Partner-Based Scheduling and Routing for Grid Workflows

by
Jawad Ashraf

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Department of Computer Science
University of Leicester

November 11, 2012

Partner-Based Scheduling and Routing for Grid Workflows

Jawad Ashraf

Supervisor: *Professor Thomas Elrebach*

Abstract

The Grid has enabled the scientific community to make faster progress. Scientific experiments and data analyses once spanning several years can now be completed in a matter of hours. With the advancement of technology, the execution of scientific experiments, often represented as workflows, has become more demanding. Thus, there is a vital need for improvements in the scheduling of scientific workflows. Efficient execution of scientific workflows can be achieved by the timely allocation of the resources. Advance reservation can ensure the future availability of heterogeneous resources and help a scheduler to produce better schedules.

We propose a novel resource mapping technique for jobs of a Grid workflow in an advance reservation environment. Using a dynamic critical path based job selection method, our proposed technique considers the conditional mapping of parent and child jobs to the same resource, trying to minimise the communication duration between jobs and thus optimising the workflow completion time. The proposed method is analysed in both static and dynamic environments, and the simulation results show encouraging performance especially for workflows where the communication costs are higher than the computation costs.

We also propose a hybrid of multiple scheduling heuristics for the aforementioned problem, which chooses the best among multiple schedules computed by different algorithms. Simulation results show a significant improvement over well known scheduling heuristics in terms of workflow completion time.

Considering the advance reservation environment, a better schedule for the earliest completion of a workflow can be achieved if better paths can be found for the transfer of data files between jobs executed on different resources. We propose a K-shortest path based routing algorithm for finding good paths in the advance reservation environment. The results show that our proposed algorithm performs very well in terms of the earliest arrival time of the data.

Finally, we also study a modified partner based scheduling heuristic for non-advance reservation environments. The results demonstrate that our proposed algorithm is a promising candidate for adoption in such Grid environments.

Declaration

The content of this submission was undertaken in the Department of Computer Science, University of Leicester, and supervised by Professor Dr. Thomas Erlebach during the period of registration. I hereby declare that the materials of this submission have not previously been published for a degree or diploma at any other university or institute. All the materials submitted for the assessment are from my own research, except the reference work in any format by other authors, which are properly acknowledged in the content.

Part of the research work presented in this submission has been published in the following papers:

- Jawad Ashraf and Thomas Erlebach. A new resource mapping technique for Grid workflows in advance reservation environments. In *Proceedings of High Performance Computing and Simulation (HPCS)*, pages 63 -70, 2010.
- Jawad Ashraf and Thomas Erlebach. A hybrid scheduling technique for Grid workflows in advance reservation environments. In *Proceedings of High Performance Computing and Simulation (HPCS)*, pages 98 - 106, 2011.

In the name of Almighty God, the Most Gracious, the Most Merciful

Acknowledgements

I thank to God for providing me the opportunity to work under the supervision of Professor Thomas Erlebach. I am grateful to him for helping me excel in research skills. In addition, I also learnt the soft skills from him, which encouraged me and kept me going throughout the PhD. He has always been available to help and encouragement. I consider myself fortunate and honored for being his student.

I also thank the Department of Computer Science, University of Leicester for arranging and supporting the participation in different events to improve my research skills. Thanks also goes to the GridSim development team who helped me getting started in the simulation world. I am also grateful to the Kohat University of Science and Technology, Pakistan for funding my PhD.

I appreciate the precious time my colleague gave me during the research period. Their discussions helped me in moving forward.

Finally, I am indebted to my parents as always, their love, prayers, support and teachings brought me this far. I highly regard the support from my wife for making the PhD easier for me. I also thank my brothers, sisters, relatives and friends for helping me whenever needed.

