RICE UNIVERSITY

Toward a Tool for Scheduling Application Workflows onto Distributed Grid Systems

by

Anirban Mandal

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree
Doctor of Philosophy

Approved, Thesis Committee:

Kenneth W. Kennedy
Ann and John Doerr University Professor in Computational Engineering, Computer Science

---

Keith D. Cooper
Professor and Chair, Computer Science

John Mellor-Crummey
Associate Professor, Computer Science

Charles Koelbel
Research Scientist, Computer Science

William W. Symes
Noah Harding Professor and Chair, Department of Computational and Applied Mathematics, Rice University

Houston, Texas
April, 2006

Toward a Tool for Scheduling Application Workflows onto Distributed Grid Systems

Anirban Mandal

Abstract

In this dissertation, we present a design and implementation of a tool for automatic mapping and scheduling of large scientific application workflows onto distributed, heterogeneous Grid environments. The thesis of this work is that plan-ahead, application-independent scheduling of workflow applications based on performance models can reduce the turnaround time for Grid execution of the application, reducing burden of Grid application development. We applied the scheduling strategies successfully to Grid applications from the domains of bio-imaging and astronomy and demonstrated the effectiveness and efficiency of the scheduling approaches. We also proposed and evaluated a novel scheduling heuristic based on a middle-out traversal of the application workflow.

A study showed that jobs have to wait in batch queues for a considerable amount of time before they begin execution. Schedulers must consider batch queue waiting times when scheduling Grid applications onto resources with batch queue front ends. Hence, we developed a smart scheduler that considers estimates of batch queue wait times when it constructs schedules for Grid applications.

We compared the proposed scheduling techniques with existing dynamic scheduling strategies. An experimental evaluation of this scheduler on data-intensive workflows shows that its approach of planning schedules in advance improves over previous

online scheduling approaches.

We studied the scalability of the proposed scheduling approaches. To deal with the scale of future Grids consisting of hundreds of thousands of resources, we designed and implemented a novel cluster-level scheduling algorithm, which scales linearly on the number of abstract resource classes. An experimental evaluation using workflows from two applications shows that the cluster-level scheduler achieves good scalability without sacrificing the quality of schedule.

Acknowledgments

I would like to thank my advisor, Ken Kennedy, for his help and support without which this dissertation would not have been possible. Ken has not only taught me how to pursue independent research, but also helped me achieve a certain level of maturity. His tremendous acumen combined with a personality that is magnetic yet gentle has been an inspiration that goes beyond this dissertation.

I am grateful to my committee members for their insights. Chuck Koelbel has been a constant source of detailed help regarding all areas of my research – defining research problems, designing experiments and writing technical papers. John Mellor-Crummey and William Symes have suggested interesting ideas during my thesis proposal, which helped be hone my thesis. Keith Cooper has been extremely supportive of all my efforts.

Throughout my PhD, I have worked on the GrADS and the VGrADS projects. The common research infrastructure provided by the projects has been extremely useful for my research and experiments. The infrastructure is a result of a lot of hard work of GrADS and VGrADS personnel and I take this opportunity to thank them for all the support. In particular, I would like to thank Mark Mazina, Holly Dail, Martin Swany, Celso Mendes, Asim Yarkhan, Anshuman Dasgupta, Gabriel Marin, Bo Liu, Lavanya Ramakrishnan and the system administrators at different sites. I would also thank Yang Zhang and Daniel Nurmi for help in implementation at various points of time. Research discussions with several GrADS and VGrADS PIs like Rich Wolski, Carl Kesselman, Andrew Chien, Henri Casanova and Lennart Johnsson have helped

me shape my research. I would like to thank my all my external research collaborators – Sanjeeb Dash, Ewa Deelman, Jim Blythe, Sonal Jain and Gurmeet Singh.

I would like to thank all my friends and research staff in the compiler group for many a stimulating discussion and a vibrant work environment – Daniel Chavarria-Miranda, Yuan Zhao, Rajarshi Bandyopadhyay, Apan Qasem, Cheryl McCosh, Cristian Coarfa, Yuri Dotsenko, Arun Chauhan, Zoran Budimlic, Timothy Harvey, Guohua Jin and Todd Waterman. I also thank my friends in the systems group – Santashil Palchaudhuri, Amit Saha, Anupam Chanda, Animesh Nandi, Atul Singh and Sitaram Iyer. My sincere thanks to all the support staff at the Computer Science department, including Penny Anderson, Darnell Price, Iva Jean Jorgensen, Lena Sifuentes, Donna Jares, Melissa Cisneros, Rhonda Guajardo, Bel Martinez and BJ Smith, for being so helpful.

I am extremely grateful to my parents and uncle, who have been with me throughout this long journey. They have provided all the support and encouragement that make me who I am. I hope I have fulfilled a part of their dream. My sincere thanks to Kuntal Das, Indrani Dasgupta and Naureen Shahid for always being there and supporting me. My school friends – Subhendu Chakraborty, Saraswata Chaudhuri, Samik Raychaudhuri, Tirtha Chatterjee, Arnab Chakraborty and Bhaskar Dutta have been extremely supportive all through. I would also like to thank Sukanya Das, Ivy Ghose, Susmita Roy, Ramkumar Balasubramanian and Jatin Shah for their encouragement.

Contents

Illustrations

Tables

Chapter 1

Introduction

Grid computing is defined as *coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations* [IKT01]. The idea is to view the global information infrastructure as an active, computational resource. Just like the electric power grid provides pervasive access to electricity, the "Grid" [FK99] promises to connect geographically distributed computers, databases, instruments and people in a fabric of computing that can be used as an integrated problem-solving resource in diverse fields of science and engineering and provide pervasive access to large scale computation and data. Grid computing typically involves using many resources (compute, network, data, instruments etc.) to solve a single, large problem that could not be solved on any single resource [NSW03].

An integral and essential component of Grid computing is *Grid Resource Management*. The responsibilities of a Grid resource management system are identifying application requirements, matching Grid resources for the application, allocating the resources, scheduling the application onto the resources and monitoring the application and resources as the application executes on them. Requirements of applications are diverse and so are the Grid resources in terms of processor capabilities, data capabilities, network capabilities, instruments and other services. Scheduling diverse applications on heterogeneous, distributed, dynamic Grid computing systems is a hard problem and is one of the most important components of a Grid Resource Management system. Scheduling applications to the "right" set of grid resources can have

a dramatic effect on the performance of a Grid application. A good job of scheduling results in faster turnaround times for the applications.

1.1 Motivation

Through initial progress in Grid Resource Management, it is possible to securely submit/run jobs across multiple administrative domains and transfer large data sets (files) across them. But, these systems are still in their infancy offering only very basic services of secured access and remote job execution. This has provided a platform for researchers to address the issues on how to use the resources more efficiently and effectively. Unfortunately, the use of good Grid Resource Management tools is far from ubiquitous and the scheduling support in the existing ones is rudimentary and has room for improvement.

From the survey of the different grid applications discussed in detail in chapter 3, it is evident that the Grid is suitable for coarse-grained parallel applications. These applications consist of a set of application components that need to be executed in a certain partial order for successful completion. The application components themselves are either coarse-grained parallel components, parameter sweeps or tightly-coupled components. Applications demonstrating these characteristics are referred to as workflow applications and represent the dominant class of applications on the Grid. There is a need for a framework to efficiently schedule and execute the workflow applications on distributed, heterogeneous Grid systems. Efficient execution is contingent upon a "right" choice of resources and a schedule on the chosen resources. Most successful Grid applications include their own application scheduler. Producing such a scheduler is tedious and error-prone. Hence, there is a definite need for generic tools for automatic application scheduling on the Grid. Large-scale scientific

application workflows can achieve higher performance when they are efficiently and effectively mapped and scheduled onto Grid resources.

1.2 Tools for Application Scheduling on the Grid

The vision for achieving successful Grid computing is to enable scientists seamlessly solve large-scale problems using resources on the Grid so that the complexity of the Grid is hidden from the scientist. For example, a physicist sitting in Houston should be able to launch from his laptop a large-scale simulation that uses instrument data from a laboratory in Florida, results of previous runs from a database in Washington and computation power from a supercomputer at Chicago and a large cluster at San Diego and is able to get back the results in Houston. The important point is that these steps need to happen seamlessly from the viewpoint of the scientist, who is shielded from the dynamic, distributed and heterogeneous nature of the Grid infrastructure.

Unfortunately, we are a long way from achieving this vision. The state of the art for the scientist is to still transfer the data manually, choose the resources manually and deal with all the Grid middleware directly to run his jobs and get the results back. Hence, research in the GrADS[ea02] and VGrADS [vgr] project has focused on the problem of "how do we develop tools to make programming for the Grid easier for the end-user/scientist?"

In order to answer that question, we need to demonstrate tools that enable automatic application level scheduling of the application workflows provided by the scientist, so that the workflows can seamlessly choose and access resources and at the same time achieve good turnaround time. In other words, the scientist no longer has to manually interact with the Grid system to choose resources and schedule the workflows onto them.

The current Grid workflow frameworks have limited scheduling capabilities [DBG+03, dag, TMSW01]. Most of them use random, online or first-come-first-serve scheduling policies, resulting in poor turnaround times. Hence, we propose tools that automatically produce efficient application schedules from input application workflows provided by the scientist using accurate performance predictions and heuristics. This enables achieving both of our objectives:

1. Ease of programming for the Grid.

2. Better application turnaround time for high performance and efficient utilization of Grid resources.

1.3 Thesis

My thesis is that *plan-ahead, application-independent scheduling of workflow applications based on performance models can reduce the turnaround time for Grid execution of the whole application, reducing the burden of Grid application development.* To support this thesis, I have developed heuristic, plan-ahead workflow scheduling algorithms that use application performance models. I have implemented the algorithms and have shown the effectiveness and efficiency of the same by evaluating the algorithms in context of real scientific workflows - one from the domain of bio-imaging (EMAN) and another from the domain of astronomy (Montage) using real Grid testbeds and simulations.

1.4 Research Contributions

The main contribution of this work is a set of techniques for scheduling application workflows onto Grid resources.

- I have developed a plan-ahead heuristic scheduling algorithm based on performance models for scheduling scientific workflows. I investigated the effect of performance models, heuristics and machine loads on the generated schedule and hence overall performance for real applications.

- I designed and implemented a novel middle-out workflow scheduling heuristic to tackle the short-sightedness of the plan-ahead top-down workflow scheduler and evaluated it with an application.

- I investigated a comparison of plan-ahead scheduling with existing dynamic, online scheduling strategies.

- I also investigated the issue of scalable workflow scheduling techniques for future Grids, which may consist of hundreds of thousands of resources. I designed and implemented a cluster-level workflow scheduler that scales to thousands of resources and evaluated it with the applications.

- I have extended the workflow scheduling tool to incorporate batch queue wait times for taking the scheduling decisions. This increases the applicability of the scheduling tool to batch-queue controlled systems.

1.5 Organization

The thesis is organized as follows. In chapter 2, we present related work. In chapter 3, we present a survey on existing Grid applications. We describe our plan-ahead workflow scheduling strategies and evaluations of the same in chapter 4. In chapter 5, we present a new "middle-out" plan-ahead workflow scheduling approach and its evaluations. We present a comparison of the plan-ahead approach with dynamic task based approaches to workflow scheduling in chapter 6. In chapter 7, we describe

scalable workflow scheduling strategies using virtual grids and a new cluster-level scheduling technique and its evaluation. We conclude the thesis in chapter 8..

Chapter 2

Related Work

In this chapter, I summarize related work on DAG scheduling on homogeneous and heterogeneous platforms, multiprocessor scheduling of independent tasks, current state of the art on workflow management, Grid resource management in general and other scheduling strategies for the Grid. I also present comparisons with my work on workflow scheduling (where appropriate).

2.1 DAG Scheduling

This section describes related work on scheduling Directed Acyclic Graphs (DAG) onto homogeneous and heterogeneous platforms.

2.1.1 DAG Scheduling for Homogeneous Platform

There is a large body of literature on scheduling directed acyclic task graphs onto a set of homogeneous processors. Kwok et al.[KA99b] [KA99a] present an excellent survey. The static task scheduling problem can be described as follows. An application is represented by a directed acyclic graph in which the nodes represent application tasks and the edges represent dependencies between the tasks. Each node in the DAG has an associated computation cost that denotes the expected execution time of the node on a processor (note that the cost is same for all the processors). Each edge in the DAG has an associated communication cost that denotes the expected communication time between the tasks. The underlying platform is a contention-free network of fully

connected homogeneous processors that can send and receive any number of messages in parallel. Tasks can begin execution only when all the inputs are available. Once started, the tasks execute to completion without interruption. This model is also called the macro-dataflow model. The aim is to find a mapping and start times of tasks to the processors such that (1) the precedence constraints are preserved and (2) the schedule-length or makespan is minimized. The general DAG scheduling problem has been shown to be NP-complete [GJ79]. Only a few very restrictive special cases have a polynomial time solution. Hence, there are many heuristic solutions in the literature broadly classified into three categories. We describe them in the following sections.

List-Scheduling Based Heuristics for Bounded Number of Processors

List-scheduling based heuristics assign priorities to tasks in the DAG and place the tasks in a ready list in descending order of priorities. Then, a task having higher priority is considered for scheduling before a task having lower priority. There are numerous methods for assigning the priorities, maintaining the ready list and assigning a task to a processor. The different heuristics differ in terms of these. Most of the time, the priority of a task is a function of top-level (t-level) , bottom-level (b-level), SL (static b-level), length of critical-path (CP) or some combination of these. The *t-level* of a task/node(n) is the length of the longest path from an entry node to n (excluding n), where length of a path is defined as the sum of node and edge weights along the path. The *b-level* of a task/node(n) is the length of the longest path from n to any exit node. The *SL* is the *b-level* without considering the edge weights. The *critical-path* is a path from an entry node to an exit node in the DAG such that the length of the path is the maximum.

Here are a few of the list-scheduling heuristics. Highest Level First with Esti-

mated Times (HLFET) [ACD74] prioritizes according to SL and schedules a task to a processor that allows the earliest start time. The Insertion Search Heuristic (ISH) [KL88] uses SL to assign priorities and schedules a task to a processor that allows earliest start time. In addition , ISH also tries to insert other unscheduled nodes from the ready list in the "holes" created by partial schedules. The Modified Critical Path (MCP) [WG90] heuristic uses the ALAP time (length of critical-path - b-level) of a node to assign the priorities, ties being broken by considering ALAP times of the children. It schedules a task to a processor that allows the earliest start time using the insertion method. The Earliest Time First (ETF) [HCAL89] heuristic computes the earliest start times for all the ready nodes on all the processors with ties being broken by a higher SL. It selects for scheduling the node-processor pair giving the earliest start time. The Dynamic Level Scheduling (DLS) [SL93] heuristic uses the Dynamic Level (DL) attribute, which is the difference between the SL of a node and its earliest start time on a processor. For all the ready nodes the value of DL is calculated for all the processors. The node-processor pair giving the highest value of DL is scheduled next.

Clustering-based Heuristics for Unbounded Number of Processors

The clustering-based heuristics map the tasks in the given DAG to an unlimited number of clusters. Mapping two tasks in the same cluster implies scheduling them onto the same processor. At each step, a task is selected for clustering. The selection for the next task depends on criteria that varies from one heuristic to another. Each iteration refines the previous clustering by merging some clusters. Any clustering heuristic requires additional steps to generate the final schedule - a cluster merging step to merge clusters if the number of clusters obtained is greater than the number of processors available, a cluster mapping step and a task ordering step for ordering

the mapped tasks within each processor.

Here are a few of the clustering based heuristics. The Edge Zeroing (EZ) [Sar89] heuristic selects the next cluster for merging based on the edge-weights. At each step, it finds the edge with the largest weight and merges the end-points (thereby mapping the two nodes to the same processor and eliminating the communication cost) if the merging process does not increase the current parallel completion time. Parallel completion time is the maximum value of the sum of t-level and b-level over all the nodes. The Linear Clustering (LC) [KB88] heuristic merges all nodes in the current critical-path into a cluster by zero-ing all the edges in the current critical-path. Then those nodes and edges incident on them are removed from the graph and the process is repeated for the unexamined portion of the graph. The Dominant Sequence Clustering (DSC) [YG94] heuristic considers the dominant sequence (DS) of the graph. DS is the length of the critical-path in the partially scheduled DAG. DSC gives higher priorities to free nodes that belong to the dominant sequence for the next merging step. So, the priority of a free node is the sum of the t-level and b-level. The free node is merged to the cluster of one of its predecessors that give the most reduction in the t-level. The t-level of the successor nodes are updated and the algorithm iterates until all the nodes are examined. The other important clustering based heuristics are the Mobility Directed (MD) [WG90] and Dynamic Critical Path (DCP) [KA96] heuristics. Both of them use a mobility attribute, M, which is a function of the current critical-path length, sum of t-level and b-level and the weight of the node. M is zero for nodes in the critical-path. Smaller M values are given higher priority. These two heuristics differ in the way the selected node is assigned a cluster. MD schedules the node in the first idle time-slot while DCP computes a look-ahead.

Other heuristics

Another class of heuristics for scheduling DAGs to homogeneous processors is the Task-Duplication-Based (TDB) heuristics [KA99b]. The idea behind TDB heuristics is to schedule a task graph by mapping some of its tasks redundantly, so as to reduce the effect of interprocessor communications. The TDB heuristics differ on the strategy to select the next task for duplication. The TDB heuristics are more expensive than heuristics belonging to the previous two classes. DAG scheduling based on guided random search techniques like genetic algorithms, simulated annealing etc. are expensive too.

We cannot apply the DAG scheduling heuristics for homogeneous platforms in the Grid context because of the heterogeneity on the Grid in terms of both computation and communication characteristics.

2.1.2 DAG Scheduling for Heterogeneous Platform

A few research groups have studied the task scheduling problem for heterogeneous systems. In most of the literature for heterogeneous DAG scheduling, the underlying platform is a fully-connected contention free network of q heterogeneous processors. The data transfer rates between processors is stored in a matrix B of size $q \times q$. In addition to that there is a $v \times q$ computation matrix, W, in which the w_{ij} gives the expected execution time of task v_i on processor p_j. The objective is to find a schedule such that the schedule-length/makespan is minimized. Most of the heuristics are based on list-scheduling techniques extended for heterogeneous platforms.

The Generalized Dynamic Level (GDL) [SL93] heuristic is the natural extension of the DL heuristic for homogeneous platforms. The node weight is calculated as the median cost of the node over all processors. The static-level for all the nodes is

calculated based on these node weights. Static-level computation ignores the communication costs. The dynamic-level attribute for a node-processor pair is calculated as the difference between the static-level and the earliest start time on the processor. The node-processor pair with the highest dynamic-level is scheduled next.

The Best Imaginary Level (BI) [OH96] heuristic is also a list-scheduling based heuristic that uses the attribute called Best Imaginary Level (BIL). BIL for a node-processor pair (N_i, P_j) is calculated as the sum of the expected execution cost of N_i on P_j and the maximum of the communication cost over all children of N_i. As priority for a node-processor pair, BI uses Best Imaginary Makespan (BIM) that is a function of BIL and the earliest start time of the node on the processor. The node-processor pair having the smallest BIM is selected next.

In the Heterogeneous Earliest Finish Time (HEFT) and Critical Path on a Processor (CPOP) [THW02] heuristics, the node weight, $\overline{w_i}$ for the i-th node is calculated as the average weight of the node over all processors: $\overline{w_i} = \sum w_{ij}/q$. The weight of the edge (i, k) is defined as the average communication cost ,$\overline{c_{ik}} = \overline{L} + \frac{data_{ik}}{\overline{B}}$ over all processor pairs. Both of them are list-scheduling based heuristics. In HEFT, the priority of the tasks is set as the upward rank value $rank_u$ of the task. $rank_u$ of a node, n_i, is $rank_u(n_i) = \overline{w_i} + maxsucc_j(\overline{c_{ij}} + rank_u(n_j))$. $rank_u$ is calculated recursively upward from the exit nodes. The highest priority task is selected and is scheduled on a processor that gives the earliest finish time for the task using an insertion based policy. In CPOP, the priority is set as the sum of upward and downward ranks. Downward rank, $rank_d$ is defined as: $rank_d(n_i) = maxpred_j(rank_d(n_j) + \overline{w_j} + \overline{c_{ji}})$. If the next task with the highest priority is in the critical-path, it is scheduled to a designated critical-path processor; else, it is assigned to the processor having the earliest finish time for the task, both cases using an insertion based approach.

