 as in Condor DAGMan [8], Data Grid resource broker [14] and GridLab resource broker [17], or random allocations or round-robin as in Pegasus [10]. The GridFlow [4] scheduler performs Workflow management and Scheduling side-by-side. It is mostly concerned with scheduling workflows of parallel message passing jobs. They take a hierarchical view of scheduling consisting of Global workflow Management with local sub-workflow scheduling. The GrADS MPI scheduler [9] is a modular, application level scheduler based on application performance model and mapping strategies. It is also concerned with scheduling parallel MPI jobs. Work in the GridLab group [24] uses genetic algorithms, simulated annealing and tabu search for workflow scheduling. However we are not aware of an experimental evaluation of their techniques. Our work is novel in its use of performance models and heuristic strategies for in-advance global workflow scheduling.

7. Conclusions and Future Work

In this work, we have come up with novel, in-advance global strategies to schedule application workflows onto the Grid using performance models and heuristics. We have implemented these strategies in the context of executing the EMAN application on the Grid. We have used the GrADS execution system that uses a novel binding scheme to deal with heterogeneity on the Grid. Our experiments on comparisons with other scheduling strategies have shown the value of performance models and heuristics in taking scheduling decisions. These strategies also result in good load balance across the Grid resources.

In the future, we plan to investigate better global heuristics that conditionally map the key steps in the workflow and solve the other mappings top-down and bottom-up. We also plan to work on optimizations like minimizing data movement by 'fusing' workflow nodes, potentially sacrificing some parallelism to reduce expensive communication. We plan to look into the issues on scheduling multiple DAGs and hierarchical workflow scheduling onto virtualized Grid resources.

8. Acknowledgments

We sincerely acknowledge the help we received from the research group of Dr. Wah Chiu and Dr. Steve Ludtke form the Baylor College of Medicine regarding all aspects of the EMAN application. We thank Anshuman Dasgupta, Mark Mazina and Dr. Keith Cooper from Rice University for their contributions to the GrADS infrastructure used for the experiments. We also acknowledge Dr. Sanjeeb Dash from IBM T. J. Watson Research center for his insights on different aspects of ILP approaches to scheduling. This material is based on work supported by the National Science Foundation under Grant No.ACI 0103759 (the GrADS Project) and Cooperative Agreement No. CCR-0331654 (the VGrADS Project). This work was supported in part by the Rice Terascale Cluster funded by NSF under Grant EIA-0216467, Intel, and HP.

References

[1] B. Barish and R. Weiss. Ligo and detection of gravitational waves. *Physics Today*, 52(10), 1999.

[2] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Resource allocation strategies for workflows in grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, to appear 2005.

[3] T. Braun. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In *IEEE Workshop on Advances in Parallel and Distributed Systems*, 1998.

[4] S. J. J. Cao, G. Nudd, and S. Saini. Gridflow: Workflow management for grid computing. In *3rd International Symposium on Cluster Computing and the Grid*, 2003.

[5] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *9th Heterogeneous Computing workshop (HCW'2000)*, 2000.

[6] A. Chudak and D. B. Shmoys. Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds. In *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 581–590, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[7] Condor Team University. Condor version 6.1.12 manual.

[8] Condor Team University. Dagman metascheduler.

[9] H. Dail, H. Casanova, and F. Berman. A modular scheduling approach for grid application development environments. Technical Report CS2002-0708, 2002.

[10] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[11] G. B. et al. Montage a grid enabled image mosaic service for the national virtual observatory. *ADASS XIII ASP Conference Series*, XXX, 2004.

[12] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Pro-*

ceedings of the 6th IEEE Symposium on High-Performance Distributed Computing, pages 365–375, 5-8 Aug. 1997.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. 1979.

[14] A. G. F. Giacomini, R. Peluso, and M. Sgaravatto. Definition of the architecture, technical plan and evaluation criteria for the resource co-allocation framework and mechanisms for parallel job partitioning, 2002.

[15] S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, and J. Saltz. Image processing on the grid:a toolkit or building grid-enabled image processing applications. In *3rd International Symposium on Cluster Computing and the Grid*, 2003.

[16] X. He, X. Sun, and G. von Laszewski. Qos guided min-min heuristic for grid task scheduling. *J. Comput. Sci. Technol.*, 18(4):442–451, 2003.

[17] K. K. Juliusz Pukacki. Wp9 resource management.

[18] Ken Kennedy et al. Toward a framework for preparing and executing adaptive grid programs. In *International Parallel and Distributed Processing Symposium*, 2002.

[19] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.

[20] J. K. Lenstra and A. H. G. R. Kan. Complexity of scheduling under precedence constraints. *Operations Research*, 26:22–35, 1978.

[21] S. Ludtke, P. Baldwin, and W. Chiu. Eman: Semiautomated software for high-resolution single-particle reconstructions. *J. Struct. Biol.*, 128:82–97, 1999.

[22] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

[23] G. Marin. Semi-Automatic Synthesis of Parameterized Performance Models for Scientific Programs. Master's thesis, Dept. of Computer Science, Rice University, Apr. 2003.

[24] M. Mika, G. Waligora, and J. Weglarz. *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.

[25] V. J. Rayward-Smith. Net scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18:55–71, 1987.

[26] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed. Autopilot: Adaptive control of distributed applications. In *HPDC*, pages 172–179, 1998.

[27] R. Sakellariou and H. Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Proceedings of 13th Heterogeneous Computing Workshop (HCW 2004), Santa Fe, New Mexico, USA*, 2004.

[28] D. B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for shop scheduling problems. In *SODA '91: Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, pages 148–157, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics.

[29] 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*, 13(3):260–274, 2002.

[30] Tracy D. Braun et al. A comparision of eleven static heuristics for maping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[31] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.