Contents

List of Figures

List of Tables

Acronyms

ADTT Absolute Data Transfer Time

AEST Absolute Earliest Starting Time

ALST Absolute Latest Starting Time

AR Advance Reservation

CAV Capacity Availability Vector

CPOP Critical Path on Processor

CT Completion Time

Comm Communication

DAG Directed Acyclic Graph

DCPG Dynamic Critical Path for Grids

DCPL Dynamic Critical Path Length

DCP Dynamic Critical Path

DS Data Size

EAT Earliest Arrival Time

ECT Earliest Completion Time

EDG European Data Grid

ETT Earliest Transfer Time

GA Genetic Algorithm

GRASP Greedy Randomised Adaptive Search Procedure

HBMCT Hybrid Balanced Minimum Completion Time

HEFT Heterogeneous Earliest Finish Time

HGP Human Genome Project

LSP Label Switched Path

MCP Modified Critical Path

MET Minimum Execution Time

MH Mapping Heuristic

MIPS Million Instructions Per Second

MI Million Instructions

MRS Multicost Routing and Scheduling

MUV Make UnAvailable

Mbps Megabit per Second

OBS Optical Burst Switching

OCS Optical Circuit Switching

OPS Optical Packet Switching

OLB Opportunistic Load Balancing

PDCPG Partner Based Dynamic Critical Path for Grids

PSPLIB Project Scheduling Problem Library

PSPWF Project Scheduling Problem Workflow

SA Simulated Annealing

WFMS Workflow Management System

estET Estimated Execution Time

expCT Expected Completion Time

rbw Requested Bandwidth

rcpu Requested CPU

Chapter 1

Introduction

In Grid computing, the computation and communication capabilities of heterogeneous, distributed resources are made available to users for solving complex problems. Many such problems require the execution of a number of jobs, each of which may require as input the output of a previously executed job. Such systems of jobs can be modelled as workflows. A fundamental issue that needs to be addressed when workflows are executed in a Grid is the scheduling problem, i.e., deciding when and where the jobs of the workflow are executed and how data is exchanged between them. In this thesis, we propose and evaluate new heuristic methods for workflow scheduling problems in advance reservation environments with and without resource failures, and in non-advance reservation environments. We also study the subproblem of routing data from one resource to another efficiently.

In this introductory chapter, we give the background and motivation of our work and state our contributions. Background on Grid Computing is discussed in Section 1.1. In Section 1.2, workflow scheduling is covered. Advance Reservation is briefly described in Section 1.3, and Optical Burst Switching (OBS) networks are discussed in Section 1.4. The motivation for our work is presented in Section 1.5. Our contributions are stated in Section 1.6. Finally, an outline of the structure of this thesis is given in Section 1.7.

1.1 Grid Computing

According to [35], John McCarthy suggested in the 1960s that the time-sharing capability of computing resources might lead to the outsourcing of computational tasks. Initially, this idea was not taken up because of the lack of capabilities in applications, computing and communication resources that would have been required for an implementation. With subsequent advancements in technology, the concept has now been realised in different forms such as Cluster, Grid and Cloud Computing.

Computing and data intensive tasks and the need for collaboration in scientific research have led to the creation of the Grid. There is no agreed upon definition of the Grid. Ian Foster presented three typical attributes of the Grid in [34], which are as follows:

- "Coordinated resources are not administered centrally".
- "Open standards are used".
- "Nontrivial quality of service is achieved".

A brief description of the above three attributes is as follows. Generally, a Grid is a pool of different types of resources that include computational, communication, storage resources and instruments. These resources are owned by different organisations or persons and are available for the utilisation by users across the globe. The user can pay for the utilisation of resources according to policies of the owner(s) of the resources. Thus there is no central administration over the resources of the Grid. Since different architectures, platforms and applications may interact with each other, open standard protocols and interfaces are required for the accomplishment of better and successful interaction. Further, the coordinated use of the Grid resources helps in providing greater quality of service, e.g., response time, as compared to the situation where resources are used in a non-coordinated way.

The Grid has effectively helped scientists and academics to achieve many of their goals in the last decade. One of the many success stories was a huge reduction in the duration of finding solutions to genome sequencing problems. In 2000, the Human Genome Project

(HGP) [5], with the main objective of determining the sequence of chemical base pairs which make up the human DNA, completed after 10 years. With the help of the Grid, in January 2008 every 4 days major genome centres could sequence the same number of base pairs as the HGP. Later on, new technologies have reduced this process to 10 hours [45].