We may not apply the above heuristics directly in the Grid context because (1)

dramatic heterogeneity in both computational and communication characteristics on the Grid make them unsuitable - implying average and median values making no sense (2) list-scheduling heuristics are too sensitive to the methods of node and edge weighting [ZS03] and (3) list-scheduling algorithms don't schedule the nodes in a chunk, which is important in the Grid context.

The Levelized-Min Time (LMT) [THW02] heuristic groups the tasks that can be executed in parallel using a level attribute. A task at a lower level has a higher priority than a task at a higher level. Each task at a level is assigned to the processor such that the sum of the task's computation cost and total communication costs with tasks at previous levels is minimized. There are some similarities between LMT and my initial approach.

My work on scheduling heuristics is closest to the work presented by Sakellariou et al. in [SZ04]. In this work, the upward rank of each node is calculated as in HEFT. Hence, the node and edge weights are average weights. The nodes are sorted according to descending order of the upward rank values. They are scanned according to this order to create independent sets of nodes that are then scheduled in a chunk at a later stage. Each independent set is scheduled using a Balanced Minimum Completion Time (BMCT) heuristic. The main drawback is considering the average values, which may not make sense in the Grid context because of high heterogeneity.

There is also related work on heterogeneous scheduling assuming a non macro-dataflow model [BLR02b]. It assumes a different communication model called the one-port model. Most of this work is on master-slave tasking, divisible-load scheduling, steady state scheduling and theoretical bounds thereof. These algorithms don't apply to the DAG model.

The work by Maheswaran et al. in [MS98] describes a dynamic matching and scheduling algorithm for heterogeneous computing systems, where scheduling deci-

sions are taken during runtime. We compare our static scheduling approach with a dynamic scheduling approach in chapter 6.

2.2 Multiprocessor Scheduling of Independent Tasks

The problem of scheduling independent tasks onto a heterogeneous set of processors has been studied extensively in the literature. The problem is stated as follows. Given a set of unrelated/independent tasks, a set of heterogeneous processors and estimates of execution times of each task on each processor, the aim is to find an assignment and order of the tasks on each processor such that the overall makespan is minimum. This problem is also known as the minimum multiprocessor scheduling problem and belongs to the class of NP-complete problems - generalization of SS8 in Garey et al. [GJ79]. There are few polynomial-time approximation schemes based on linear programming to solve this problem [LST90] [JP99]. Though they are polynomial time schemes, they are expensive to implement.

Most of the literature in this area has resorted to heuristic techniques to solve this problem. Braun et al. [Tra01] present an excellent survey and comparison of eleven static heuristics of mapping independent tasks onto heterogeneous distributed computing systems. Opportunistic Load Balancing (OLB) heuristic assigns each task in an arbitrary order to the next available machine irrespective of expected execution times. It is a simple but ineffective heuristic. The Minimum Execution Time (MET) heuristic maps each task to the processor on which it has the best expected execution time. It is also a simple heuristic but causes severe load imbalances across the processors. The Minimum Completion Time (MCT) heuristic assigns the next task to the processor on which it has the earliest completion time. It tries to overcome some of the weaknesses of OLB and MET. The min-min heuristic considers the whole set of

unmapped tasks. It finds the set of minimum completion times (MCT) corresponding to each unmapped task. It then selects the task with the overall minimum MCT from the set to be mapped next. It continues until all the unmapped tasks are mapped. Min-min considers all unmapped tasks while MCT considers just one task. The intuition behind min-min heuristic is that at each mapping step the current makespan increases the least. The max-min heuristic is similar to min-min, the only difference being that after the set of MCT is calculated the overall maximum MCT value task is selected next for mapping. The intuition is that long tasks can be overlapped with shorter tasks in case of max-min. Duplex heuristic runs both min-min and max-min and selects the best makespan among them. The sufferage heuristic stores the sufferage value for each of the unmapped tasks. Sufferage value is the difference between the minimum completion time and the second minimum completion time. The task having the largest sufferage value is selected next for mapping. There are other AI-based approaches/heuristics for mapping, which are based on Genetic Algorithms (GA), simulated annealing (SA), Tabu search (Tabu), A* search etc. Simulation results in [Tra01] show that min-min and sufferage heuristics are the best in terms of quality of makespans over a large range of processor characteristics and the time it takes to find the mappings.

Heuristics for scheduling independent tasks have also been applied in the context of Grid applications that are parameter sweeps. The study in [CLZB00] extends the min-min and sufferage heuristics to incorporate data movement costs for input and output files. The same work also introduces a heuristic called XSufferage that is an extension of the sufferage heuristic. In XSufferage, the sufferage value is computed not with MCTs but with cluster-level MCTs, i.e. by computing the minimum MCTs over all hosts in each cluster. The results show that when file input/output is considered, XSufferage works best.

The study by He et al. in [HSvL03] uses a Quality of Service (QoS) guided min-min heuristic. Their basic premise is that tasks on the Grid may have extremely diverse QoS requirements. This means that some tasks may have very stringent QoS requirements and others may have low QoS requirements. They adapt the min-min heuristic to this scenario by first mapping all the available tasks with high QoS requirement followed by mapping of the low QoS tasks.

The work by Beaumont et al. in [BLR02a] deals with scheduling divisible workloads on heterogeneous systems with one-port communication model. It is mainly concerned with efficient distribution of workloads to processors (slaves) from the master processor for single and multi-round algorithms. It also has theoretical optimality results for the same. There are some other related work on heuristics for mapping the tasks online (non-batch mode). Maheswaran et al. [MAS+99] present a survey of the same. Caniou et al. [CJ04] present heuristics for dynamic, online mapping in the Grid scenario that outperform the MCT in several metrics including makespan.

The work in this area uses a set of independent tasks as the application model. In my work, the application model is a DAG model and hence these algorithms can't be applied directly. However, I use some of the heuristics from this domain (namely min-min, max-min and sufferage) to schedule a chunk of available tasks onto available Grid resources.

2.3 Workflow Management on the Grid

There are a few research groups working on automating management, scheduling and execution of application workflows on the Grid.

2.3.1 GriPhyN - Pegasus

The Pegasus [DBG+03] effort from the GriPhyN [grib] project deals with automating generation of job workflows on the Grid. These workflows describe the execution of a complex application built from individual application components. Their Concrete Workflow Generator (CWG) maps an abstract workflow defined in terms of application level components to a concrete workflow defined in terms of real executables and input/output file names. The concrete workflow is mapped to the set of available Grid resources.

Their infrastructure uses the Globus components for resource discovery (MDS), remote job submission (GRAM), high-performance data transfer (GridFTP) and management of replicated files (RLS). The Transformation Catalog (TC) provides a mapping between the application components and their physical location. The abstract workflow has information about all the jobs that need to be executed to materialize the required data. The CWG first reduces the abstract workflow, in the sense that if the required data has already been materialized for a component and can be obtained via an RLS query and data transfer, the node corresponding to the component is deleted. The CWG then finds out whether all required inputs for a component can be materialized or not. If yes, it assigns a random available Grid resource to execute the component. Data transfer nodes are added in the workflow, if necessary. Once the final concrete workflow is generated, the workflow is executed using Condor-G and DAGMan. The entire system was used to execute production runs of CMS [Wul98] workflows.

The strong point of this work is that this is an end-to-end system for execution of application workflows. However, the most important drawback of the system is that there is only bare-bone provision for resource selection and scheduling. Random or

round-robin scheduling can result in poor makespans for workflows. My work aims to address this issue when scheduling concrete workflows.

Pegasus also presents the option of using AI techniques. This part of the Pegasus effort aims at producing the concrete workflows from application metadata attributes using AI-Planning techniques. This workflow generator is called the Abstract Concrete Workflow Generator (ACWG). A typical AI-planning system takes as input the initial state (the current state of the world in some representation, the goal (a partial description of the desired world state) and a set of operators (described as a set of preconditions and effects that must hold true when an operator is applied). The planning system uses different search algorithms to achieve the goals from the initial state using the operators. There is a wide variation of search algorithms that can be used.

The authors cast the concrete workflow generation as an AI-planning problem. The ACWG models the application components along with data transfer as operators. State information includes description of available resources and files already registered in the RLS. The goal description may include a metadata specification of the information the user requires and the desired location of the output file. After the modeling is done, the ACWG uses the Prodigy planner to come up with a final plan. The planner uses a combination of local heuristics (for good choice of individual component assignments) and global heuristics (for overall makespan reduction) to come up with a high quality plan(schedule) and a concrete workflow. The main drawback of this approach is that AI-Planning is very expensive because of exhaustive search and backtracking.

2.3.2 GridFlow

The GridFlow [CNS03] work performs workflow management and scheduling side-by-side. It is mostly concerned with scheduling workflows of parallel message passing jobs. It takes a hierarchical view of scheduling. The application is viewed hierarchically. The applications is assumed to represented as a global workflow of local sub-workflows. The resources are viewed hierarchically too: the global Grid and the local Grids. There is a global workflow management step followed by local sub-workflow scheduling. The global workflow management step looks for a schedulable sub-workflow, the predecessor sub-workflows of which have all been scheduled. The start time of the chosen sub-workflow is configured with the latest end-time of its pre-sub-workflows. The details of the sub-workflow and the start time are then submitted to an ARMS (Agent Based Resource Management System) [CNS03] agent. ARMS agents work together to discover an available local Grid that can finish the sub-workflow execution at the earliest time. The sub-workflows are mapped to the local Grids using the Titan [CNS03] system. This system is used mostly for mapping workflows of tightly-coupled components. My work mainly focuses on mapping workflows of coarse-grained, loosely-coupled components.

2.3.3 Condor DAGMan

One of the most popular tools supporting the execution of workflow applications is the Condor DAGMan [dag] tool. DAGMan is a meta-scheduler for Condor jobs. DAGMan submits jobs to Condor in an order represented by the DAG and processes the results. Whenever a component is available, DAGMan sends it to Condor. Condor uses its matchmaking framework [RLS98] [Uni] to map the component to a resource. DAGMan schedules the workflow dynamically at each step, so that the overall map-

ping and schedule is determined by the state of the Grid at the series of points when the application components are ready to execute. My work uses performance models to take a more global approach that schedules an entire workflow in advance.

2.3.4 Meta-heuristic Approach - GridLab

Mika et al. in [MWW03] consider the problem of scheduling workflow tasks onto the Grid. They formulate the problem as a multi-mode resource-constrained project scheduling problem with schedule-dependent setup times, which is an extension of the classical NP-complete resource-constrained project scheduling problem to minimize the makespan. They present a 0-1 linear programming formulation of the problem, and propose AI-based local search meta-heuristic approach to solve the problem. The different AI meta-heuristics considered are simulated annealing, Tabu search and genetic algorithms. This work is being done in the context of the GridLab project and there are no published results on the effectiveness and efficiency of this approach.

The GridLab effort has also developed a system for workflow specification called the Triana workflow Specification [TMSW01]. Triana provides an easy-to-use graphical user interface to construct an application workflow. The user specifies the tasks, dependencies, parameters and the set of machines for each component. Once the workflow is created, it is converted to a WSFL-like XML taskgraph representation. The taskgraph is then executed by a Triana controller through a gateway Triana Service. There is no notion of scheduling in the Triana architecture. The user chooses the resources.

2.3.5 TrellisDAG

The TrellisDAG [GLS] system deals with providing a framework to support computations of non-trivial workloads with inter-job dependencies on high performance

computing platforms. The system provides novel mechanisms for users to specify both simple and complicated workflows, especially DAG description scripts (along with flat submission scripts and TrellisDAG Makefiles). They also provide mechanisms to express groups of jobs (that can be potentially pipelined) and dependencies between groups. It also provides a framework for implementing different scheduling policies. Their scheduling policies have been simple, mostly first-come-first serve (FCFS) of jobs with satisfied dependencies (like DAGMan) and simple approaches to data-locality when placing jobs on processors. The framework also provides mechanisms to monitor the progress of complicated workflows. In contrast, my work deals with global scheduling schemes for complicated workflows, which may overcome the poor makespans that may result from simple FCFS scheduling.

2.3.6 GEL - Grid Execution Language

Lian et al. [LFPA05] have designed a simple scripting language to write Grid workflow applications and have built interpretor support for the same. The interpreter builds and executes portions of the application workflow DAG on the fly (as it interprets, different application components are executed). All concerns specific to middleware are pushed to different instances of DAG executor that is responsible for actually launching the computation for the components. Iterative cyclic dependencies in the workflow are supported mostly because of the interpretive architecture. Taking care of cyclic dependencies is a novel feature of their work. Their infrastructure doesn't have any scheduling support and proposes to leverage external schedulers.

2.4 Grid Resource Management - Scheduling in Current Grid Projects

My work belongs to the general area of Grid resource management. The editors of [NSW03] give an excellent overview of the state of the art in Grid resource management - the general concerns and various projects and research works that address those concerns. Hence, I herein don't provide a survey of Grid resource management in general. However, I will try to specify how my work fits in the general area. My work specifically falls in the area of application-level scheduling and execution of Grid application workflows. So, my work has to interface with different components of the Grid resource management system - mainly the ways to access Grid resource information and their predictions [WSH99] [FFK+97] [NSW03], the ways to determine application requirements [NSW03] and application performance models [TWG+01] [CDK+04] [Mar03], the ways to interact with the Grid runtime system for secure job execution over multiple administrative domains and monitoring and ways to interact with resource level schedulers, batch schedulers etc. [NSW03].

Here is a short survey on the scheduling/resource brokering strategies used in some of the other current Grid projects. In the Condor-G [FFLT01] project, each job is submitted to Condor and the Match-making framework is invoked to match a resource to the job. The Match-making framework uses a flexible scheme of matching resources to jobs via resource and job Classified Advertisements (ClassAds) [Uni]. Jobs are ranked according to rank functions present in the job ClassAds. The job is mapped to the highest ranked resource. So, scheduling is on a job by job basis. In the DataGrid resource broker [dat], the jobs are expressed in JDL (Job Description Language), a ClassAds-type language. The broker performs matchmaking returning all resources satisfying the JDL expression. Then it uses a ranking strategy to get

the "best" match for the JDL expression. Here too, scheduling is on a job-by-job basis and very similar to what happens in Condor-G. The GrADS MPI scheduler [DCB02] is a modular, application-level scheduler for tightly coupled applications. Scheduling is based on application performance models and data and computation mapping strategies. Resource modeling is done using NWS and MDS. It uses a heuristic search method to find the best possible candidate resources for the MPI application to run on. My work is based on the same philosophy of performance model based scheduling, but for a different class of application.

Chapter 3

Grid Applications

In this chapter, we provide a summary of different Grid applications from several current Grid projects. We also present in detail three applications that are representative of many of the Grid applications. We then infer about the general class to which many of the Grid applications belong to. We come up with a generalized representation for this class of applications.

3.1 Grid Applications Survey

Applications running on the Grid belong to scientific fields as diverse as high-energy physics, astronomy, mesoscale weather modeling, bio-informatics and life sciences, earthquake modeling, image processing etc. Here are a few important ones categorized by different fields.

- *High-Energy Physics, Astrophysics*: The applications for the GriPhyN [grib], PPDG [ppd] and iVDGL [ivd] projects are mostly from the fields of high-energy physics, astrophysics and astronomy. These applications manage and process large amounts of data, run large compute and data-intensive simulations and go through a software pipeline of various stages of analysis and transformations. Here are some of these applications.

  The Compact Muon Solenoid (CMS) application [Wul98] is their flagship high-energy physics application. The CMS is a high-energy physics detector planned

for the Large Hadron Collider (LHC) at the European Center for Nuclear Research(CERN). CMS is scheduled to be completed by 2007 when it will begin to record data from the high-energy proton-proton collisions. The huge amounts of data (about several petabytes/year after filtering) from these collisions is expected to shed light on many fundamental scientific issues. For now, the CMS collaboration has been doing compute-intensive Monte Carlo simulation experiments, the output of which will be compared with the actual data once the CMS is functional. Each production run of these simulations corresponds to a Monte Carlo Production (MOP) job. Each MOP job is converted to a DAG that is submitted for Grid execution. Millions of such runs need to be completed. Each run corresponds to a CMS event that need to be processed through different stages.

The LIGO [BW99] application deals with detection and measurement of gravitational waves predicted by theories of relativity. The huge amount of raw data collected during experiments is analyzed in both time and Fourier domains in a software pipeline. In one of the specific LIGO problems for pulsar search, once the inputs (like the channel name, start time etc.) are submitted, the processing required to produce the output data could be represented as a DAG. The DAGs are then executed on the Grid.

The Cactus [ABD+01] code, a main application for the GridLab [gria] project, is used by numerical relativists to build and run large-scale simulations that model black holes, neutron stars, gravitational waves etc. with Einstein's theory of relativity. Application code developed using the Cactus toolkit uses task farming to run large parameter studies corresponding to simulations not requiring much communication. Also, a large Cactus application may be tightly-coupled and

use resources from various sites to satisfy its resource needs via use of Grid computing. Cactus also has advanced features like migration and spawning. In short, Cactus covers a wide range of Grid usage scenarios.

- *Astronomy*: The National Virtual Observatory (NVO) [nvo] initiative, often described as the Datagrid for astronomy, is aimed at developing tools that make it easy to locate, retrieve, and analyze astronomical data from archives and catalogs worldwide, and to compare theoretical models and simulations with observations. The transformation of raw instrument data (like the ones from Sloan Digital Sky Survey) into calibrated and cataloged data is a demanding, compute-intensive task. The NVO applications perform computationally-intensive, complex analysis on large, heterogeneous and distributed datasets using a computational workflow that generate catalogs of stars, galaxies and other astronomical objects.

- *Bio-informatics*: There are a host of Grid applications from several sub-fields of bio-informatics. They include applications that deal with bio-molecular simulation and analysis, distributed cell simulations and other systems biology simulations, sequence comparisons, genome analysis, comparative genomics, proteomics, drug design, medical imaging, macromolecular and neuroimaging, immunology and so on. Important initiatives include BIRN (Biomedical Informatics Research Network) [bir], EU-funded GEMSS [gem], Mammogrid [mam] and BioGrid [bio], UK myGrid [myg] , EOL [LBH+04]. The common themes of these applications are that (1) most of them involve large computationally intensive parameter studies and simulations (2) they may access and update large, distributed and diverse databases and (3) the searches may involve large data movement and replication.

- *Mesoscale Weather Modeling*: The weather modeling and forecasting community has also been running large scale weather simulation models on the Grid. Their applications also involve a computational workflow of setting up initial models and running suites of large computationally intensive simulations (many of which can be tightly-coupled codes) of different models. The simulations are run in order to do parameter studies. The simulation results are visualized and data-mined to obtain a forecast. The MEAD [mea] and the LEAD [lea] projects have been involved in this.

- *Other Fields*: There are other Grid applications from many diverse fields like earthquake engineering [FK03], distributed aircraft maintenance [FK03], parallel tomography [SCF+00], desktop grid [FK03] and many others.

3.2 Representative Applications

We now discuss three applications (EMAN, EOL and Montage), which are representative of many Grid applications.

3.2.1 EMAN - Electron Micrograph Analysis

EMAN [LBC99] is a bio-imaging application developed at the Baylor College of Medicine. It is mainly used for doing 3D-reconstruction of single particles from 2D electron micrographs. Figure 3.1 gives an overview of the functionality of EMAN. Biologists' intervention and expertise is needed to come up with the preliminary 3D model from the electron micrographs. The refinement from a preliminary model to a final 3D is fully automated. This "refinement" step is highly compute-intensive and benefits from harnessing the power of the Grid. The "refinement" step builds up a refined 3D model of the single particle from a coarser 3D model of the same and

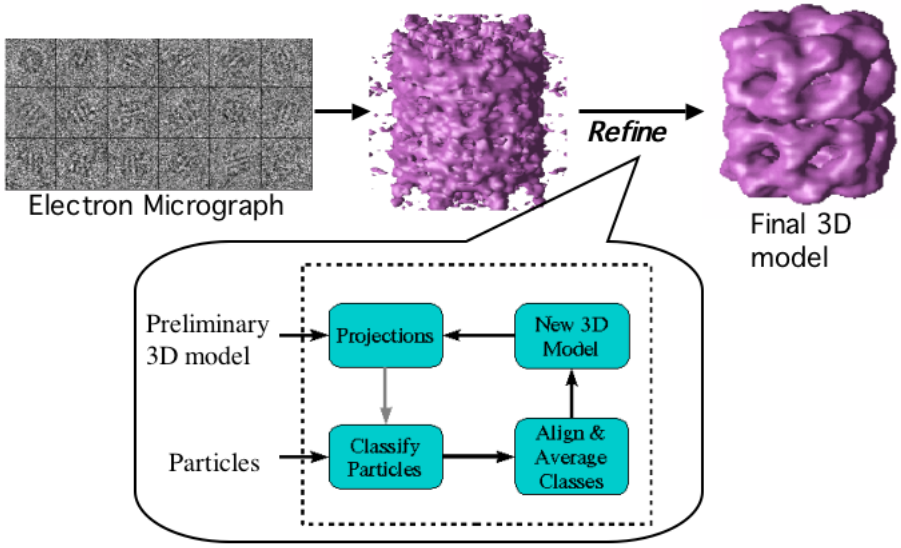

Figure 3.1 : EMAN Overview

iterates until a "good" quality 3D model is obtained. EMAN refinement is structured as follows.