Depending on the requirements of applications and the resource design, Grids can be categorised as follows [48]:

1. **Computational Grid.** This category accommodates the systems which are built to solve computing-intensive tasks. It is further subdivided into two categories, Distributed Supercomputing and High Throughput Computing. Distributed Supercomputing focuses on completing a task, by running it in parallel on multiple machines, within minimum time possible. Some examples of the problems for which Distributed Supercomputing is used are weather forecasting, molecular modeling and fluid dynamics. High Throughput computing targets increased job finishing rates. In case of processor design verification, tests are run with different parameter settings. These experiments are run on a high throughput Grid to increase the completion rate. Some of the many Grid projects that lie in this category are MyGrid [28] and Seti@home [14].

2. **Data Grid.** The characteristic of this category is to specifically address the issue of sharing and transmitting huge amounts of data. These systems allow scientists to collaborate with each other by sharing their datasets. To provide these mentioned facilities the data Grid has specialised infrastructure which the computational Grid lacks. In a computational Grid the storage management is handled by the applications themselves rather than by using the Grid's services. Fields like astronomy, high energy physics and climate simulation need such facilities to accomplish collaboration. There are several projects that focus on these demands, including The DataGrid Project [9], Particle Physics DataGrid Project [8] and BioGrid [1].

3. **Service Grid.** The objective of this category of systems is to provide services that

a single machine cannot provide. The services can include interaction services, multimedia services for heavy image/video rendering, data mining services for extensive data analysis and application services. Furthermore, these services can also be provided as utilities to consumers on a pay-to-access basis. *Darwin* and *2K* are examples of Service Grids.

Our work is mainly concerned with scheduling problems in Computational and Data Grids. From now on only these types will be considered. In the following section we will discuss the scheduling of workflows in Grids.

1.2 Grid Workflow Scheduling

Grid Workflow According to [43], a "workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal". Similarly, scientific experiments are comprised of different applications which run in a specific order on the basis of a defined set of rules, and data needs to be moved among the applications as input and output. Thus, they are referred to as scientific workflows. When a workflow is executed on a Grid, it can be called a Grid Workflow. Some common workflow applications along with the related field of science are as follows:

- EMAN [2], for electron micrograph analysis.
- GridPhyN [4], for physics.
- e-Protein [57], in biotechnology.
- LEAD [6], in the field of weather forecasting.
- WIEN2K [11], for quantum chemistry.
- Montage [7], for astronomy.

Grid Workflow Management Systems According to [43], a Workflow Management System *(WFMS)* is "a system that completely defines, manages and executes workflows

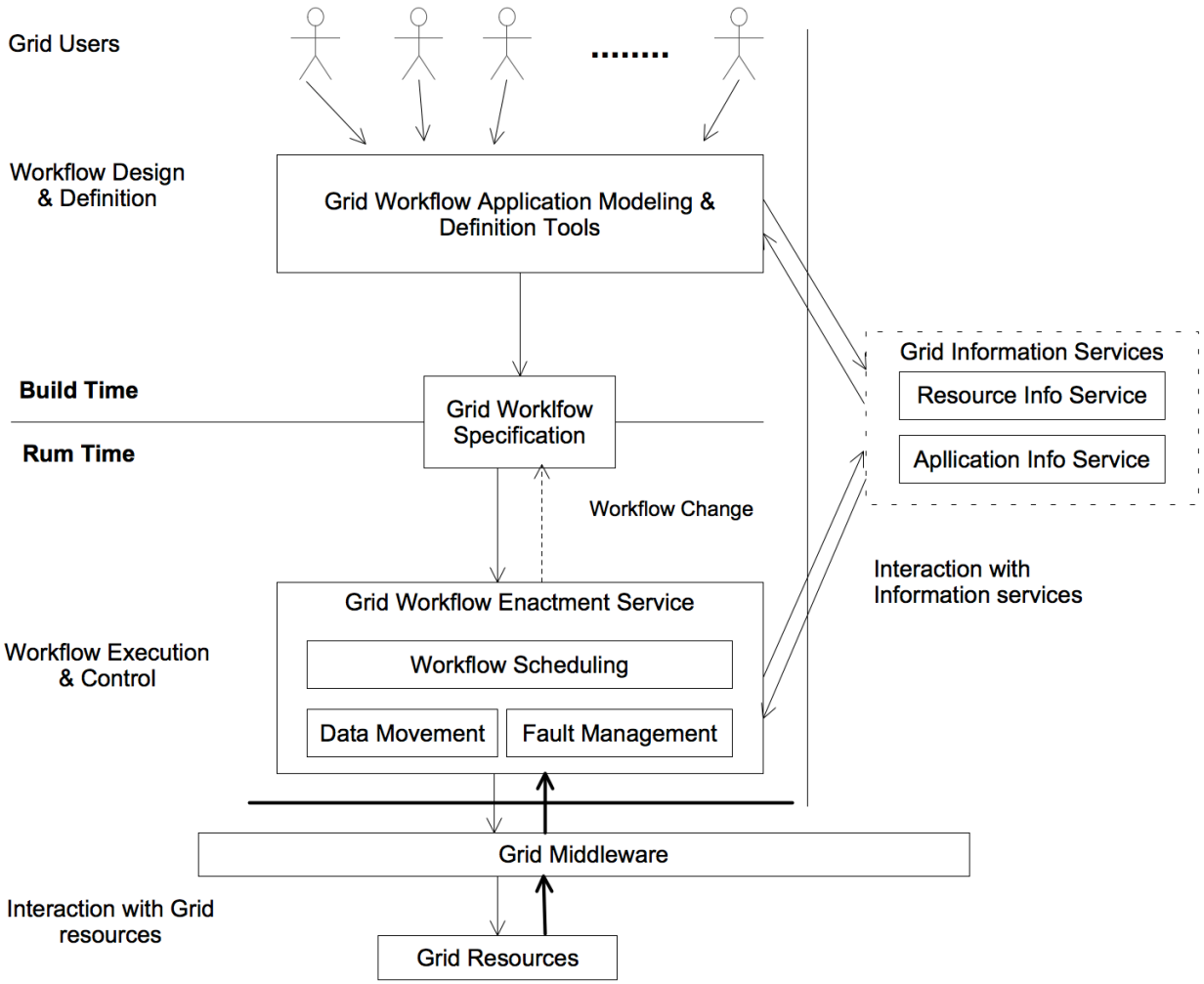


Figure 1.1. Grid Workflow Management System [79].

through the execution of software whose order of execution is driven by a computer representation of the workflow logic". On the basis of this, the architecture and functionalities supported by various components of Grid workflow systems are given in [79] as shown in Figure 1.1. According to Figure 1.1, the WFMS functions can be classified into two high-level categories. One is *Build Time* and the other is *Run Time*. Build time functions are responsible for defining and modelling workflow tasks and their dependencies while run time functions are concerned with the management of workflow execution and interaction with Grid resources for processing workflow application. Workflow modelling tools (build-time) are used for the specification of workflows, which are then submitted to a workflow enactment service (run-time) for execution. A workflow enactment service handles scheduling, fault management and data movement. To obtain services from Grid resources, workflow enactment services act through low-level Grid middleware like

Globus [3] or Unicore [10].

There are many factors that can improve the efficiency of WFMS for the execution of workflows, but scheduling has the most vital role to play for this purpose [79]. Workflow scheduling is the process of selecting a certain available job, for which all dependency requirements are met, from a workflow to be assigned to a resource while satisfying objective(s) requested by the user. Usually, a directed acyclic graph (*DAG*) is used to model a workflow. Workflow/DAG scheduling will be discussed in detail in Section 3.1.

1.3 Advance Reservation in the Grid

There are applications in the Grid which need multiple resources simultaneously. These resources can be computing resources, network resources, storage resources and other instruments. But the availability of these resources is difficult to be guaranteed, which may cause delay in the completion of the application. To guarantee the availability of resources in the above-mentioned scenario and enable simultaneous allocation, Advance Reservation (AR) was introduced [64]. AR enables the resources to be reserved in advance for a user, thus guaranteeing the availability at the requested time. AR is beneficial for workflow applications. The dependent jobs run when all dependency requirements are fulfilled. These dependency requirements can be fulfilled well within time with the help of AR, thus improving the overall completion time of the workflow [62].