One iteration of "refinement" is structured as a linear DAG. Some nodes in the DAG are inexpensive sequential programs, while the interesting computationally intensive ones are large parallel parameter sweeps. Some of the parallel programs can be run across multiple clusters while others must run on a single cluster. For multiple iterations of "refinement", the DAG is unrolled multiple times. Figure 3.2 gives a detailed view of the EMAN "refinement" computation. There are eight nodes in each "refinement" chain of computation. Of these, four are parallel nodes and rest are sequential ones. The classification step in EMAN, "classesbymra" is the most compute-intensive (and parallel) step in the "refinement" workflow.

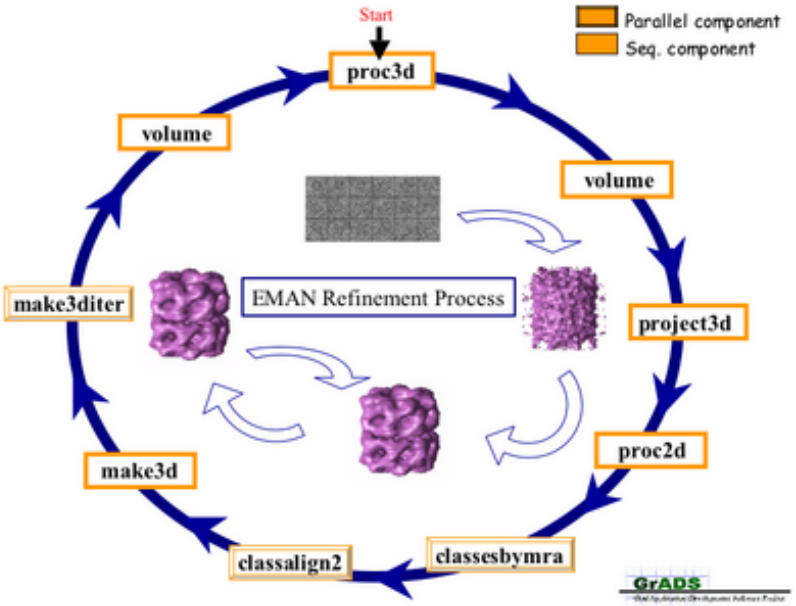

Figure 3.2 : EMAN "refinement" workflow

EMAN "refinement" is not only compute-intensive but also data-intensive. The input electron micrograph data for a moderate sized experiment can be a few GBs and for large experiments, it can be 100GB. So, having the computations near the data and minimizing intermediate data movements are important concerns when scheduling EMAN on the Grid.

The current EMAN "refinement" is representative of applications having a linear software pipeline of programs some of which are parameter sweeps. In the future, each of the EMAN computations will evolve to a DAG that may be described using high-level Python scripts [per]. Multiple such DAGs can be submitted for Grid execution.

The more general model that future EMAN application will be representative of is a mixed parallel application where there will be

- A set of large input files (replicated or distributed)

- A set of software components arranged in a DAG (a simple chain or a more complicated DAG described through a high-level scripting language), where each component accesses the input data and/or data produced from preceding components. Some of the components will be parameter sweeps themselves.
- A set of such DAGs needs to be processed, as many as possible and as fast as possible.

3.2.2 EOL - Encyclopedia of Life

The Encyclopedia of Life (EOL) [LBH+04] is a collaborative global project designed to catalog the complete genome of every living species in a flexible reference system. It is an open collaboration led by the San Diego Supercomputer Center, and currently has three major development areas: (i) creation of protein sequence annotations using the integrated genome annotation pipeline (iGAP); (ii) storage of these annotations in a data warehouse where they are integrated with other data sources; and (iii) a toolkit area that presents the data to users in the presence of useful annotation and visualization tools. The key computation-intensive step is the first one since its goal is to discover relationships across genomes, and thus involves extensive computation, access to databases that contain data derived from the iGAP processing of multiple genomes (dozens initially and ultimately hundreds). However, this coupling across genomes is achieved exclusively through accesses and updates to these shared databases.

EOL is not a single code, but rather a script (iGAP) that glues together a number of well-known community software packages. These packages operate on input files ("sequence files") as well as on community databases. EOL processes genomes that can be all processed independently and in any order. Each genome consists of a

number of sequences. These sequences can be processed independently and in any order. Each sequence is processed in a series of steps (a workflow called the iGAP "pipeline"), implemented as a set of shell scripts that call community codes one after another genomes and subsequences, as well as for the protein sequences described in the genome.

3.2.3 Montage

Montage[ea04] is a large-scale data-intensive astronomy application and comes from the NVO initiative. The Grid-enabled version of Montage is suitable for large scale processing of 2MASS images (from the NASA/IPAC Infrared Science Archive) of a large portion of the sky with the goal of providing custom, science grade image mosaics in FITS format. The application consists of the computational workflow of re-projection of input images that can be done in parallel utilizing required Grid resources, modeling of background radiation, rectification of images and co-addition of re-projected, background corrected images into a final mosaic. In short, Montage has the DAG structure shown in Figure 3.3. mProject, mDiffFit and mBackground are parallel components. There are equal number of mProject and mBackground nodes. Each mDiff node has two mProject nodes as predecessors.

3.3 Generalized Representation of Grid Applications - Workflow Application

From the above survey of the different Grid applications, we have come up with a generalized representation of a Grid Application. We represent the application as a Directed Acyclic Graph (DAG).

- *Node Semantics*: The nodes in the DAG represent sequential, parameter sweep

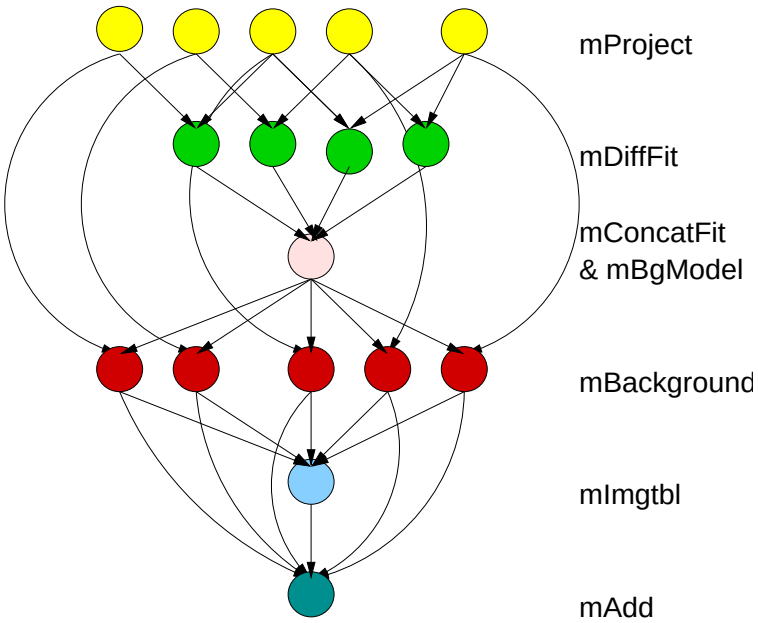

Figure 3.3 : Montage Application

or tightly-coupled/parallel computations. Each node must run to completion without interruption. The nodes save the output data to a local file-system. The nodes may access and update databases or distributed data-sets. Some of the nodes may be control nodes like nodes for data assimilation and explicit data movements.

- *Edge Semantics*: The edges represent data and control dependencies. All communication is done using file transfers over the network. The volume of communication is annotated on the edges. We don't consider pipelining of communication as a part of the thesis.

We call an application that can be represented using this format a workflow application and the DAG with all the attributes the workflow representation of the application. 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. Figure 3.4 gives an example of the representation. Most Grid applications can be instantiated using this representation. The main issues of

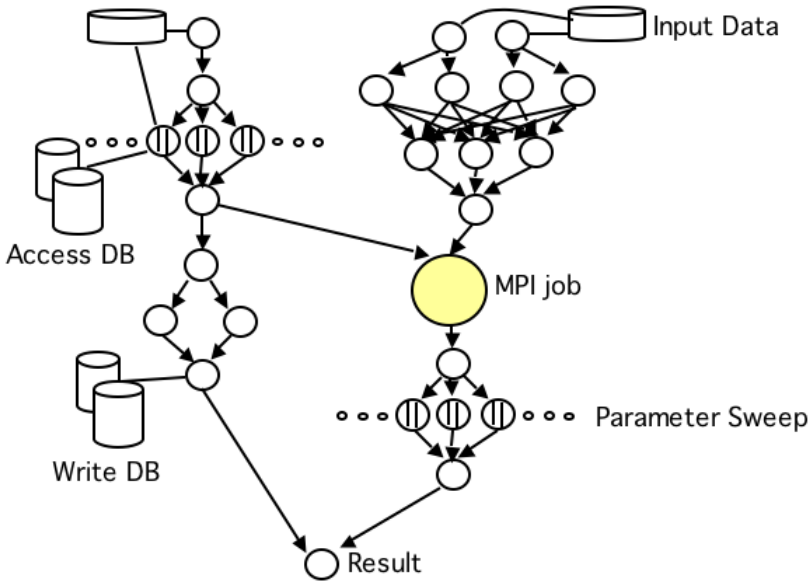

Figure 3.4 : Example workflow representation

executing workflow applications on the Grid are

- The ability for the components to seamlessly access the required Grid resources. This may be achieved by leveraging a framework like VGrADS vgES [vgr] using Globus services (authentication, remote job launch etc.).
- Efficient resource selection and scheduling for the components in order to achieve good performance. This means harnessing the Grid resources effectively for the

compute-intensive steps and minimizing communication, thereby minimizing the makespan for the whole workflow. Chapter 4, 5 and 7 provides the details of efficient workflow scheduling.

- Satisfying all dependencies and automating the Grid execution of the entire workflow. This may be achieved by leveraging workflow management frameworks (like Pegasus).

Chapter 4

Plan-Ahead Workflow Scheduling

As discussed in the last chapter, large scientific workflows arise from scientific experiments involving large scale simulations, search, analysis, mining etc. Most of these workflows are coarse-grained and hence suited for execution in Grid environments. A fundamental issue for efficient execution of the workflows on the Grid is selection and scheduling of Grid resources for workflow execution. In this chapter, we describe our approach to "plan-ahead" scheduling of large scientific workflows onto the Grid. Our scheduling strategy is based on accurate performance predictions of the workflow components. We call our strategy "plan-ahead" because we take the scheduling decisions for all the components in the workflow before executing any component of the workflow. This is in contrast to dynamic scheduling approach where scheduling decisions are taken in tandem with workflow execution.

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 replanning is required. In our workflow scheduling work, we have chosen the following

design parameters. Our objective 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) [FFK+97] and Network Weather Service (NWS) [WSH99]. We derive application component performance models using methods described in a later section. We state the scheduling problem we are trying to solve as follows.

4.1 Problem Statement

Given a DAG of the workflow representation of the application, let the set of currently available application components (for the purpose of scheduling) 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 solution to the scheduling problem is an output of a mapping and schedule of elements of C onto elements of G, or in other words, an output of which component runs on which Grid resource in what order.

4.2 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.

4.2.1 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. We assign the rank value for each resource in the following manner.

- At first we check 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, we assign it a rank value of infinity. We derive the resource properties from calls to MDS services.

- Next, we evaluate the rank values for the eligible resources. 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 $dCost(c_i, r_j)$.

$$rank(c_i, r_j) = w_1 \times eCost(c_i, r_j) + w_2 \times dCost(c_i, r_j) \tag{4.1}$$

  We can customize the weights to give more importance to one over the other. We derive $eCost(c_i, r_j)$ from application component performance model and $dCost(c_i, r_j)$ from communication performance model. We describe them in detail in the next two sections.

4.2.2 Communication Performance Modeling

We calculate $dCost(c_i, r_j)$ as follows. Let $map(c_i)$ denote the resource onto which c_i has been mapped, $vol(c_i)$ denote the volume of data produced by c_i, $Parent(c_i)$ denote the set of parent components for c_i , $Lat(r_p, r_q)$ denote the estimated latency from resource r_p to resource r_q and $BW(r_p, r_q)$ denote the estimated bandwidth between resource r_p and r_q. Then, we define $dCost(c_i, r_j)$ as

$$dCost(c_i, r_j) = \sum_{p \in Parent(c_i)} (Lat(map(p), r_j) + vol(p) \times BW(map(p), r_j)) \qquad (4.2)$$

We estimate $Lat(r_p, r_q)$ and $BW(r_p, r_q)$ from latency and 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.

4.2.3 Component Performance Modeling

To estimate the $eCost(c_i, r_j)$, we build architecture independent component performance models semi-automatically. The performance models take into account both the number of floating point operations executed and the memory access pattern for the component. [CDK+04] describes the strategy.

The rank values can now be determined using the performance models and equation (4.1). 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 ith component on the jth resource. Once we have this matrix, we can solve for the final mapping of components to resources.

4.2.4 Solving for the final mapping

The mapping problem is an NP-Complete problem since the Minimum Multiprocessor Scheduling problem is NP-Complete [GJ79] and can be reduced to our mapping problem. We use an heuristic approach to solve the mapping problem.

4.2.5 Heuristic Approach

We chose to apply three heuristics from the domain of scheduling parameter sweep applications [CLZB00, Tra01]. 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 over the entire workflow 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). maxECTparents(j) is the maximum over the estimated completion times of the parents of job,j. The overall algorithm, described in Algorithm 4.1, 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 4.2 is invoked with the current rank matrix, available components and heuristic as arguments. Algorithm 4.2 implements the core scheduling heuristics. Depending on the current heuristic, it returns a mapping for the current set of available components. Algorithm 4.1 updates the mappings and current makespan. When the outer loop of Algorithm 4.1 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.

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

Algorithm 4.1: Workflow Scheduling

Other Heuristic

We have also implemented a simple Greedy heuristic to map the available jobs. In the greedy heuristic, each available component is immediately mapped to the resource that gives the minimum ECT over all resources. The EAT vector and makespan are updated with each mapping.

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

Algorithm 4.2: findBestSchedule

4.3 Experimental Evaluation

4.3.1 Experimental Framework

Infrastructure for 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. Figure 4.1 describes the GrADS execution cycle. The application along with the performance model is handed over to the Workflow

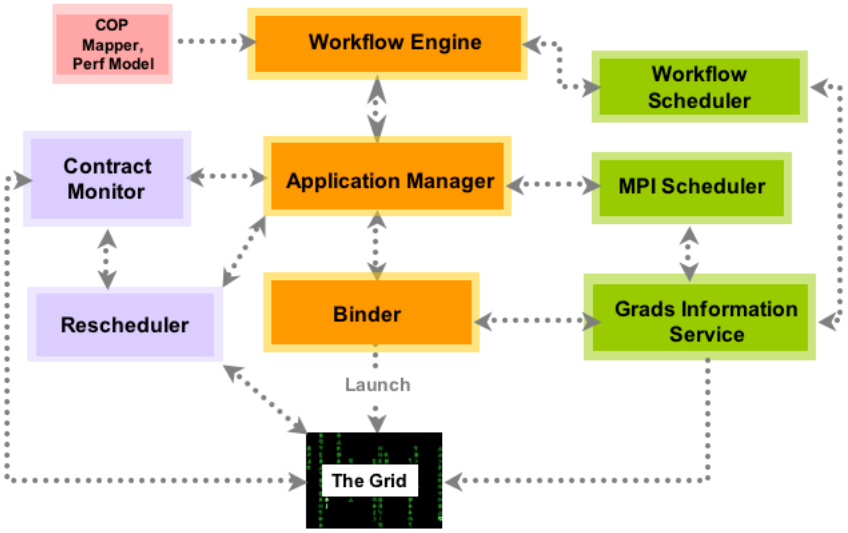

Figure 4.1 : GrADS Execution Cycle

Engine, which in combination with the workflow scheduler schedules the application components onto available resources. 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. Once the schedule is obtained, the execution system invokes the binder to execute the components on the chosen resources. The binder queries the information system to find out the application specific libraries, inserts sensors for run-time performance monitoring, configures/compiles/links the components on the target resources and actually launches the components. Heterogeneity is handled as a part of the design since the binder compiles from source at each target resource. Cooper et al. [CDK+04] describe the GrADS binder in detail.

Testbed and Data-set for Experiments

The testbed for the experiments consisted of distributed, heterogeneous resources from multiple participating institutions of the GrADS and VGrADS projects. There were 64 dual-processor 900MHz Itanium IA-64 nodes at the Rice University Terascale Cluster (RTC). There were 16 Opteron nodes (2009MHz) 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). We also used 7 Pentium IA-32 nodes from the TORC cluster at the University of Tennessee, Knoxville and 6 Itanium IA-64 nodes from the MCKINLEY cluster at University of Houston for some experiments. Note that the testbed was heterogeneous in terms of architecture, CPU speeds, memory and storage.

We used two EMAN refinement data sets 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. We used version 1.6 of the EMAN software for the

experiments along with per component performance models for this version.

4.3.2 Value of Performance Models and Heuristics

Our goal was 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, and quality of application makespan. We chose 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 four scheduling strategies were compared.

Competing Strategies

- 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. 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 chapter, but with only crude performance models to determine the rank values. Rank value of a component on a resource is just the relative value, or

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 chapter.

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

Results

Table 4.1 and Figure 4.2 show 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 we obtain a makespan improvement by a factor of 2.21.

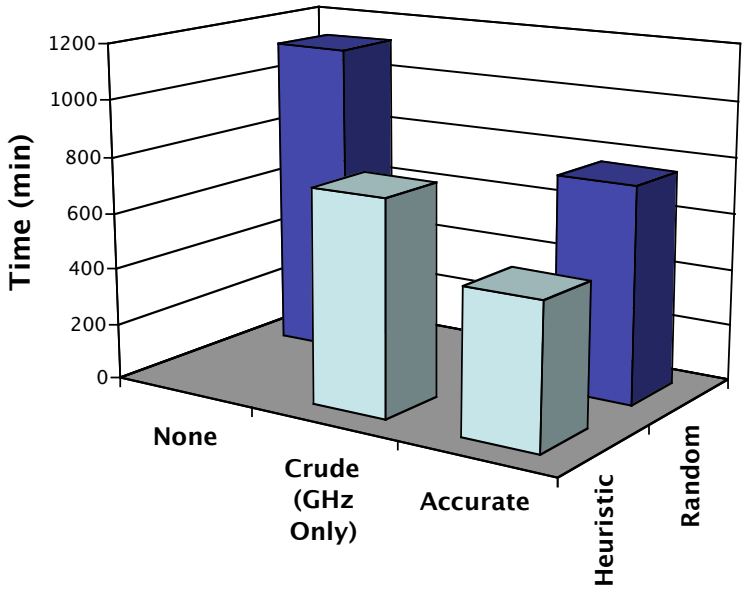

Figure 4.2 : Comparison of different scheduling strategies

Strategy	**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 4.1 : Comparison of makespans for different scheduling strategies with rdv data

Component	**i(M)**	**i(T)**	**n(M)**	**n(T)**	**t(M)**	**t(T)**
proc3d	1	0	1	0	<1min	0min
project3d	1	0	1	0	1hr. 48min	0min
proc2d	1	0	1	0	< 1min	0min
classesbymra	*68*	*42*	*6*	*7*	*84hr. 30min*	*81hr. 41min*
classalign2	379	0	6	0	45min	0min
make3d	1	0	1	0	47min	0min
proc3d	1	0	1	0	< 1min	0min

Table 4.2 : Results of EMAN workflow execution with rdv data

4.3.3 Load Balancing

Table 4.2 shows the results of the run of the rdv data on unloaded resources on the testbed consisting of the TORC (TORC1-7) and MCKINLEY (i2-53 to 58) nodes. The first column represents the name of the component in the linear DAG. The next two columns denote the number of instances mapped by the workflow scheduler to the selected clusters (i(M) for MCKINLEY and i(T) for TORC). The next two columns denote the number of nodes chosen by the workflow scheduler in each cluster (n(M) for MCKINLEY and n(T) for TORC). The last two columns denote the time it took for that component to run on each cluster (t(M) for MCKINLEY and t(T)for TORC).