There are trade-offs regarding AR implementation in Grid environments. Some of them are as follows. AR is harder to implement as compared to other non-AR techniques. AR may cause ready jobs, with no reservation, in a queue to wait longer, which is not desirable for the owner of the waiting jobs. With AR, fragmentation can occur in the utilisation of system resources, which is not desirable for the resource provider.

A survey regarding the availability and usage of advance reservation in different Grid sites across the globe was presented in [62]. 25 Grid sites participated in the survey. Half of the sites did not support AR and only two sites provided an AR facility without human

intervention. The reasons for this lack of automated AR support mentioned in [62] are as follows:

- Tools lack support for AR.
- The site administrators consider it less important due to its less frequent occurrence.
- The belief that AR will cause under-utilisation of resources.
- The site administrators do not encourage users to make reservations unless it is necessary.

Among these four reasons, three are related to Grid administrators' fears and attitudes towards AR. However, since the publication of [62], a lot of effort has been invested into overcoming the underlying technical issues [62, 66, 46]. Furthermore, increased automation will help to reduce the costs of AR, thus helping to overcome human-centered objections. Together with effective demonstrations that AR does not lead to under-utilisation of resources, increased automation will help to ensure that AR is used more widely, leading to better QoS outcomes for users.

By focusing on the improvement of tool capabilities, automated AR processes can be made available. Varvarigos et al. [70] presented a technique that can help in automating the AR process. They proposed the use of a vector data structure to store the resource utilisation status at a certain time. Each component of the vector represents a minimum time slot for which a resource can be reserved. Utilisation profile maintenance using the vector data structure makes it easier for algorithms to check the availability of resources and to make decisions.

1.4 Optical Burst Switching Network

Optical Burst Switching (OBS) [77] is a network architecture that adopts the best aspects of Optical Circuit Switching (OCS) and Optical Packet Switching (OPS) and avoids their shortcomings. In OCS, a lightpath is established between two nodes using

a specific wavelength. Due to the limited number of wavelengths, not every node can have a dedicated lightpath to all other nodes, which may result in adopting a longer path. Further, OCS may not be a feasible way of transferring smaller data, since the establishment and the releasing of the connection takes several hundred milliseconds. In OPS, the packet is sent along with its header. The optical packet may be buffered at the intermediate nodes while the header is being processed with or without O/E (Optical/Electronic) conversion. The limitations of optical buffering, due to the high-speed optical logic and the optical memory technology required, minimise the benefits of popular electronic system routing, like worm-hole routing method [77], for optical packets. Further, optical packet switching involves greater processing overhead than circuit switching for a data unit. OBS borrows the high bandwidth utilisation, low setup latency and high adaptivity to faults from packet switching, which circuit switching lacks, and adopts low processing per unit data and avoids buffering, which are the features that packet switching lacks.

In an OBS network, large chunks of data are transferred in a burst. An ingress node produces a burst by assembling data packets for the same destination. After a certain number of packets have been assembled or a certain time interval has passed, depending on the technique being followed, a control packet is sent to the egress node to set up a Label Switched Path (LSP) and also reserve bandwidth for the data burst to follow. A request for burst transmission is like a connection requesting an advance reservation with unspecified starting time and specified duration [83]. This approach makes OBS suitable for the AR environment. The promising capabilities of OBS for future Grids as discussed in [56, 51] also attract attention from researchers in the field of Grid Computing.

1.5 Motivation

The potential of the AR environment to provide a better Quality of Service (QoS) to the user for workflow execution drew our attention to it. Unlike the non-AR environment, where the load on a system may cause a job to wait in a queue, resulting in delay in

the completion time of the job, in the AR environment the situation of the resources is clearer, making it possible to produce a better schedule. Our intention was to design a workflow scheduling algorithm that takes advantage of the information about the future utilisation of resources that is available because of AR. Furthermore, we also intended to target data-intensive workflows, in which the communication cost is greater than the computation cost of the jobs. In the case of data-intensive workflows there is a potential for improvement in the schedule by minimising the communication cost between interdependent jobs. It was anticipated that in the AR environment, getting an improvement over existing heuristics will not be a trivial task. It was considered likely that different heuristics will produce similar schedules for a workflow because of AR. This situation motivated us to study the scheduling heuristics in different AR scenarios where they can be misled or distracted from a better schedule, and design a heuristic which can achieve a better schedule in such situations.