For the sequential and single-cluster components, the scheduler chose the best node or cluster for execution. The interesting case is the case of the parameter sweep step called "classesbymra". From the execution time of the "classesbymra" step, we can infer the following:

- The makespan of the "classesbymra" step was 84hrs. 30 minutes (the time the

Component	**i(R)**	**i(M)**	**i(A)**	**t(R)**	**t(M)**	**t(A)**	**MS**
classesbymra	29	42	39	383min	410min	308min	410min

Table 4.3 : Load balancing of the "classesbymra" instances using the HA strategy

instances finished on the MCKINLEY cluster). Since the instances at the TORC machines finished in 81 hrs 41min, we can infer that the load was optimally balanced across the two clusters since the granularity of a single instance is greater than 7hrs.

- The optimal load balance is primarily due to accurate performance models and efficient workflow scheduling. Rank of a "classesbymra" instance on a node in MCKINLEY cluster was 5077.76 and on a node in TORC cluster was 8844.91.

Table 4.3 shows that the HA strategy achieves good load balance when "classesbymra" is launched across the three clusters - RTC, MEDUSA and ACRL. 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. Based 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. 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" 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 clusters matched closely to the relative actual execution times on the clusters. The ratio of performance model values for an instance on a RTC node and a MEDUSA node was 3.41 while

Strategy	**i(M)**	**i(T)**	**n(M)**	**n(T)**	**t(M)**	**t(T)**	**MS**
Heuristic	38	60	2	7	12min 42sec	11min 47sec	12min 42sec
Random	17	81	2	7	6min 3sec	15min 48sec	15min 48sec

Table 4.4 : Results for groel data with unloaded resources

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 obtained similar results from our experiments with the smaller groel data set. In these tests, we compared the makespan for the "classesbymra" step for heuristic scheduling with that obtained from random scheduling. Random scheduling picks a node randomly for the next available instance. Table 4.4 shows the results. The first column represents the scheduling strategy used. The next two columns denote the number of instances mapped by the workflow scheduler to the selected clusters (i(M) for MCKINLEY and i(T) for TORC). The next two columns denote the number of nodes chosen by the workflow scheduler in each cluster (n(M) for MCKINLEY and n(T) for TORC). The next two columns denote the time it took for that component to run on each cluster (t(M) for MCKINLEY and t(T)for TORC). The last column denotes the overall makespan (MS). From these results, we can conclude that accurate relative performance models on heterogeneous platforms combined with heuristic scheduling result in good load balance of the classesbymra instances when the grid resources are unloaded. Heuristic scheduling is better than random scheduling by 25 percent in terms of makespan length for this data set.

Strategy	i(M)	i(T)	n(M)	n(T)	t(M)	t(T)	MS
Heuristic	60	38	5	7	16min 41sec	7min 51sec	16min 41sec
Random	44	54	5	7	9min 38sec	10min 28sec	10min 28sec

Table 4.5 : Results for groel data with loaded resources

4.3.4 Resource Loads and Inaccurate Performance Models

The following set of results show the effect of loaded machines on the quality of schedule. We used 5 loaded nodes from the MCKINLEY cluster and 7 unloaded nodes from the TORC cluster for these experiments. From the results in Table 4.5, we observe that there is uneven load balance due to loading of the MCKINLEY nodes. Random scheduling does better because the random distribution maps more instances to the unloaded TORC cluster which had more nodes in the universe of resources. So, we can infer that for performance model based scheduling to work, either the underlying set of resources should be reliable (implying advanced reservation) or the variability of resource performance can be predicted and taken into account during scheduling.

The third set of results show the effect of inaccurate performance models on the quality of schedule. We have used a rank value of 4.57 instead of 7.60 for a classes-bymra instance on a MCKINLEY node. We kept the TORC rank value correct. We used 6 nodes from the MCKINLEY cluster and 7 nodes from the TORC cluster. From the results in Table 4.6, we can infer that, inaccurate relative performance models on different heterogeneous platforms result in poor load balance of the classesbymra instances.

Strategy	i(M)	i(T)	n(M)	n(T)	t(M)	t(T)	MS
Heuristic	77	21	6	7	21min 37sec	3min 57sec	21min 37sec
Random	45	53	6	7	5min 24sec	10min 30sec	10min 30sec

Table 4.6 : Results for groel data with inaccurate performance models

4.4 Scalability of Workflow Scheduler

4.4.1 Theoretical Complexity

Let C denote the total number of components in the input workflow DAG, R denote the total number of available resources and E denote the number of edges (dependencies) in the workflow DAG. Topologically sorting the DAG takes $O(C+E)$ time, which is $O(C^2)$ time since E is $O(C^2)$. Calculating the rank matrices takes $O(CR)$ time. Running time for findBestSchedule() method is at most $CR+(C-1)R+(C-2)R+...+R$, which is $O(C^2R)$ when one of min-min, max-min or sufferage heuristic is used. When the greedy heuristic is used, the running time for the findBestSchedule() method is $O(CR)$. Hence, the running time for the entire heuristic workflow scheduling algorithm is dominated by the complexity of the findBestSchedule() method, which is $O(C^2R)$ for the heuristic case and $O(CR)$ for the greedy case.

4.4.2 Experimental Evaluation of Scaling of Plan-Ahead Scheduler

In order to evaluate the scalability of the Plan-ahead scheduling approaches, we plotted the scheduling times for the heuristic and greedy plan-ahead workflow schedulers for increasing number of available resources. Scheduling time is defined as the time it takes to compute the schedule for the entire workflow in-advance.

We used the EMAN and the Montage application workflows as example workflow

application scenarios. We used three versions of the EMAN workflow and three versions of the Montage workflow. We obtained the different versions of each application workflow by varying the number of parallel components at the parameter-sweep steps of each workflow. For EMAN we used the 55-110, 110-379 and 220-440 versions, where 55-110 version stands for 55 parallel components in the first parameter sweep step and 110 parallel components in the second parameter sweep step. For Montage we used the 24-51-24, 50-121-50 and 153-407-153 versions. We used the performance model values for per component estimates of execution times. We also generated resource and topology for varying number of resources, from about 250 resources to about 36k resources.

Figure 4.3 and Figure 4.4 show the results for scheduler scaling for EMAN and Montage workflows respectively. The X-axis denotes the number of available resources and Y-axis denotes the scheduling time in seconds for the particular scheduling algorithm. The scaling results show that the heuristic scheduler can take considerable time to schedule for large resource sets. Also, the greedy scheduler is cheaper than the heuristic scheduler because the complexity of findBestSchedule for the greedy case is $O(CR)$, in contrast to $O(C^2R)$ for the heuristic case.

4.5 Scheduler Applications: Incorporating Batch Queue Wait Times

Many large scale Grid systems have batch queue front ends. This means that unlike in case of interactive resources, which are available instantly, a job when submitted to such Grid systems may have to wait in the batch queue, potentially for a long time, before it can begin execution. So, the scheduler needs to be modified when the assumption of instant resource availability no longer holds. The plan-ahead scheduler

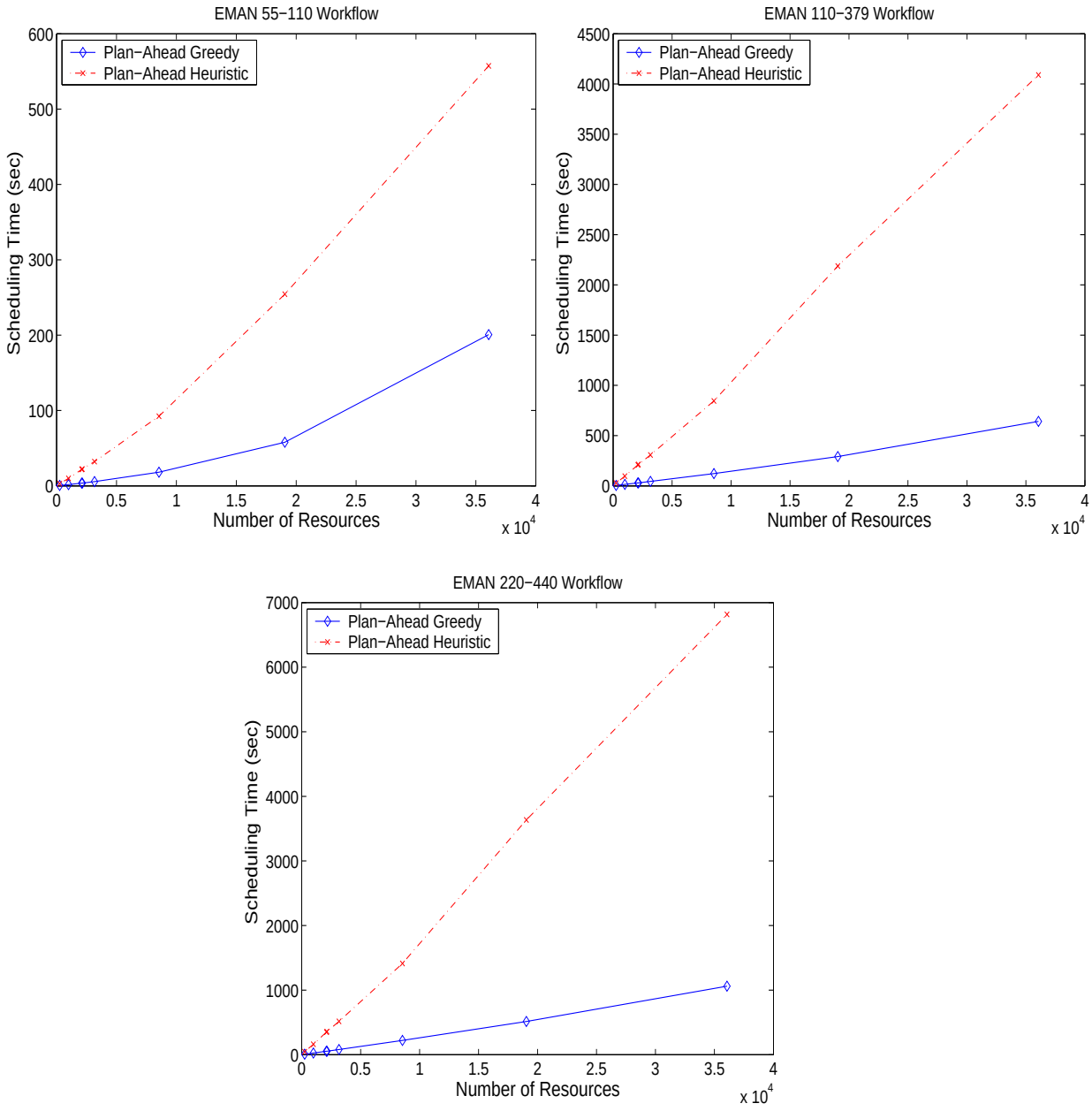

Figure 4.3 : Scheduler Scaling for EMAN

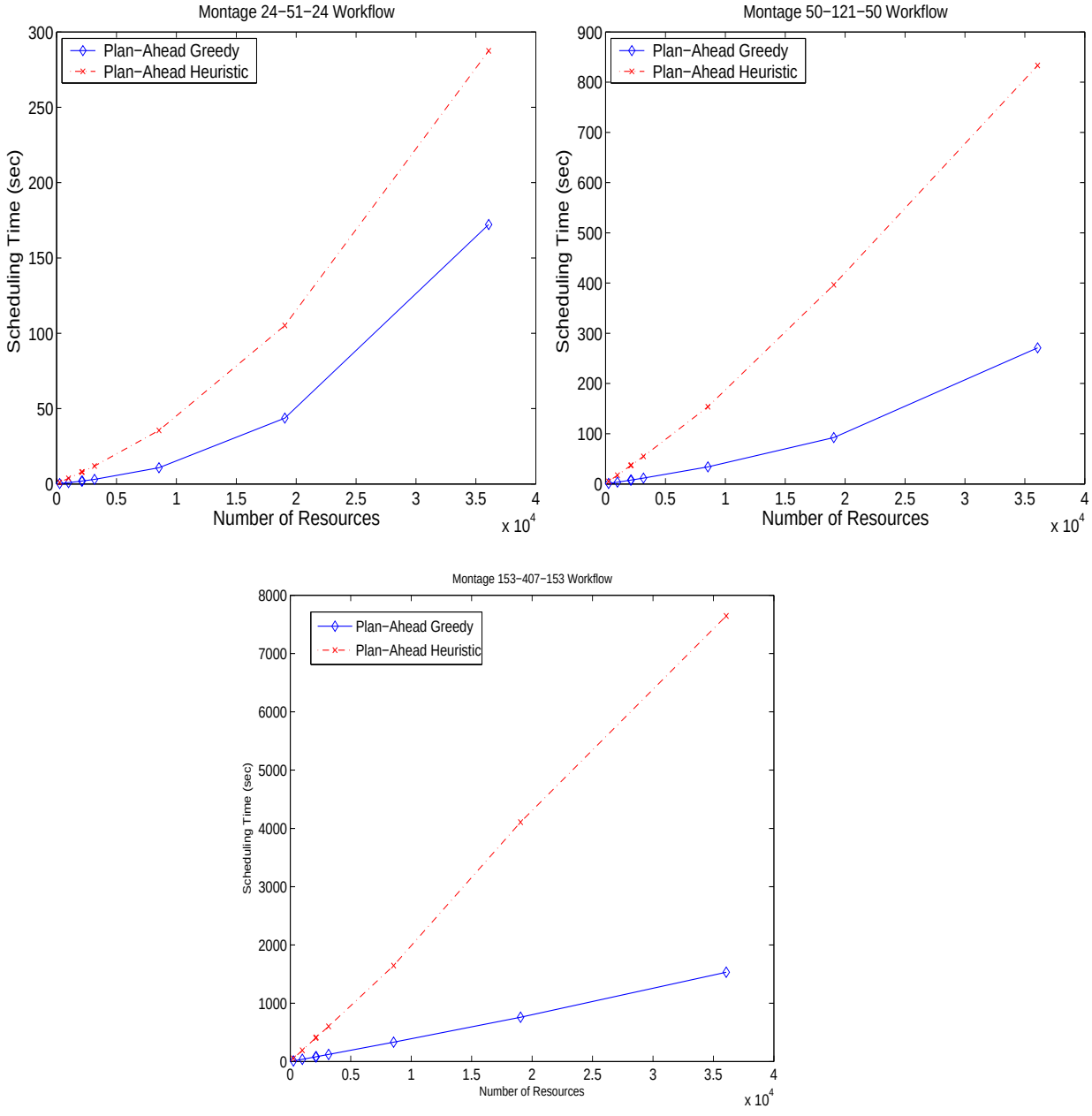

Figure 4.4 : Scheduler Scaling for Montage

as discussed in the previous sections assumes that all the resources are simultaneously available for scheduling.

4.5.1 Workflow Scheduler Modifications

To tackle this issue of batch queues, we have incorporated point value predictions for batch queue wait times in the scheduling smarts of the plan-ahead scheduler. We obtain the batch queue prediction estimates from tools developed by Dan Nurmi and Rich Wolski at UCSB. Brevik, Nurmi and Wolski[BNW05] describe the technology to obtain these estimates.

We introduced a slight modification in the plan-ahead scheduler to incorporate the batch queue wait times. At every scheduling step, we add the predicted time the job has to wait in the queue to the estimated completion time for the job, ECT(j,r) in Algorithm 4.2. We keep track of the queue wait time for each cluster and the number of nodes that correspond to the queue wait time. With each mapping, we update the estimated availability time for a resource (EAT(R) in Algorithm 4.2), with the queue wait time. Once a specified number of nodes have been acquired from the batch queue system, there is no need to wait for those nodes in the future.

The following example illustrates how batch queue wait times are incorporated in the scheduling process for a toy workflow and resource set.

Example: Workflow Scheduling with Batch Queues

Suppose we have an input workflow DAG as in Figure 4.5 and two clusters, Cluster 0 and Cluster 1 to schedule the DAG onto. Assume the batch queue wait time for Cluster 0 is 30 time units and the number of nodes to obtain this wait time is 1. The batch queue wait time for Cluster 1 is 10 time units and the number of nodes to obtain this wait time is 2. Figure 4.5 shows the state where the first level jobs in

the DAG have been mapped. The yellow bar denotes the waiting times on the queue. Also, the performance matrix is shown. The bars corresponding to the jobs include both computation and communication cost from parents.

To map the job at level 2, we find the ECT of that job for all the resources and Figure 4.6 shows that we have to incur a wait time if we want to use resources in Cluster 0. But it so happens that the job can run 5 times faster on Cluster 0 (see corresponding performance matrix). So, even with a larger wait time, the scheduler maps the level 2 job to R0.

For the third level job, we need to evaluate the ECT for that job on all resources. It turns out that, since the number of nodes to attain the 30 unit wait time for Cluster 0 is 1, we have to incur a wait time to access R1 in Cluster 0. Since, we have already acquired the two nodes in Cluster 1 and 1 node in Cluster 0, we no longer need to wait for them. Also the third level job is 1.5 times faster in Cluster 0 than in Cluster 1 (see performance matrix). So, the resulting schedule chosen by the scheduler is shown in Figure 4.8.

An evaluation of application scheduling using batch queue wait times is presented by Nurmi et al. in [NBW$^+$].