A sub-problem of the workflow scheduling is the routing problem. As mentioned in [26], better results can be achieved by jointly scheduling the communication and computation resources in an AR environment. This observation has motivated us to study the AR routing problem as well.

1.6 Contributions

The main objective of our work was to design a workflow scheduling heuristic for AR environments which can minimise the communication cost between the interdependent tasks by scheduling them on the same resource where appropriate, thus minimising the workflow completion time. Following this objective, the contributions we are able to make are as follows:

- We propose a novel resource mapping heuristic for workflow scheduling in the AR environment which conditionally schedules interdependent tasks on the same resource to minimise the communication cost between them to achieve the goal of minimising the completion time of the workflow. The condition is used to set

a limit for the affordable delay in the completion time of a job so that scheduling interdependent tasks on the same resource does not worsen the schedule length. This heuristic is studied in both static and dynamic Grid environments.

A variant of the above-mentioned resource mapping heuristic is proposed for scheduling workflows in the non-AR environment as well.

- A hybrid of multiple workflow scheduling heuristics is proposed in which the heuristics can run in parallel to compute a schedule to minimise the workflow completion time using the advance reservation information. The best of the schedules is selected as the final schedule.
- We introduce a new variation of critical path scheduling that ignores the communication times between dependent tasks under certain conditions in order to predict a critical path that can be used to minimise the workflow completion time.
- We analyse the complexities of different routing problems in the AR environment when the network architecture is OBS. We also propose a variant of a K-shortest paths routing algorithm to find a simple path in an OBS network with earliest arrival time of the data in the AR environment.

The algorithms proposed in this thesis are evaluated and compared with previously proposed scheduling heuristics in simulation experiments for different scenarios and with a number of different workflows.

1.7 Thesis Organisation

The organisation of the thesis is as follows. In Chapter 2, the definitions and terminology is provided which is used throughout the thesis. Previous work related to our chosen problems is surveyed in Chapter 3. In Chapter 4, we present as preliminaries the AR framework and corresponding path calculation technique as well as the dynamic critical path method for job selection, which are employed in the rest of the thesis. Further, the Grid simulator which is used for all the experiments is described as well. Our novel

partner-based resource mapping technique is presented and studied in both static and dynamic Grid AR environments in Chapter 5. In Chapter 6, a hybrid scheduling heuristic for workflow scheduling in the AR environment is proposed and evaluated. A modified version of the partner-based resource mapping technique for non-AR environments is proposed in Chapter 7. The complexity of routing problems in AR environments is studied and a variant of a K-shortest path algorithm to find a path in the AR environment is presented in Chapter 8. Conclusions are given in Chapter 9. The modifications and additions made to the simulator according to our requirements are discussed in Appendix A.

Chapter 2

Definitions and Terminology

This chapter provides formal definitions and terminology related to Grid resources, DAG workflows, workflow scheduling and advance reservation. It also describes the instances of the European Data Grid test bed and the workflows used in the experiments. Some additional definitions will be given in later chapters where they are needed.

2.1 Grid Resource Model

In this thesis, the computational and communication resources of the Grid are modelled as follows. $R = \{r_1, r_2, ..., r_m\}$ is a set of m resources (the terms cluster and resource will be used interchangeably). The resource r_i has W_{r_i} CPUs of the same processing speed. The processing speed in Million Instructions per Second (MIPS) is denoted by PS_{r_i}, where $1 \leq i \leq m$. The owner of the workflow to be scheduled (also referred to as the user) is connected to a special resource r_0, where $r_0 \notin R$. This resource is used to submit jobs and stores the data files required by starting job(s). $L = \{l_1, l_2, ..., l_n\}$ is a set of n links by which resources are connected. C_{l_i} and d_{l_i} denote the capacity and delay in milliseconds of link l_i, for $1 \leq i \leq n$, respectively and both are integer numbers.

For most of the experiments, the network architecture is assumed to be OBS (see Section 1.4). Further aspects of the Grid environment that are related to AR and resource failures will be mentioned in the corresponding sections.