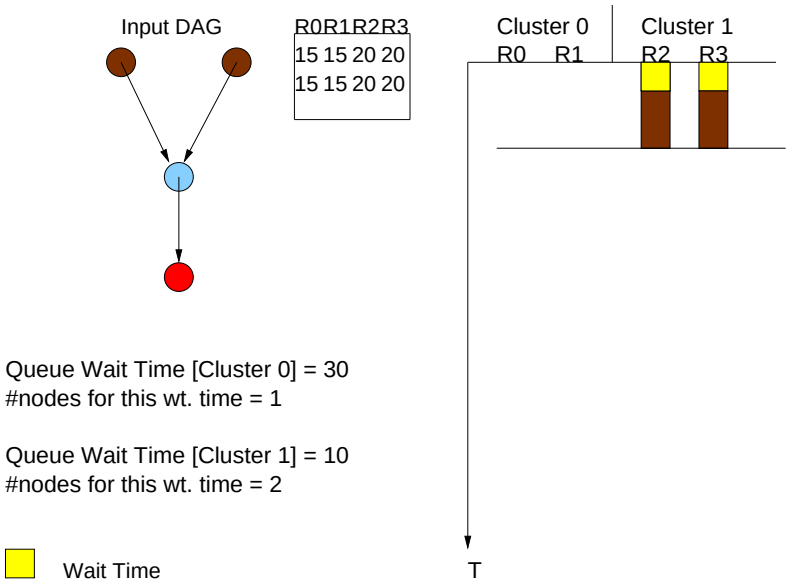

Figure 4.5 : Example: Workflow Scheduling with Batch Queues - Step 1

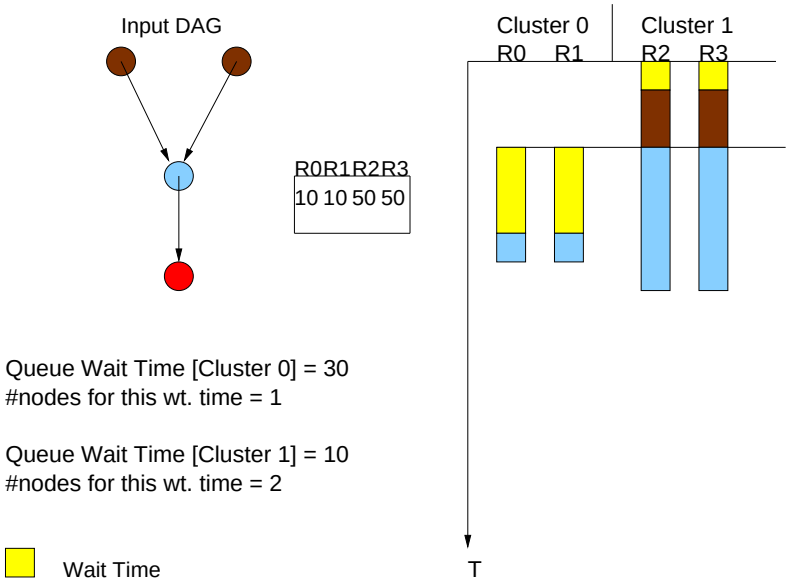

Figure 4.6 : Example: Workflow Scheduling with Batch Queues - Step 2

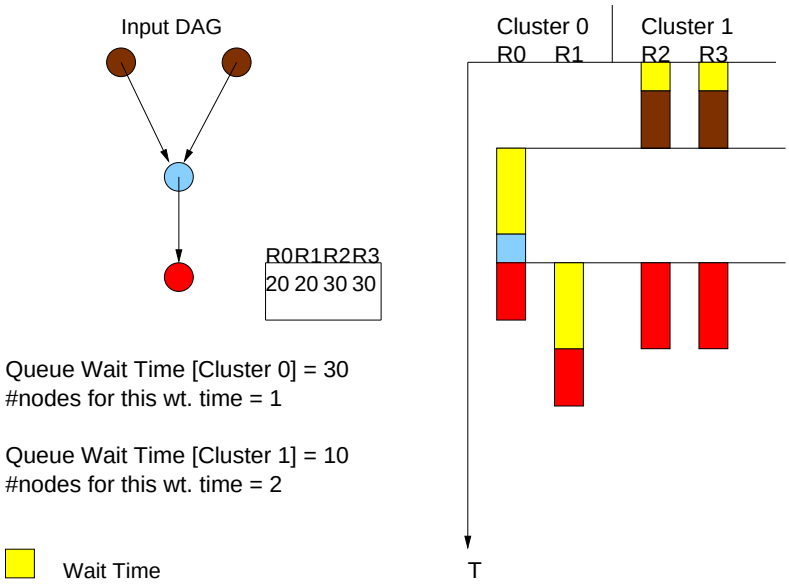

Figure 4.7 : Example: Workflow Scheduling with Batch Queues - Step 3

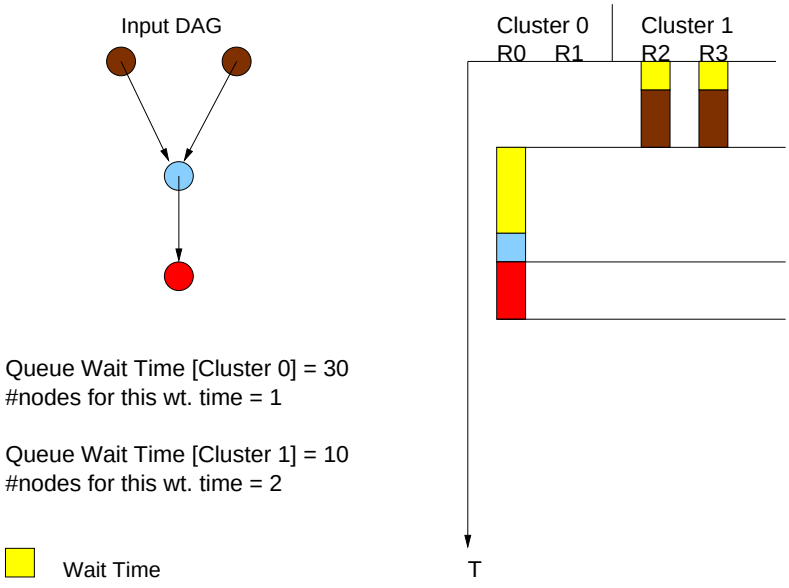

Figure 4.8 : Example: Workflow Scheduling with Batch Queues - Step 4

Chapter 5

Middle-Out Workflow Scheduling

5.1 Drawbacks of Basic Plan-Ahead Workflow Scheduling

Although, the workflow scheduling algorithm we present in the previous chapter achieves good load balance for the computationally intensive components in the workflow, it has a drawback: the traversal of the workflow DAG is from top to bottom, irrespective of what the critical steps in the DAG are. Since components are scheduled as they become available in the workflow, there is a chance that key computationally intensive steps that come later in the workflow may not be mapped to their best resource because of prohibitively high data movement costs. In other words, the solution obtained may be stuck in one of the local minima, implying that a scheduling decision taken toward the beginning of the workflow affects the scheduling decisions for components that appear later in the workflow. Our experience shows that, in many scientific applications, the critical steps are executed only after the data "massaging" steps in the workflow are done. Hence it is highly likely that the critical step may end up "stuck" at a sub-optimal compute resource.

In this chapter, we propose a variant of the plan-ahead workflow scheduler that overcomes this drawback.

5.2 Middle-Out Approach

In order to tackle the "myopia" in the top-down plan-ahead scheduler, we propose a middle-out approach to scheduling workflows, which traverses the workflow DAG in a middle-out manner.

5.2.1 Intuition

The intuition behind the middle-out traversal is as follows. Some application components in the DAG are more critical than the other steps. The critical steps may be specified by the scientist as a part of the input workflow specification or may be inferred from the performance models of the components. The idea is to map these critical components in the workflow DAG first. Once we map the critical steps, we use the mapping to assign resources to the the rest of the components, first by traversing the DAG from the key level in the DAG to the root-level in the DAG and then traversing from key-level in the DAG to the leaf-level of the DAG. The algorithm assumes that the key compute-intensive steps are in the same level of the workflow DAG. We describe the details of the algorithm in the following section.

5.2.2 Algorithm

The main steps in the proposed global workflow scheduling algorithm are (1) mapping the key computational components (2) percolating the mappings for the rest of the components by a bottom-up and top-down sweep of the DAG. The steps of the algorithm are as follows.

1. Identify the set of critical nodes, CN in the DAG - the key computational components. These can be either specified by the scientist or the application writer/scientist or can be obtained from the performance model of the compo-

nents. This algorithm works for the case where all the critical nodes are at the same level in the workflow DAG.

2. Topologically sort the workflow DAG.

3. Heuristically schedule the critical level in the DAG.

4. Traverse the DAG from (keyLevel-1) to rootLevel. For each level, estimate data-movement costs to children. Heuristically schedule each level.

5. Traverse the DAG from (keyLevel+1) to leafLevel. For each level, estimate data-movement costs from parents. Heuristically schedule each level.

6. Save the schedule for the current heuristic.

7. Repeat steps 3 through 6 for each heuristic.

8. Calculate the makespan for each heuristic and output mapping that obtains the minimum estimated makespan.

Algorithm 5.1 is the driver for the Middle-Out algorithm. findBestSchedule() method is the same as in Algorithm 4.2 in chapter 4.

5.2.3 Theoretical Complexity

The theoretical complexity in terms of number of components, C of the workflow DAG and number of available resources, R is as follows. If E is the number of edges in the workflow DAG, it takes $O(C + E) \leq O(C^2)$ to topologically sort the DAG. Calculating the rank matrix takes $O(CR)$ time. So, the running time of the Middle-Out algorithm is dominated by the complexity of the findBestSchedule method, which is $O(C^2R)$ when one of min-min, max-min or sufferage heuristic is used as shown in

```
Topologically sort the DAG;
foreach heuristic do
    availComponents = components at the keyLevel;
    Calculate rank matrix using computational perf model;
    findBestSchedule(availComponents, heuristic);
    foreach level, l from keyLevel-1 to rootLevel do
        availComponents = components at the l Level;
        Calculate rank matrix; // Use data-Cost to children
        findBestSchedule(availComponents, heuristic);
    endforeach
    foreach level, l from keyLevel+1 to leafLevel do
        availComponents = components at the l Level;
        Calculate rank matrix; // Use data-Cost from parents
        findBestSchedule(availComponents, heuristic);
    endforeach
endforeach
Select mapping with minimum makespan among all heuristics;
Output selected mapping;
```

Algorithm 5.1: Middle-Out workflow scheduling

chapter 4. When greedy heuristic is used, the complexity of findbestSchedule method is $O(CR)$. This implies that the overall complexity of Middle-out algorithm for the greedy case is $O(CR) + O(C^2)$. If the number of resources is larger than the number of components, the complexity for the greedy case becomes $O(CR)$.

5.3 Experimental Evaluation of Middle-Out Approach

We evaluate our Middle-Out approach by comparing it with the top-down workflow scheduler using EMAN workflows for test cases.

5.3.1 Simulation Framework

We created different resource scenarios using 2 Opteron clusters each having 64 nodes and 3 Itanium clusters having 64, 64 and 32 nodes. In one case, one of the Itanium clusters was faster than the fastest Opteron cluster. We refer to this case as the "Fast Itanium" case. In another case, the fastest Itanium cluster was slower than the fastest Opteron cluster. We refer to this case as the "Slow Itanium" case. Here, faster or slower is in terms of MHz rating of the nodes in the clusters.

As far as network connectivity is concerned, we generated three scenarios. In one case, the network connectivity between the 2 Opteron clusters was better than the average network connectivity between the rest of the clusters. We refer to this scenario as the "Opt close" case. In the second case, the network connectivity between the two Opteron clusters was similar to the average network connectivity between the rest of the clusters. We refer to this scenario as the "Opt normal" case. In the third case, the network connectivity between the 2 Opteron clusters was worse than the average network connectivity between the rest of the clusters. We refer to this scenario as the "Opt far" case. The network connectivity is in terms of bandwidth and latency

between the clusters.

So, for each workflow DAG, we had 6 resource scenarios resulting from all combinations of network connectivities and speed of the Itaniums.

5.3.2 Benchmark DAGs

For each resource scenario, we ran the middle-out and top-down schedulers for three versions of the EMAN workflow DAG that had 110 components for the first parameter sweep level and 379 components for the second parameter sweep step. The three versions correspond to three Communication-to-Computation Ratios (CCR): 0.1 for the compute-intensive case, 10 for the data-intensive case and 1 for the case where we have the same average computation time and communication time. We constructed the three versions using the resource scenarios and performance models.

5.3.3 Results

Figure 5.1 shows the results for the case where CCR=0.1, the compute-intensive case. The X-axis denotes values of three types of Opteron connectivity and the Y-axis denotes the simulated makespan. The left sub-figure is the "Fast Itanium" case while the right sub-figure is the "Slow Itanium" case. For each connectivity, the left bar denotes the makespan with the middle-out scheduler and the right-bar denotes the makespan obtained with the top-down scheduler. The results show that in the "Fast Itanium" case, the top-down scheduler "got stuck" at the Itanium clusters, while the middle-out scheduler scheduled the key step on both the Opteron clusters. Incidentally, for EMAN, the Opteron nodes are about three times faster than Itanium nodes for the compute-intensive step, while the ratio is less skewed for the preliminary steps. The top-down scheduler schedules the higher levels at the Itanium cluster and fails to move the computation to the Opteron clusters (which are way faster for

the compute-intensive step) because of communication costs. However, with worse Opteron connectivities, the gain due to middle-out scheduling disappears. In the "Slow Itanium" case, gain from middle-out scheduling is not significant because the top-down scheduler "got lucky" as it scheduled the higher levels to the Opteron clusters, which were also the best for the key step.

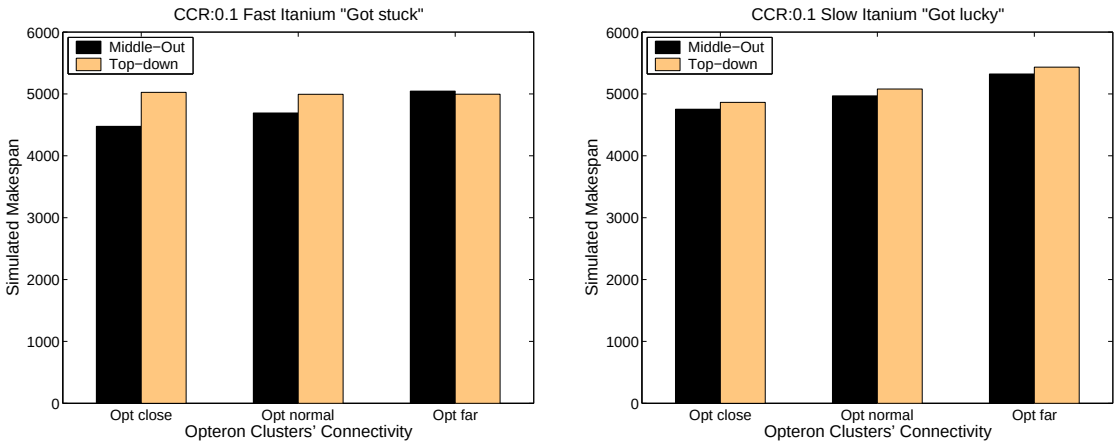

Figure 5.1 : Middle-Out vs. Top-Down: CCR 0.1

Figure 5.2 shows the results for the case where CCR=1. The results are similar to the case of CCR=0.1 for the "Fast Itanium" case. For the "Slow Itanium" case, the top-down scheduler "got lucky" and the middle-out scheduler paid more data movement costs with worse Opteron connectivities.

Figure 5.3 shows the results for the case where CCR=10, the communication-intensive case. The results are similar to the case of CCR=1. Because of higher communication costs due to a high value of CCR, the difference in makespans get amplified.

The performance of the middle-out algorithm is sensitive to the characteristics of workflow and the underlying resource characteristics. Using the middle-out approach doesn't always guarantee a better makespan.

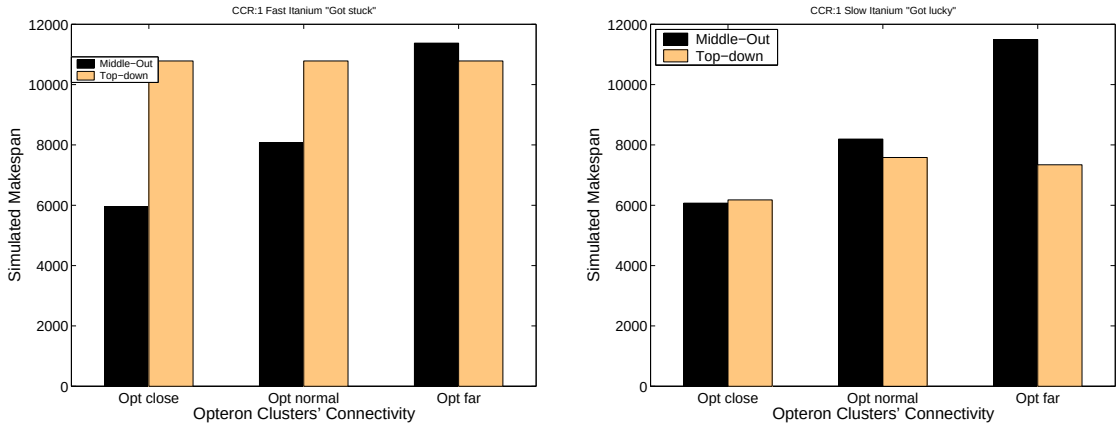

Figure 5.2 : Middle-Out vs. Top-Down: CCR 1

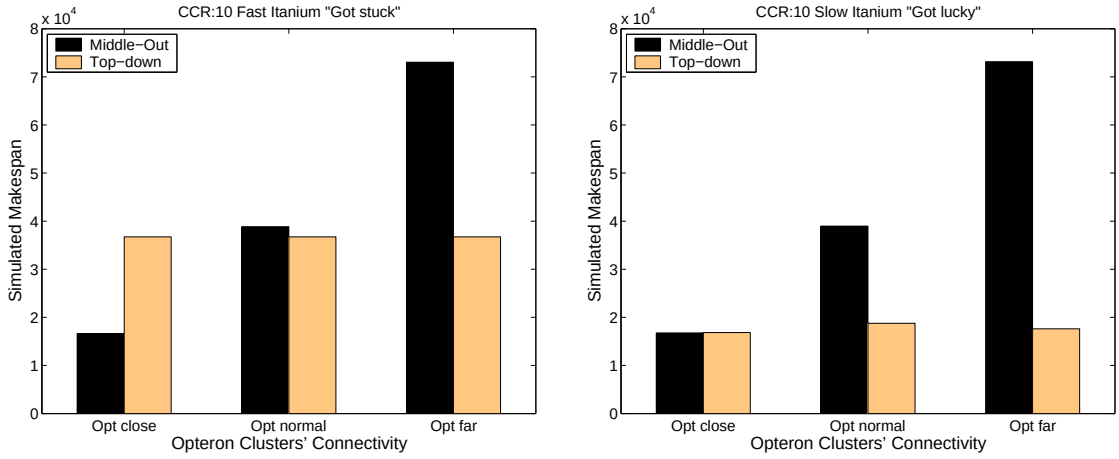

Figure 5.3 : Middle-Out vs. Top-Down: CCR 10

Chapter 6

Comparison with Dynamic Workflow Scheduling Approach

In this chapter, we compare plan-ahead, whole workflow scheduling (like the one described in chapter 4) with dynamic, task-based workflow scheduling. The plan-ahead strategy schedules the entire workflow in-advance while the dynamic strategy schedules the jobs locally in the workflow only when they are available for execution.

6.1 Dynamic/Online vs Static Scheduling Strategies

6.1.1 Online "Task-Based" Algorithm

Online, task-based approach algorithms (TBA) only reason about the tasks or jobs that are ready to run at any given instant. These algorithms make local decisions about which job to send to which resource. In these algorithms, scheduling events occur at various points during the execution of the workflow. At every scheduling event, the available jobs are scheduled and then the scheduler waits till those jobs finish and a new set of jobs are ready to be scheduled and executed. For example, consider the workflow in Figure 6.1, 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.

For every set of available jobs, we apply the widely-used min-min heuristic [Tra01] to study task-based approaches. Min-min runs in polynomial time but produces

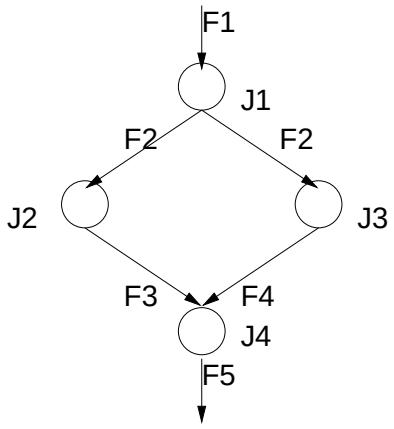

Figure 6.1 : Example workflow for task-based algorithm

efficient schedules and has been evaluated for many different situations involving independent/non-communicating tasks. However, it does not guarantee an optimal mapping. We define some terms that will be used to describe the algorithm. We assume that the estimation of execution time of a particular job on a particular resource is accurate. 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.

$$ECT(j,r) = EET(j,r) + max(EAT(j,r), FAT(j,r)) \tag{6.1}$$

At every scheduling instant, we use the the min-min heuristic (that uses the ECT attribute described) to schedule the ready jobs (ones with dependencies satisfied) and execute them on the chosen resources. We do this until all jobs in the workflow are done. Algorithm 6.1 summarizes the online task-based algorithm.

6.1.2 Static "Workflow-Based" Algorithm

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. 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 considers available jobs. The workflow-based algorithm described in the next section is a slight modification of the plan-ahead strategy described in chapter 4. This algorithm uses a randomized min-min heuristic instead of the vanilla min-min heuristic for scheduling jobs at the current level. Also, since randomization is used, the final schedule for the entire workflow is obtained by iteratively scheduling the entire workflow a number of times and then choosing the schedule that gives minimum makespan.

Randomized min-min

The randomized min-min is a local search algorithm for workflow allocation based on generalized GRASP procedure (Greedy randomized adaptive search) [RR02], which

```
while all Jobs not finished EXECUTION do
    Find availJobs = jobs with every parent finished;
    Schedule(availJobs);
endwhile

procedure Schedule(availJobs)

while all availJobs not SCHEDULED do
    foreach job, j do
        foreach resource, r do
            Calculate ECT(j,r);
        endforeach
        Find min(ECT(j,r) over all R;
    endforeach
    Find min(min(ECT(j,r)) over all J;
    Schedule and run j;
    Update EAT(j,r);
endwhile
```

Algorithm 6.1: Dynamic, task-based workflow scheduling algorithm

has been shown to be effective for job-shop scheduling [fJSS01]. In this approach, we execute the algorithm a number of times to find a good 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, the algorithm constructs a schedule. Each run of the 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, $0 \leq \alpha \leq 1$ 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 $\alpha = 0$, each iteration of the algorithm is identical to the plan-ahead algorithm using min-min described in chapter 4. When $\alpha = 1$, each iteration is random. The algorithm for our workflow-based approach is shown in Algorithm 6.2.

6.2 Experimental Evaluation

We compared Algorithm 6.1 and 6.2. We picked Montage as the example application. Chapter 3 has a description of the Montage application workflow DAG.

6.2.1 Simulation Framework

We investigated the performance of the two approaches using a Grid simulator built over the popular network simulator NS-2 [ns2]. 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 connect-

repeat

concreteWF = CreateMapping(workflow);

if *concreteWF has less makespan than bestConcreteWF* **then**

bestConcreteWF = concreteWF;

endif

until *until time limit is reached*;

`procedure CreateMapping(workflow)`

while *all jobs in workflow not SCHEDULED* **do**

Find availJobs = unmapped jobs with every parent mapped;

Map(availJobs);

endwhile

`procedure Map(availJobs)`

while *all availJobs not SCHEDULED* **do**

foreach *job, j* **do**

foreach *resource, r* **do**

Calculate ECT(j,r);

endforeach

Calculate min(ECT(j,r));

endforeach

I-min = min(min(ECT(j,r)); I-max = max(min(ECT(j,r));

availPairs = all pairs (j', r') such that

makespan increase $\leq$ I-min + α(I-max - I-min);

(j*,r*) = random choice from availPairs;

map(j*,r*); Update EAT(j*,r*);

endwhile

Algorithm 6.2: Randomized min-min, "workflow-based" algorithm

ing them, etc. Figure 6.2 shows the basic architecture of the simulator. We model resources as sites containing one or more hosts, where jobs are executed, and storage objects, where data and 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. We model the hosts with first-come-first-serve queues. 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, we used different scheduling strategies 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 chapter, with each site having a single host and storage unit. The hosts can have the same or different computational speeds resulting in a homogeneous 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. 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.

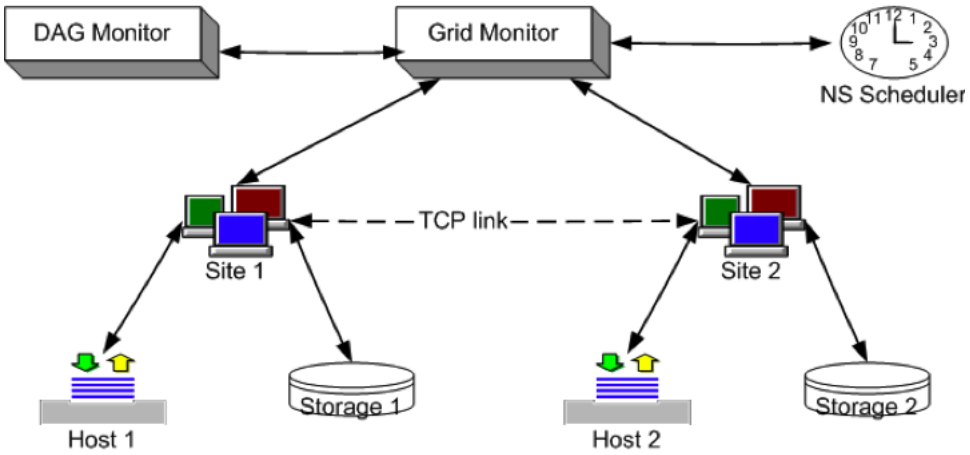

Figure 6.2 : Architecture of Grid Simulator

6.2.2 Experimental Design

We conducted several experiments in order to compare the two scheduling approaches using workflows drawn from the Montage astronomy application [ea04]. These workflows have a basic structure shown in chapter 3, but are varied in the number of jobs at each horizontal level. We assigned the job compute times at a level using a distribution for given mean and a variance of 10%. We varied the mean for each level. We modeled all files produced by jobs at same level to have the same size. We conducted all the experiments on the Grid simulator and with the host topology described in the previous section. 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 multiplied 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.

6.2.3 Results

We compared the makespan of the allocation found using the task-based approach of Algorithm 6.1 with that found using the workflow-based approach of Algorithm 6.2 and a random allocation for a workflow with 1185 job Montage workflow for (a) homogeneous and (b) heterogeneous resource scenarios.

Figure 6.3 compares the makespan of TBA and WBA for homogeneous and heterogeneous platforms and data-intensive case. Figure 6.4 compares the same for the compute-intensive case. The results show that WBA produces schedules with lower makespan for the data-intensive cases compared to TBA. The ratio of makespan of TBA to the makespan of WBA ranges from 1.47 to 2.04 for the data-intensive cases. There is almost no difference for compute-intensive cases. We also compared the performance of these algorithms for smaller Montage workflows with 57 and 739 jobs with similar results. One of the most important reasons for the difference in perfor-

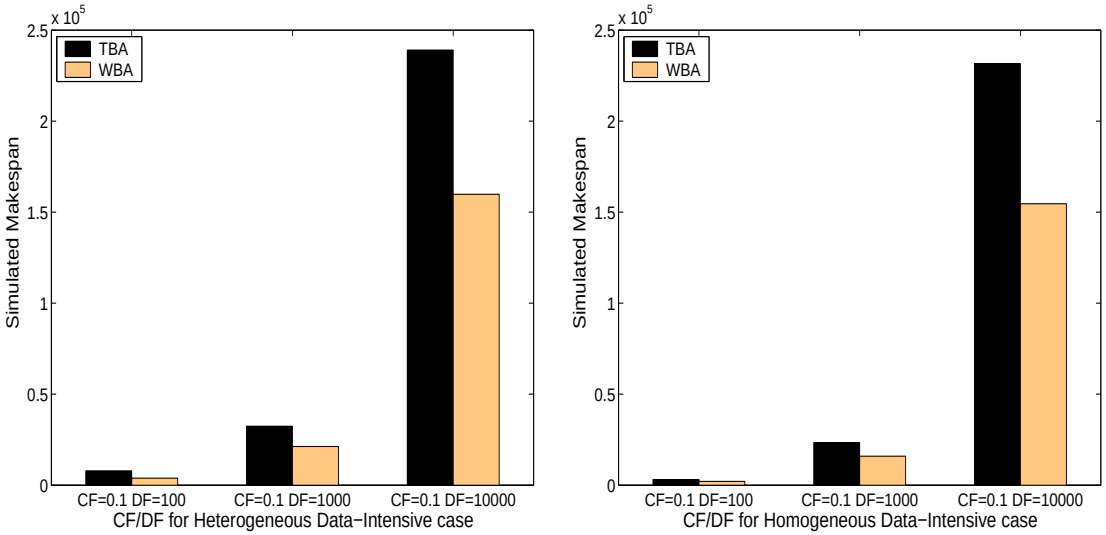

Figure 6.3 : Makespan of TBA vs WBA for data-intensive scenario

mance 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.

Figure 6.5 compares the makespan of random, WBA and TBA for homogeneous and heterogeneous platforms for the data-intensive scenario. The results show that both the workflow and task-based approaches outperform a random allocation algorithm by a factor ranging from 6 to 11. Figure 6.6 shows that 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.

The effect of uncertain estimates on quality of schedule and a new local selection heuristic are presented Blythe et al. [BJD+05].

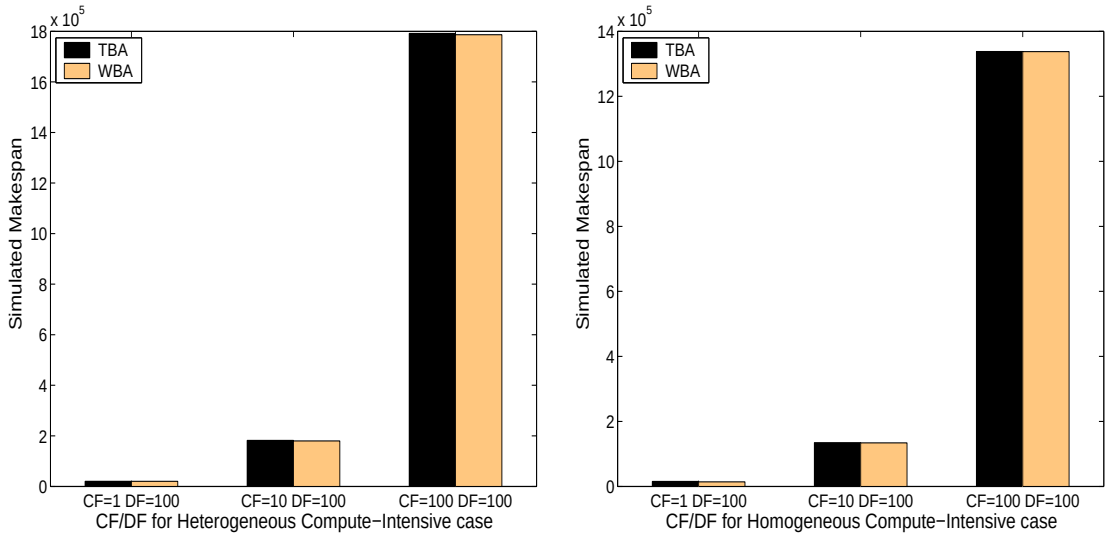

Figure 6.4 : Makespan of TBA vs WBA for compute-intensive scenario

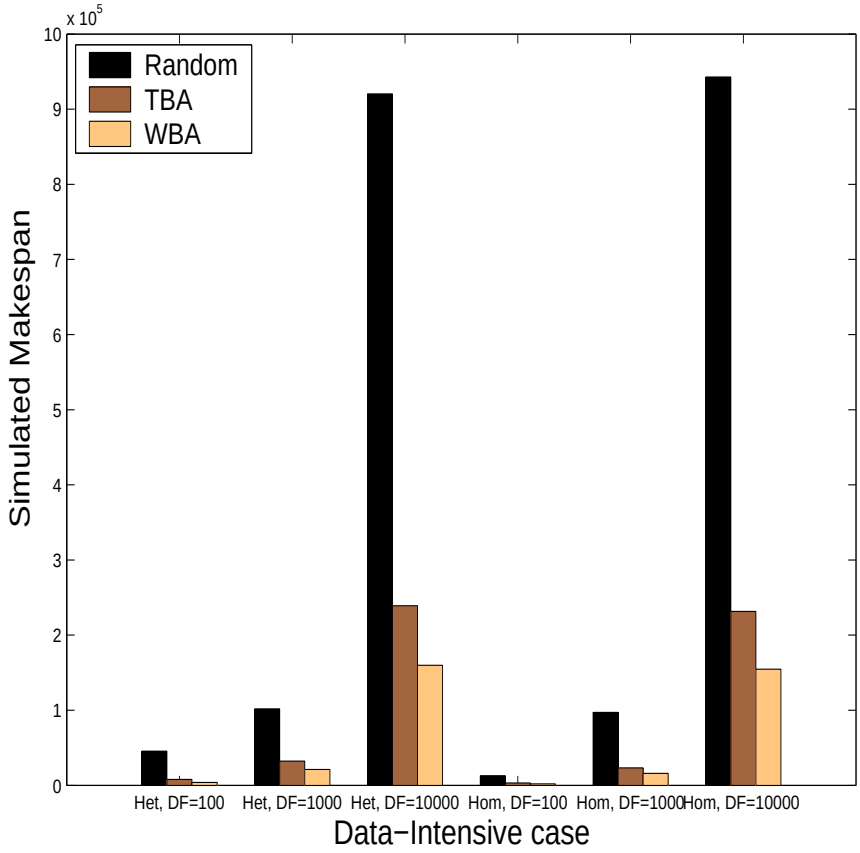

Figure 6.5 : Makespan of random vs TBA/WBA for data-intensive scenario

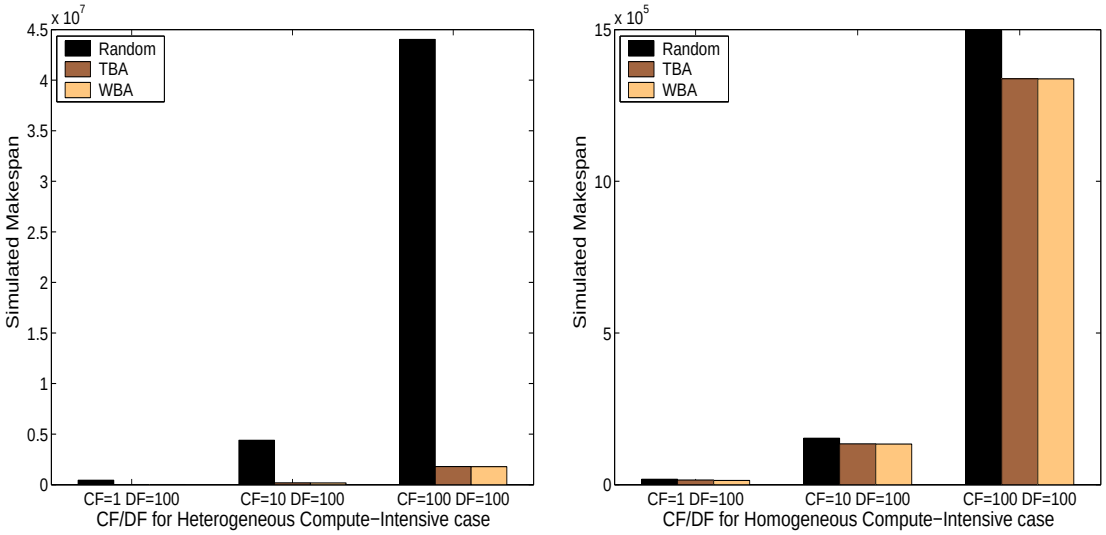

Figure 6.6 : Makespan of random vs TBA/WBA for compute-intensive scenario

Chapter 7

Scalable Workflow Scheduling

One of the drawbacks of the one step plan-ahead workflow scheduler is the time-complexity of the scheduling algorithm. This poses scaling problems in large Grid resource environments consisting of thousands of resources.

In this chapter, we propose two approaches to tackle the scalability problem. In the first approach, we run the expensive plan-ahead scheduler on a pruned resource set so that the scheduler doesn't go through all individual nodes. We prune the universe of resources either using simple heuristics for resource selection or through intelligent pruning using virtual Grid descriptions. The first section describes this approach in detail. The second approach is to modify the workflow scheduler to schedule the workflows directly onto abstract resource classes like clusters and not onto individual nodes in the clusters. We envision that in future, the Grid will have "manageable" number of abstract resource classes, though there may be thousands of nodes in the entire resource set.

7.1 Scalable Workflow Scheduling using Virtual Grids

Scheduling of applications onto Grid platforms poses new challenges and has been studied for many relevant application classes. One distinguishing feature of Grid platforms is the large number of individual resources, with current systems comprising thousands of individual devices and future systems comprising tens of thousands of devices and beyond. Such large numbers of resources pose many scalability problems

(e.g., resource discovery, resource monitoring). In this section we specifically address the scalability of the scheduling algorithm itself: how can one compute an application schedule in a short amount of time while considering a large number of potential resources? Although most scheduling heuristics exhibit polynomial time complexity, their running time can be prohibitive when used for computing a schedule over a large resource environment.

Although the resource environment may contain large numbers of resources, all taken into consideration when computing a schedule, typically only a small subset of these resources is used for running the application. In essence, most scheduling heuristics typically perform *implicit resource selection*: the set of resources used by the application emerges from the computation of the schedule. In this section, we improve the scalability of the scheduling process by performing *explicit resource selection*: we decouple resource selection from scheduling.

7.1.1 Virtual Grids - vgDL and vgES

A fundamental challenge for Grid applications is to describe and obtain appropriate resources to enable efficient, high performance execution. This is challenging from many standpoints, including the definition of an appropriate abstraction to describe resource needs, the difficulty of finding appropriate resources quickly in an environment that may contain tens or hundreds of thousands of resources, and interacting with diverse, autonomous resource managers that implement their own resource management and access policies. The typical approach is for the application to provide a resource description written in some language, and for the system to return a list of matching resources that can be used by the application [RLS98, LYFA02, RLS03, LF04].

For the purpose of explicit resource selection, we use concepts from the VGrADS

project [vgr], which provides combined resource selection and binding, embodied in a *"Virtual Grid" (VG)* abstraction [CCKH05, KLH+05]. In contrast with traditional low-level resource description and selection systems (e.g. [ea02]) that focus on individual, quantitative resource characteristics, the VG provides a high-level, hierarchical abstraction of the resource collection that is needed by an application. The application specifies its resource needs using a high-level language, *vgDL (Virtual Grid Description Language)*, and the *Virtual Grid Execution System (vgES)*, finds and allocates appropriate resources for the application. A VG, which is really an active entity (i.e., runtime object) that is the instantiation of the desired resource collection, is returned to the application. The application can then use the VG to find specific information about the allocated physical resources, to deploy application components, and to modify or evolve the resource collection. vgES uses efficient search techniques [KLH+05] based on resource classification in a relational database. In what follows, we only describe features of vgDL relevant for explicit resource selection.

The vgDL language uses high-level resource abstractions that correspond to what Grid application programmers typically use to organize their applications portably across many different resource environments. vgDL was designed based on a detailed study of half a dozen real-world applications [LBC99, lea]. This study showed that in order to design for performance (and to manage complexity) portably, application developers typically use three simple resource abstractions to aggregate individual resources. Consequently, vgDL contains three resource aggregates, distinguished based on homogeneity and network connectivity: (1)*LooseBag* – a collection of heterogeneous nodes with poor connectivity; (2)*TightBag* – a collection of heterogeneous nodes with good connectivity; (3)*Cluster* – adding homogeneity, a well-connected set of nodes with identical (or nearly so) individual resource attributes. Each aggregate specifies a range for its size (i.e., number of elements). A user can specify constraints

on attributes of individual elements within the aggregate (e.g., clock rate, processor architecture, memory, etc.), or constraints on aggregate attributes (e.g., total aggregate memory, total aggregate disk space). Aggregates can be nested (e.g., a *LooseBag* of *Cluster*) to arbitrary depth. vgDL has been used to express resource abstractions for over half a dozen Grid applications, and appears to be sufficient for all of the applications we have studied to date.

An important aspect of vgDL is that many of the characteristics of vgDL descriptions are qualitative. In our analysis of application needs, we found that detailed quantitative specifications are often a distraction, and as such cause resource specifications to be fragile when moving to new resource environments. In view of this, vgDL provides four operators that define network connectivity: *close*, *far*, *highBW*, and *lowBW*. These composers indicate coarse notions of network proximity in terms of latency and bandwidth. A particular implementation will use specific quantitative values as definitions for these operators, as appropriate for distribution of Grid resources, and changing as technology advances. Applications that require detailed quantitative resource information for the resources they obtain can query Grid information services [WSH99, FFK+97] for the resources in a VG once it has been instantiated. With these aggregates and network operators, an application can structure the specification of its resource environment in top-down fashion and decorate components with constraints when desired. In addition to constraints, applications can also express resource preference by using a scalar rank function: a user-defined expression of basic arithmetic operators, resource attribute and resource aggregate attribute values that define a scalar value that represents the quality of that resource set for the applications request. Because the vgDL requests are hierarchical, a specification may include multiple ranking functions (one at each level in each subcomponent). Compared to other systems [RLS98, LYFA02, RLS03, LF04], the vgDL ranking functions

provide significant flexibility, allowing combination of multiple attributes in complex fashion easily.

Given that (1) vgDL makes it possible to specify high-level, qualitative resource requirements easily and (2) vgES system can perform fast resource selection in large-scale resource environments, the VGrADS project provides and ideal foundation for decoupling resource selection from application scheduling.

7.1.2 Decoupled Approach: Scheduling in Large Scale Grids

In some cases, scheduling algorithms map at least one task to each resource, for instance when resources are roughly identical, application tasks are independent, and the number of tasks is larger than the number of resources. However, when there are many heterogeneous resources and/or when the application has less concurrent tasks than there are resources, many resources are unused. This is a common case when scheduling scientific applications onto large-scale Grid platforms, with current platforms comprising multiple thousands of resources and future platforms envisioned to comprise tens of thousands of resources and beyond. In fact, most applications use only a very small fraction of all available resources. We say that in such cases, the scheduling algorithm performs *implicit* resource selection. The problem is that, on platforms that comprise this many resources, the time to run even a polynomial-time heuristic can be prohibitive (especially if the algorithm takes into account communication of data over network links), making application scheduling unscalable.