2.2 DAG Workflow

A Grid workflow is represented by a directed acyclic graph (DAG) $G = (V, E)$, where V is the set of jobs and E is the set of edges connecting these jobs. Each edge $e(i, j) \in E$ represents a precedence constraint, meaning that execution of job j (the *child job* or *direct successor* of job i) can only start after the completion of job i (the *parent job* or *direct predecessor* of job j). The size of job j, measured in Million Instructions (MI), is denoted by JS_j, an integer number. Jobs without parents are called *starting jobs*, and jobs without children are called *sink jobs*. For convenience, we introduce for every workflow a single *entry* job (without parents) and a single *exit* job (without children). These jobs have computation length 0. The *entry* job is made the parent job of all the jobs with no predecessor jobs, and the *exit* job is made the child job of all the jobs with no successor jobs. $F = \{f_1, f_2, ..., f_g\}$ is a set of files. The data size of file f in Megabytes is denoted by DS_f, an integer number. Each job requires some integer number h of input data files, where $1 \leq h \leq g$, and generates a single output file, which is added to F. Some files may already exist on a resource before the start of workflow execution. We assume that only the starting jobs take already existing files as input, and other jobs use the output files of their direct predecessors as input. By $PTR_j = \{ptr_1, ptr_2, \ldots, ptr_q\}$ we denote the set of q partner jobs of job j, where jobs are said to be partners if they have at least one common direct successor (child) job. For a given schedule, $RP_j \subset R$ denotes the set of resources on which the q partner jobs of job j are scheduled.

2.3 Workflow Schedule

Given the resource model from Section 2.1 and the workflow model from Section 2.2, workflow scheduling is the process of determining for each job when and on which resource it should be executed, while satisfying the precedence constraints and with the goal of optimising some objective. There can be numerous objectives including

minimising the completion time of all jobs of a workflow, minimising the cost of job execution, finishing all jobs within a deadline and a budget, and efficiently utilising all available resources. In our case we consider the objective of minimising the completion time of a workflow, it will be referred to as the makespan, i.e., the difference between the finishing time of the last job of the workflow and the submission time of the first job of the workflow.

2.4 Scheduling Terminology

We briefly describe some terminology that is relevant for the scheduling algorithms considered in this thesis. The precise meanings of some of the terms may depend on the context or the algorithm in which they are used and will be explained in the respective sections. Many of the terms refer to a specific job or data transfer that will be clear from the context where the terms are used.

Absolute Earliest Starting Time *AEST* refers to a lower bound on the starting time of a job, and Absolute Latest Start Time *ALST* refers to an upper bound. The calculation of *AEST* and *ALST* may vary depending on the algorithm. By $\overline{estET}$ we denote the estimated Execution Time of a job, which may again be calculated differently by different algorithms, for example, one may take the average execution time over all the resources in the Grid. The estimated Execution Time of a job on a particular computing resource is denoted by *estET*. Absolute Data Transfer Time *ADTT* is the data transfer time for the output file of a job, which different algorithms may also calculate differently.

Earliest Transfer Time *ETT* refers to the earliest time when a data file can be transferred from one resource to another. *EAT* is the Earliest Arrival Time of data at a specific resource. *ECT* is the Earliest Completion Time of a job on a resource. *ST* is the execution Start Time and *CT* is the actual Completion Time of a job. $expCT_j^r$ is the expected Completion Time of job j on resource r.

Dynamic Critical Path Length *DCPL* is an estimate for the schedule length of a partially

mapped workflow. The calculation of *DCPL* may vary throughout the thesis. The *makespan* of a workflow is the difference between the submission time of the workflow and the completion time of the last job of the workflow. The *granularity* of a workflow is the average of the ratio of computation and communication cost of the $v - snk$ parent jobs in the workflow, where v is the total number of nodes and $snk \geq 1$ is the number of sink nodes which do not produce output files. The granularity is calculated as follows:

$$Granularity = \frac{1}{v - snk}\left(\sum_{i=1}^{v-snk} \frac{\overline{estET(j_i)}}{ADTT_{j_i}} \right), \tag{2.1}$$

assuming that the jobs j_i for $1 \leq i \leq v - snk$ are the jobs that are not sink jobs. $Comm_{j,pj_i}(r_j, r_{pj_i})$, which may be calculated differently in different contexts, denotes an estimate for the communication time between the ith parent job pj_i of job j and job j if they are allocated to resources r_{pj_i} and r_j, respectively. The ith child job of job j is denoted by cj_i.