A solution to address this scalability problem consists of decoupling resource selection from application scheduling. In a first phase, one performs *explicit* resource selection. In a second phase, one performs scheduling on the selected resources rather than on the whole resource universe. The key point here is that a decoupled approach makes it possible to compute schedules faster, by several orders of magnitude, making

application scheduling scalable to large-scale platforms. In fact, this decoupling may make it possible to run expensive scheduling algorithms on the explicitly selected resources.

Several straightforward approaches exist for selecting resources explicitly (for instance random selection), and we examine a few in the rest of this section. We claim that using a system such as vgES to perform explicit resource selection makes it possible to achieve schedules that are comparable in quality to the ones obtained when letting the scheduling algorithms perform implicit resource selection over the whole resource universe, at dramatically higher scalability. In our approach, the scheduler issues a vgDL specification to the vgES system, which quickly finds (and binds) matching physical resources, and a VG is returned. The scheduler then extracts from the VG precise information about the physical resources and runs its scheduling algorithm considering only these resources.

While decoupling resource selection from scheduling in large-scale systems as described above clearly improves scalability, a key question is: what is the impact of decoupled resource selection and scheduling on the quality of the resulting schedule? In the following sections, we study decoupled resource selection and scheduling in the context of workflow applications in large-scale Grid environments. We use several versions of the EMAN and Montage workflows as example workflow applications.

7.1.3 Scheduling Algorithms

While our decoupling approach is applicable to any scheduling algorithm, we chose to apply two workflow scheduling algorithms to evaluate our approach. We use a simple greedy workflow scheduling scheme and a substantially more expensive heuristic workflow scheduling scheme as in chapter 4. We understand that different scheduling algorithms scale differently and hence chose these two representative algorithms

having differing costs. We apply the two scheduling schemes on (a) the universe of available resources and (b) the set of resources obtained using explicit resource selection for instances of EMAN and Montage DAGs.

7.1.4 What VG to ask for?

In the scenario of decoupled resource selection and scheduling, a key question is what to ask for for the initial resource selection. From our experience, there are mainly a few resource abstractions most applications ask for. The job is to then generate an appropriate vgDL description of the required resources.

Structure of the VG

The structure of the required VG is an important concern because it has to "match" well with the application structure. For example a "LooseBag" of nodes is a poor resource requirement description for a very tightly coupled computation. Inferring the structure of an application automatically is an interesting and important research question. For our scenario of workflow applications, we already know the structure of the application from the DAG representation of the application. For our cases of EMAN and Montage the abstraction that looks most suitable is a "LooseBag" of "TightBag" of nodes. We already know that the set of available components at a given point in the workflow are embarrassingly parallel or parameter sweep steps. Since they are not communicating heavily, a "LooseBag" of "TightBag" of nodes seems to the ideal abstraction for the Montage and EMAN workflows. We want to mention that the structure of the VG to ask for is very application specific. Hence, potentially different vgDLs will be generated for other applications.

Type of resources

It is also possible to ask for specific types of resources in the vgDL description. Our approach is based on the types of resources for which we have performance models. Since most scheduling algorithms rely on some form of expected execution times for resources, the type of resources for which that information is available seems to be a good criteria for pruning the resource set. Randomly projected expected execution times may result in poor makespans in practice (as we see in chapter 4 and 6). An issue is what if we don't have accurate performance models. In that case, we can substitute them with simple static models like MHz ratings with potential loss of accuracy of makespan.

Number of resources

The other key element in the vgDL description is to specify a bound on the number of required resources. Potentially, we can ask for the universe of resources. But, then that does not show any value of intelligent resource selection and won't decrease the cost to schedule. Our approach is to use a simple static scheme based on the structure of the DAG. For the number of resources to ask for, we ask for N resources where N as the maximum width of any parallel phase in the input DAG. This seems to be a simple upper bound.

7.1.5 Evaluation

The results of the experiments confirm the hypothesis of better scalability of the decoupled approach over the one-step approach, where (selection + scheduling) time for the decoupled approach is only a fraction of the scheduling time for the one-step approach. The results also show that, for a wide variety of computation-communication

characteristics of DAGs (from EMAN and Montage), the turnaround time (makespan + scheduling time) for the decoupled approach is much less than that for the one-step approach. For communication intensive DAGs, the maximum gain was about 66%. For DAGs with relatively balanced communication and computation, the gain was about 50% on average. Zhang et al. [ZMC+06] present the experiments and results in detail.

7.2 Scheduling onto Abstract Resource Classes

The scheduling algorithms described so far schedule the workflow components onto individual nodes on the available Grid resource set. As explained earlier, this causes a scaling problem for the scheduler in large resource environments. In the last section, we solved the scaling problem by restricting the size of the universe for the scheduler. Another approach to tackle the scaling problem is to schedule directly onto abstract resource classes like clusters. An abstract resource class is defined as an aggregate of nodes like "Clusters", "TightBags", "LooseBags" etc. This approach implies that the scheduler, instead of going over all nodes in the resource set, goes over only the available resource classes. This particular scheduling problem can be formulated as follows.

7.2.1 Problem Formulation

The input to the scheduler is the workflow DAG with a restricted structure. We restrict the workflow DAG to have identical nodes at each logical level. By identical, we mean that nodes in the same level perform the same conceptual function (generally corresponding to same component executable) with potentially different input data sets (of similar size). This is very typical for many scientific workflows as in

EMAN and Montage, where the workflow consists of several parallel sweep steps in a sequence. From the resource side, the input is a set of available clusters. Each of the available clusters have attributes like the number of nodes, architecture, CPU speed of the nodes, available memory per node etc. It is also assumed that the network connectivity (latency and bandwidth) between the available clusters is known. It is also assumed that per-node performance models for each component are known for each cluster. The desired output from the scheduler is to find the mapping for each DAG level, i.e. for each level we have to find the number of instances mapped to each cluster (resulting in an eventual mapping for the whole DAG). The objective is to minimize the makespan at each level of the workflow DAG. We model the problem as follows.

7.2.2 Abstract Modeling

For each DAG level, we are given N instances of the parallel component. We have M clusters available to map them to. There are $r_1, r_2, ...r_M$ nodes in the M clusters, i.e. cluster C_i has r_i nodes. $t_1, t_2, ...t_M$ are the per node rank values for each of the M clusters. The rank values incorporate both the computation and communication costs. The aim is to find a partition $(n_1, n_2, ...n_M)$ of N with $n_1 + n_2 + ...n_M = N$ such that the overall completion time for the level is minimized. The output is n_i, the number of instances mapped to each cluster C_i.

The analytical method to solving this problem is to find the solution for the following min-max problem.

$$\min_{partitions: p_i=[n_i]} (\max_i (t_i \lceil \frac{n_i}{r_i} \rceil)) \tag{7.1}$$

However, there is no obvious, easy analytical solution to this equation because of the discrete nature of the problem. Any analytical solution has to search the space of

partitions of N, which in general is a large space. So, we resorted to a simple intuitive, iterative approach to solving this problem.

7.2.3 Iterative DP Approach

The idea is to iteratively map an instance onto a cluster by keeping track of the number of instances already mapped to the cluster and which "round" the cluster is at in the mapping process. The next instance is mapped to the cluster that obtains minimum increment in the current makespan. This is akin to distributing the instances to clusters. Note that whenever a cluster goes to the next "round" meaning r_i instances have been mapped to C_i since the last "round", the makespan for that cluster makes a jump by t_i and stays there until r_i instances are mapped to C_i in the current round. Every instance is tentatively mapped to each cluster and the cluster that results in minimum makespan increase is chosen for the instance. This goes on until all instances are mapped. The algorithm follows.

Algorithm

The following algorithm, Algorithm 7.1, implements the iterative DP (Dynamic Programming) approach to find the mapping. For each cluster, p the following are maintained: round(p) to keep track of the current round for the cluster, numMapped(p) to keep track of the number of instances mapped to the cluster and makespan(p) to keep track of the makespan for each cluster. In order to map the entire workflow, this algorithm needs to be executed for each topological level of the workflow.

Complexity

The theoretical complexity of this scheduling algorithm for a given level in the workflow is $O(\#instances \star \#clusters)$. Hence, overall complexity for the entire workflow

foreach *instance, i* **do**

foreach *cluster, j* **do**

Tentatively map i to j;

Record makespan for each j by taking care of round(j);

endforeach

Find cluster, p with minimum makespan increase;

Map i to p;

Update round(p), numMapped(p) and makespan(p);

endforeach

Algorithm 7.1: Scheduling onto Clusters

is $\sum_{level(l)} O(\#instances(l) \star \#clusters)$.

7.2.4 Experimental Evaluation

In order to evaluate the scalability of the Cluster-Level Scheduler, we compared the scheduling time of the Cluster-Level scheduler with the one-step workflow scheduler (both greedy and heuristic versions). Scheduling time is defined as the time it takes to compute the schedule for the entire workflow in-advance. We also compare the makespan generated by the Cluster-Level scheduler with the one generated by the one-step scheduler (both greedy and heuristic versions).

We used the EMAN and the Montage application workflows as example workflow application scenarios. We used three versions of the EMAN workflow and three versions of the Montage workflow. We obtained the different versions of each application workflow by varying the number of parallel components at the parameter-sweep steps of each workflow. For EMAN we used the 55-110, 110-379 and 220-440 versions, where 55-110 version stands for 55 parallel components in the first parameter sweep

Number of Clusters	10	25	50	75	100	250	500	1000
Number of Nodes	258	996	2104	2144	3168	8550	19050	36061

Table 7.1 : Number of Clusters and Nodes

step and 110 parallel components in the second parameter sweep step. For Montage we used the 24-51-24, 50-121-50 and 153-407-153 versions. We used the performance model values for per component estimates of execution times.

The resource model is based on a tool that generates populations of representative compute clusters [KCC04]. This tool uses empirical statistical models of cluster characteristics (e.g., number of processors, processor architecture, processor clock rate) obtained from a survey of 114 real-world clusters. Table 7.1 shows the summary of number of clusters and the total number of nodes in each set of clusters. The largest resource set consisted of 1000 clusters with about 36k nodes. We executed the scheduler on a 900MHz Itanium node.

The network model is as follows. Conforming to the results in [LS01], we generated end-to-end latencies between compute clusters according to a Normal distribution. We set the mean of this distribution to 100ms, conforming to the results in [Mea04] and we confine the latencies from 1 to 200ms. For the network bandwidths, we set the connection within a cluster as 1000Mb/s and all the inter connection between clusters range from 10Mb to 100Mb/s. These numbers are primarily based on results listed in [YSI05, DAH$^+$04]. Furthermore, we ensure that the higher the latency the lower the bandwidth.

Comparison with one-step plan-ahead scheduler

Figure 7.1 shows the results of the scheduler scaling experiments for the three EMAN workflows and figure 7.2 shows the results of the scheduler scaling experiments for the three Montage workflows. The X-axis denotes the number of available clusters and Y-axis denotes the scheduling time in seconds for the particular scheduling algorithm. From the results, we can infer that the Cluster-Level scheduler outperforms the others on large Grids with much better scaling. This is because the Cluster-Level scheduler is linear on the number of clusters, not on the number of nodes in the clusters. The results from Figure 7.3 verify the same for a given workflow (EMAN 220-440 in this case). This is true for all the workflows we tested.

Figure 7.4 and Figure 7.5 show the turnaround time (makespan + scheduling time) for Montage and EMAN DAGs when using the heuristic, greedy and Cluster-Level schedulers for varying number of available clusters. The results show that the Cluster-Level scheduler has the minimum turnaround time for all cases, because of reduced scheduling time. Also the quality of makespan doesn't suffer for the EMAN DAGs. For Montage DAGs, the makespan is about 27% more (on average) than that obtained using the heuristic scheduler.

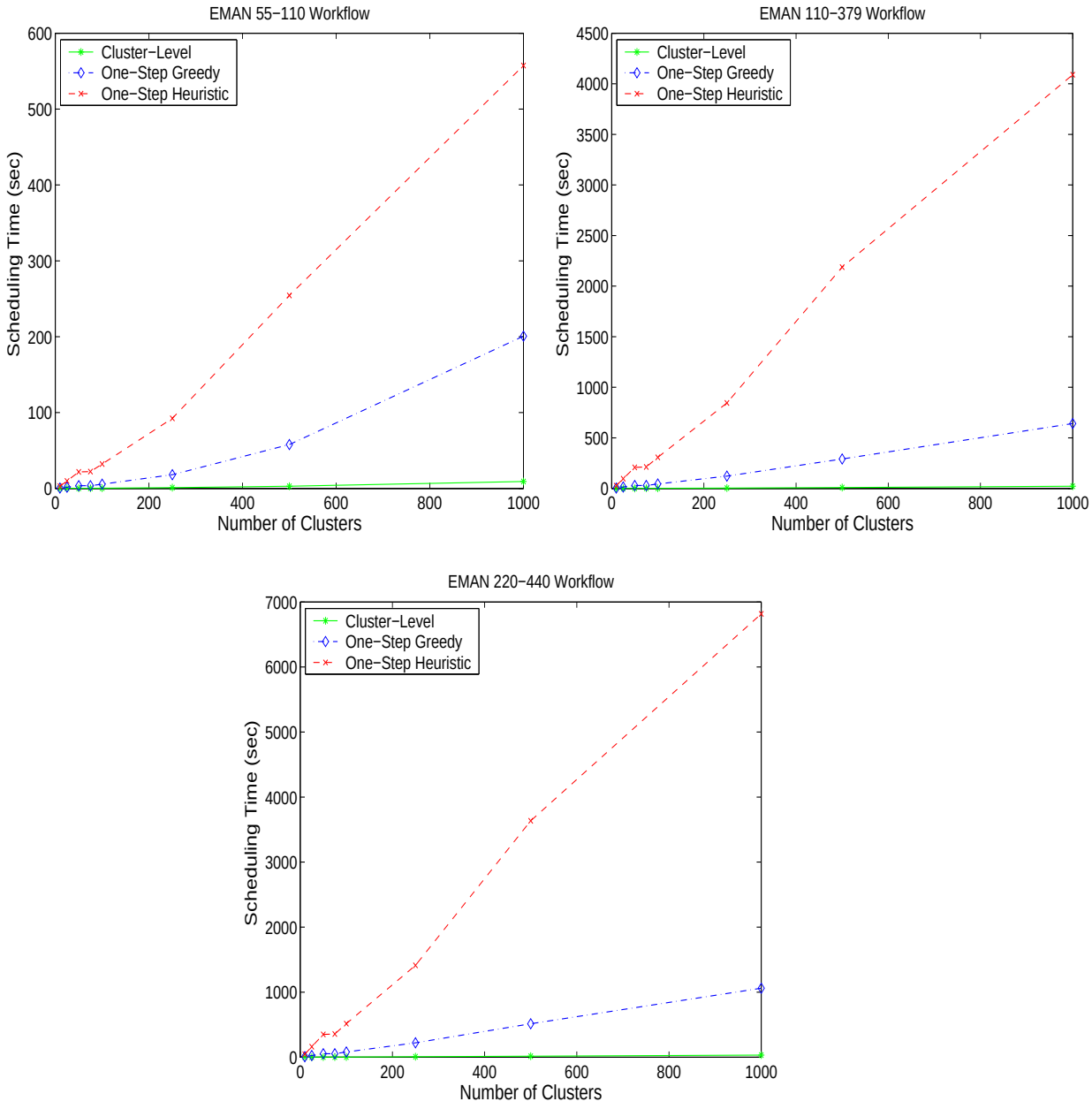

Figure 7.1 : Cluster-Level vs. One-Step (greedy and heuristic) for EMAN

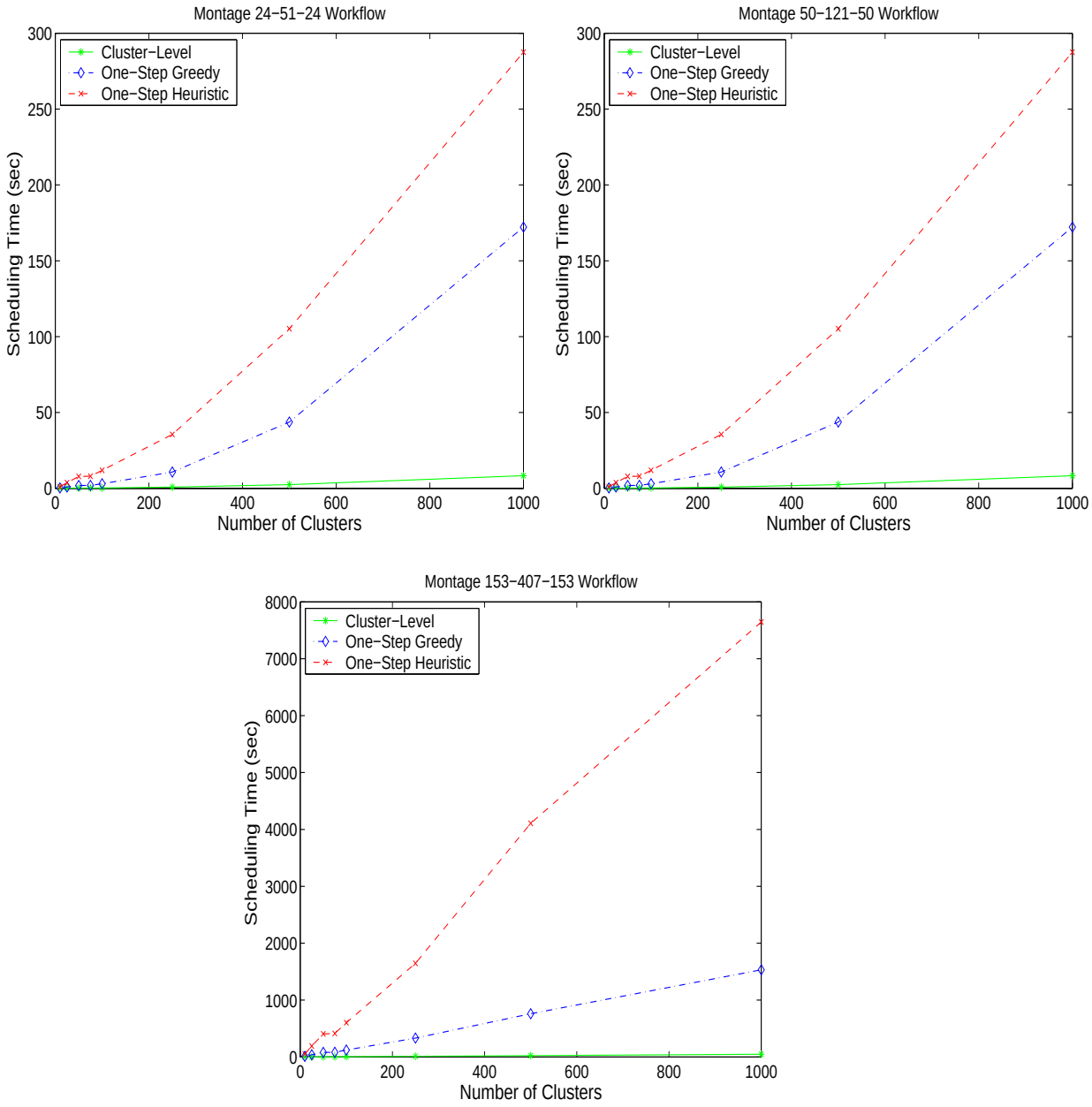

Figure 7.2 : Cluster-Level vs. One-Step (greedy and heuristic) for Montage

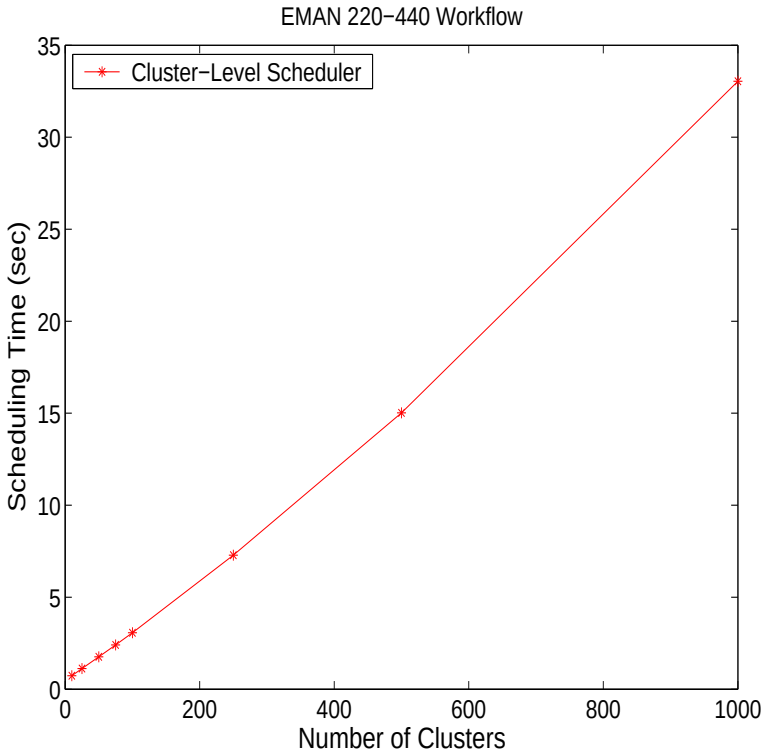

Figure 7.3 : Scaling of Cluster Level Scheduler

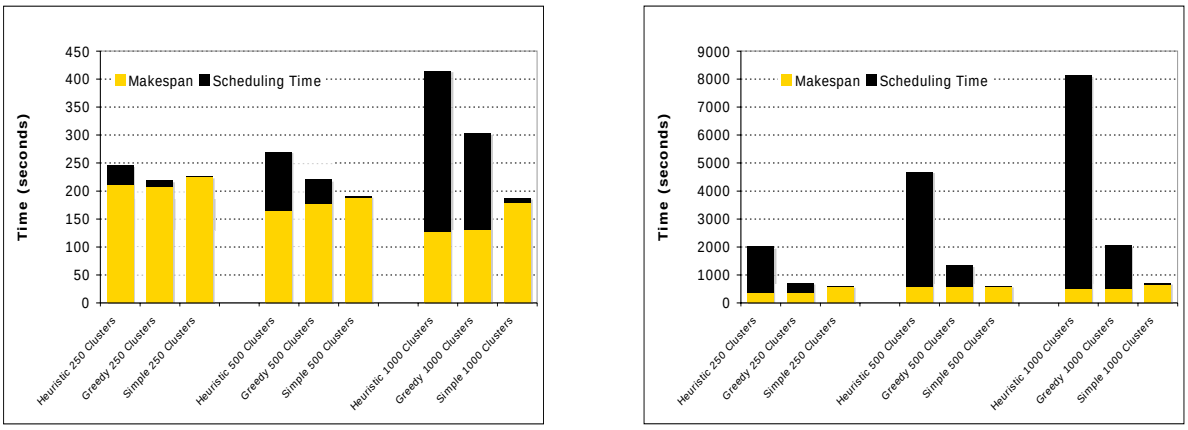

Figure 7.4 : Makespan and scheduling time for small and large Montage DAGs

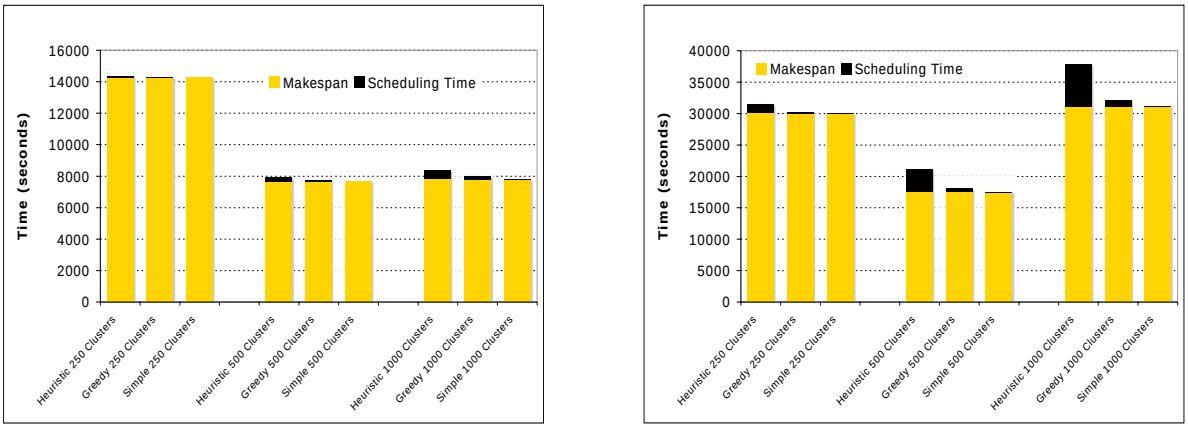

Figure 7.5 : Makespan and scheduling time for small and large EMAN DAGs

Chapter 8

Conclusions

In this dissertation we describe several techniques for scheduling scientific application workflows onto heterogeneous, distributed Grid systems. The objective of this work has been to automate the scheduling process for application workflows and achieve good turnaround times for the workflows as well by effective and efficient scheduling.

To achieve this objective, the research leading to this dissertation resulted in designing and implementing techniques to schedule workflows.

- We developed new strategies for scheduling and executing workflow applications on Grid resources. Workflow scheduling is based on heuristic scheduling strategies that use application component performance models. These strategies were applied in the context of executing EMAN, a Bio-imaging workflow application, on the Grid.

  The results of the 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 for the studied application. This strategy also achieves optimal load balance across the different Grid sites for this instance of the application. We have also shown that machine loads and inaccurate performance models have an adverse affect on the quality of schedule and hence the performance of the application.

  In order to make the scheduling tool applicable to HPC systems with batch queue front ends, we modified the scheduler to take into account the batch

queue wait times in order to take scheduling decisions.

- We proposed a new middle-out scheduling heuristic to address the myopia of the top-down scheduler. Middle-out scheduling schedules the key components of the workflow first and then propagates the mapping bottom up and top down. The results of this heuristic show that, in some cases where the top-down scheduler gets stuck to a non-optimal resource, the middle-out scheduler chooses the better resources. In some other cases, the performance from the resulting schedule depends on network connectivities.

- We compared the plan-ahead scheduling strategy to existing task-based, online/dynamic strategies to workflow scheduling. The results from the experiments with the Montage application workflows show that, the global workflow based plan-ahead scheduler outperforms the dynamic/task-based scheduler for the data-intensive Montage workflows. The difference is negligible for compute intensive workflows. Both the workflow based and task based approaches outperform random scheduling strategies by an order of magnitude, as large as 11.

- We presented scalable workflow scheduling techniques for future Grids, which may consist of hundreds of thousands of resources. The first technique is based on a decoupled approach of explicit resource pruning using virtual grids and then scheduling onto the pruned resource set. The decoupled strategy results in better turnaround time than the one-step approach for EMAN and Montage workflows with varying computation/communication characteristics. The second technique schedules the workflows directly onto abstract resource classes like clusters instead of individual resources. The results show directly scheduling onto clusters improves the scheduling time by orders of magnitude because

the Cluster-level scheduler is linear on the number of clusters.

8.1 Future Work

The work presented in this thesis is only a step toward making Grid programming easier and efficient. It opens up research on various interesting related problems.

8.1.1 Short and Medium Term

Currently, the XML description of the input application workflow is created by hand. To make programming for the Grid easier, we need tools to discover the application workflow from the application code. Sophisticated program analysis techniques need to be employed to output the application workflow from the application code. As a side effect of this, it would then be possible to generate vgDL queries for a given workflow automatically.

Increasingly, programs are being written in scripting languages like Matlab and Python. Other than the ease of use of these languages, glue code can be written very easily in these languages to tie up several application pieces to make up a whole application. Incidentally, the next port of EMAN happens to be a have a Python interface where scientists can glue together core EMAN components using Python. Generating application workflows from a piece of code written in Matlab or Python is an interesting problem and warrants research. If that is possible, it will be a step closer to our vision of easing the job of the end scientist. All these combined can produce a more end-to-end tool for Grid programming.

Several workflow optimizations like pipelining two steps of the workflow or fusing nodes in the workflow can be employed. Pipelining will enable a subsequent step in the workflow to start as soon as a segment of data is created on the parents of the

component. This can have an impact on the turnaround time of the application.

The issue of scheduling multiple DAGs is also an important research problem. This problem arises when the scheduler receives multiple workflows to be executed on the same set of available resources from a single or multiple users. How the scheduler prioritizes and selects resources in this scenario is an interesting problem.

8.1.2 Long Term

The work in this thesis has concentrated on optimizing only one metric of performance - the application makespan or turnaround time. In the future, one may need to optimize other criteria like throughput, resource utilization, real-time deadlines etc. It may also be required to optimize a combination of these metrics.

The issue of fault-tolerant workflow execution is an important research problem. This problem arises when the execution of large workflows have to be interrupted or stopped for some reason - failure of the nodes, disk crash etc. There need to be strategies to start the workflows again from the point where it had left off. One needs to deal with issues of identifying these failure points (maybe by monitoring and check-pointing) and rescheduling the workflows.

Economic scheduling is a new area of research. When Grid economies are considered, time is not the only entity we are concerned with. New scheduling techniques need to be developed when "cost to use a particular resource" is a concern. One may trade a very fast expensive machine for a slower cheaper machine in this scenario. Scheduling workflows from a combined standpoint of economics and time is an important research problem.

In the long term, the scheduler needs to deal with information from various resource management systems, systems having advanced reservations and other vagaries of co-allocation.

The advent of new architectural features like multi-core processors, reconfigurable chips and in-the-box heterogeneity of supercomputers poses new challenges in terms of developing or modifying existing scheduling tools that take advantage of these new features. Implications of these new architectures on scheduling tools (or programming tools in general) need to be investigated.

Bibliography

[ABD+01] Gabrielle Allen, Werner Benger, Thomas Dramlitsch, Tom Goodale, Hans-Christian Hege, Gerd Lanfermann, Andre Merzky, Thomas Radke, Edward Seidel, and John Shalf. Cactus tools for grid applications. *Cluster Computing*, 4(3):179–188, 2001.

[ACD74] T.L. Adam, K.M. Chandy, and J. Dickson. A comparision of list scheduling for parallel processing systems. *Communications of the ACM*, 17(12):685–690, 1974.

[bio] Biogrid - http://www.biogrid.net/.

[bir] Biomedical informatics research network - http://www.nbirn.net/.

[BJD+05] Jim Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[BLR02a] Olivier Beaumont, Arnaud Legrand, and Yves Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. Technical Report LIP RR-2002-36, 2002.

[BLR02b] Olivier Beaumont, Arnaud Legrand, and Yves Robert. Static scheduling strategies for heterogeneous system. Technical Report LIP RR-2002-29, 2002.

[BNW05] John Brevik, Daniel Nurmi, and Rich Wolski. Predicting bounds on queuing delay in space-shared computing environments. Technical Report CS2005-09, Department of Computer Science, University of California at Santa Barbara, 2005.

[BW99] B.C. Barish and R. Weiss. Ligo and detection of gravitational waves. *Physics Today*, 52(10), 1999.

[CCKH05] Andrew Chien, Henri Casanova, Yang-Suk Kee, and Richard Huang. The virtual grid description language: vgdl. Technical Report CS2005-0817, Department of Computer Science, University of California at San Diego, 2005.

[CDK+04] K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan, and J. Dongarra. New grid scheduling and rescheduling methods in the grads project. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 10 (NSF NGS Workshop)*, 2004.

[CJ04] Yves Caniou and Emmanuel Jeannot. Efficient scheduling heuristics for gridrpc systems. In *QOS and Dynamic System workshop of IEEE ICPADS (International Conference on Parallel and Distributed Systems)*, pages 621–630, 2004.

[CLZB00] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, and Francine Berman. Heuristics for scheduling parameter sweep applications in grid

environments. In *9th Heterogeneous Computing workshop (HCW'2000)*, 2000.

[CNS03] S. J. J. Cao, G.R. Nudd, and S. Saini. Gridflow: Workflow management for grid computing. In *3rd International Symposium on Cluster Computing and the Grid*, 2003.

[dag] Dagman metascheduler - http://www.cs.wisc.edu/condor/dagman.

[DAH+04] A. Denis, O. Aumage, R. Hofman, K. Verstoep, T. Kielmann, and H. E. Bal. Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems. In *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC'04)*, pages 97–106, Washington, DC, USA, 2004. IEEE Computer Society.

[dat] Datagrid project: Work package 01.

[DBG+03] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn, Albert Lazzarini, Adam Arbree, Richard Cavanaugh, and Scott Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.

[DCB02] H. Dail, H. Casanova, and F. Berman. A modular scheduling approach for grid application development environments. Technical Report CS2002-0708, 2002.

[ea02] Ken Kennedy et al. Toward a framework for preparing and executing

adaptive grid programs. In *International Parallel and Distributed Processing Symposium*, 2002.

[ea04] G.B. Berriman et al. Montage a grid enabled image mosaic service for the national virtual observatory. *ADASS XIII ASP Conference Series*, XXX, 2004.

[FFK+97] Steve Fitzgerald, Ian Foster, Carl Kesselman, Gregor von Laszewski, Warren Smith, and Steve Tuecke. A directory service for configuring high-performance distributed computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, 1997.

[FFLT01] T. T. James Frey, Ian Foster, Miron Livny, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. In *Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, 2001.

[fJSS01] A GRASP for Job Shop Scheduling. *Essays and Surveys on Metaheuristics*. Kluwer Academic Publishers, 2001.

[FK99] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffmann Publishers, Inc., 1999.

[FK03] Ian Foster and Carl Kesselman. *Grid 2*. Morgan Kauffmann Publishers, Inc., 2003.

[gem] Grid-enabled medical simulation services - http://www.gemss.de/.

[GJ79] Michael R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. 1979.

[GLS] Mark Goldenberg, Paul Lu, and Jonathan Schaeffer. Trellisdag: A system for structured dag scheduling.

[gria] Gridlab project: http://www.gridlab.org.

[grib] Griphyn project: http://www.griphyn.org.

[HCAL89] J.J. Hwang, Y.C. Chow, F.D. Anger, and C.Y. Lee. Scheduling precedence graphs in systems with interprocessor communication costs. *SIAM Journal of Computing*, 18(2):244–257, 1989.

[HSvL03] XiaoShan He, XianHe Sun, and Gregor von Laszewski. Qos guided min-min heuristic for grid task scheduling. *J. Comput. Sci. Technol.*, 18(4):442–451, 2003.

[IKT01] I.Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[ivd] ivdgl project: http://www.ivdgl.org.

[JP99] Klaus Jansen and Lorant Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. pages 408–417, 1999.

[KA96] Yu-Kwong Kwok and Ishfaq Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors:. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521, 1996.

[KA99a] Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[KA99b] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999.

[KB88] S.J. Kim and J.C. Browne. A general approach to mapping of parallel computations upon multiprocessor architectures. In *Proceedings of International Conference on Parallel Processing*, pages 1–8, 1988.

[KCC04] Y.-S. Kee, H. Casanova, and A. A. Chien. Realistic Modeling and Svnthesis of Resources for Computational Grids. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2004. IEEE Computer Society.

[KL88] B. Kruatrachue and T.G. Lewis. Grain size determination for parallel processing. *IEEE Software*, pages 22–32, January 1988.

[KLH+05] Yang-Suk Kee, Dionysios Logothetis, Richard Huang, Henri Casanova, and Andrew Chien. Efficient resource description and high quality selection for virtual grids. In *Proceedings of the 5th IEEE Symposium on Cluster Computing and the Grid (CCGrid'05), Cardiff, U.K.* IEEE, 2005.

[LBC99] S.J. Ludtke, P.R. Baldwin, and W. Chiu. Eman: Semiautomated software for high-resolution single-particle reconstructions. *J. Struct. Biol.*, 128:82–97, 1999.

[LBH+04] W. Li, R. Byrnes, J. Hayes, V. Reyes, A. Birnbaum, A. Shabab, C. Mosley, D. Pekurowsky, G. Quinn, I. Shindyalov, H. Casanova, L. Ang, F. Berman, M. Miller, and P. Bourne. The encyclopedia of life project: Grid software and deployment. *Journal of New Generation Computing on Grid Systems for Life Sciences*, 2004.

[lea] Lead project - http://lead.ou.edu/.

[LF04] C. Liu and I. Foster. A constraint language approach to matchmaking. In *RIDE*, 2004.

[LFPA05] Chua Ching Lian, Tang F, Issac P, and Krishnan A. Gel: Grid execution language. *Journal of Parallel and Distributed Computing*, 2005.

[LS01] C. Lee and J. Stepanek. On future Global Grid communication Performance. *Heterogeneous Computing Workshop*, 2001.

[LST90] J. K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[LYFA02] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and implementation of a resource selection framework. In *11-th IEEE Symposium on High Performance Distributed Computing (HPDC11)*, pages 63–72, 2002.

[mam] Mammogrid - http://mammogrid.vitamib.com/.

[Mar03] Gabriel Marin. Semi-Automatic Synthesis of Parameterized Performance Models for Scientific Programs. Master's thesis, Dept. of Computer Science, Rice University, April 2003.

[MAS+99] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard 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.

[mea] Mead project: http://www.ncsa.uiuc.edu/expeditions/mead/.

[Mea04] Robert Morris and et al. `http://pdos.csail.mit.edu/p2psim/kingdata`, 2004.

[MS98] M. Maheswaran and H. Siegel. A dynamic matching and scheduling algorithm for heterogeneous computing systems, 1998.

[MWW03] Marek Mika, Grzegorz Waligora, and Jan Weglarz. *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.

[myg] mygrid - http://www.ebi.ac.uk/mygrid/.

[NBW+] Daniel Nurmi, John Brevik, Rich Wolski, Anirban Mandal, Chuck Koelbel, and Ken Kennedy. Evaluation of a workflow scheduler using integrated performance modelling and batch queue wait time prediction. In *In submission to ACM/IEEE conference on Supercomputing - SC'06*.

[ns2] Network simulator - http://www.isi.edu/nsnam/ns.

[NSW03] Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz. *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.

[nvo] The national virtual observatory project: http://www.us-vo.org/.

[OH96] Hyunok Oh and Soonhoi Ha. A static scheduling heuristic for heterogeneous processors. In *Euro-Par, Vol. II*, pages 573–577, 1996.

[per] Personal communication with dr. wah chiu and dr. steve ludtke at baylor college of medicine.

[ppd] Particle physics datagrid project: http://www.ppdg.net.

[RLS98] Rajesh Raman, Miron Livny, and Marvin H. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *HPDC*, pages 140–, 1998.

[RLS03] Rajesh Raman, Miron Livny, and Marvin H. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *HPDC-12*, pages 80–89, 2003.

[RR02] M Resende and C Ribeiro. *Greedy Randomized Adaptive Search Procedures, State-of-the-art Handbook in Metaheuristics*. Kluwer Academic Publishers, 2002.

[Sar89] V. Sarkar. *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge, MA, 1989.

[SCF+00] Shava Smallen, Walfredo Cirne, Jaime Frey, Francine Berman, Richard Wolski, Mei-Hui Su, Carl Kesselman, Stephen J. Young, and Mark H. Ellisman. Combining workstations and supercomputers to support grid applications: The parallel tomography experience. In *Heterogeneous Computing Workshop*, pages 241–252, 2000.

[SL93] G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architecture. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–186, 1993.

[SZ04] Rizos Sakellariou and Henan 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.

[THW02] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[TMSW01] Ian Taylor, Shalil Majithia, Matthew Shields, and Ian Wang. Triana workflow specification, 2001.

[Tra01] Tracy D. Braun et al. A comparision of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61:810–837, 2001.

[TWG+01] Valerie Taylor, Xingfu Wu, Jonathan Geisler, Xin Li, Zhiling Lan, Mark Herald, Ivan R. Judson, and Rick Stevens. Prophesy: Automating the modeling process. In *Proceedings of the Third International Workshop on Active Middleware Services in conjunction with High-Performance Distributed Computing-10*, 2001.

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

[vgr] Vgrads project - http://hipersoft.cs.rice.edu/vgrads.

[WG90] Min-You Wu and Daniel Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, 1990.

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[Wul98] C.-E. Wulz. Cms concept and physics potential. In *2nd Latin American Symposium on High Energy Physics (II-SILAFAE), San Juan, Puerto Rico*, 1998.

[YG94] T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. Technical Report TRCS94-12, 20, 1994.

[YSI05] L. Yang, J. M. Schopf, and I.Foster. Improving Parallel Data Transfer Times Using Predicted Variances in Shared Networks. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*. IEEE Press, 2005.

[ZMC+06] Yang Zhang, Anirban Mandal, Henri Casanova, Andrew Chien, Yang-Suk Kee, Ken Kennedy, and Charles Koelbel. Scalable grid application scheduling via decoupled resource selection and scheduling. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006)*. IEEE Press, 2006.

[ZS03] Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In *Proceedings of Euro-Par*, 2003.