2.5 Advance Reservation and Utilisation Profiles

In the AR scenario, we consider the setting that both CPUs on resources and the capacity of links can be reserved in advance. As in [26], we assume that information about reservations that have already been made is stored in *utilisation profiles*. The time limit within which the reservation can be made is $\mathcal{T}$ and it is the size of the utilisation profile as well. We consider two types of AR requests, those for bandwidth and those for CPUs, denoted by *rbw* and *rcpu*, respectively, and both are integers. A request is checked against the utilisation profiles of links or resources. The utilisation profile vector of a link l is denoted by U_l, each component of which stores the bandwidth committed for the connection in future time. The capacity availability profile of a link l at time t is defined by $C_l^t = C_l - U_l^t$, where U_l^t is the bandwidth (an integer value) that has already been committed to connections on link l at time t. For ease of processing reservation data, time is discretized in timeslots of duration τ_l (an integer value). A request for

bandwidth *rbw* is made for a file transfer, and the duration of the requested connection b can be calculated as

$$b = \frac{DS_f}{rbw}.$$

The value of b is always truncated to the integer value. The binary rbw-capacity availability vector $\hat{C}_l(rbw)$, abbreviated *CAV*, has as its k-th entry:

$$\left\{\hat{C}_l(rbw)\right\}_k = \left\{ \begin{array}{ll} 1, & \text{if } C_l - U_l^t \geq rbw \\ & \text{for all } (k\text{-}1)\cdot\tau_l < t \leq k\cdot\tau_l \\ 0, & \text{otherwise} \end{array} \right\},$$

$$\text{for } k = 1, 2, \ldots, z_l$$

where z_l is the dimension of *CAV*. If the requested bandwidth rbw is clear from the context, we omit the argument and simply write $\hat{C}_l$ for $\hat{C}_l(rbw)$. The utilisation profile U_r^t of cluster r is defined as an integer function of time, which records the number of CPUs that have been committed at time t. The cluster availability at time t is $W_r^t = W_r - U_r^t$. The time axis is discretized in steps of duration τ_r (an integer value) and the binary rcpu-cluster availability vector $\hat{W}_r(rcpu)$ is defined as follows:

$$\left\{\hat{W}_r(rcpu)\right\}_k = \left\{ \begin{array}{ll} 1, & \text{if } W_r - U_r^t \geq rcpu \\ & \text{for all } (k\text{-}1)\cdot\tau_r < t \leq k\cdot\tau_r \\ 0, & otherwise \end{array} \right\},$$

$$\text{for } k = 1, 2, \ldots, z_r$$

where z_r is the dimension of $\hat{W}_r(rcpu)$.If the requested CPU $rcpu$ is clear from the context, we omit the argument and simply write $\hat{W}_r$ for $\hat{W}_r(rcpu)$. We assume that τ_l and τ_r are the same for all links and resources.

Sometimes we will consider the subproblem of transmitting a data file over the network to some resource and then executing a job on that resource that needs the data file as input. In that context, we use $P_{n\text{-}d}$ to refer to the set of non-dominated paths (i.e., paths that are not strictly worse than some other path) from the source of the data file to different destination clusters. $PR_{n\text{-}d}$ refers to the set of non-dominated path-cluster pairs $(p\text{-}c)$, where p is a path from the source of the data file to resource c and the

Resource Name (Location)	No. of Nodes	CPU Rating (MIPS)
CERN (Switzerland)	10	1200
Padova (Italy)	13	1000
Bologna (Italy)	20	1140
Catania (Italy)	5	1200
Torino (Italy)	5	1330
Milano (Italy)	7	1000
NIKHEF (Netherlands)	12	1320
Nordugrids (France)	17	1176
Imperial College (UK)	52	1320
RAL (UK)	41	1140
Lyon (France)	12	1320

Table 2.1. EDG testbed resources used for experiments.

earliest completion of the job on cluster c is also taken into account.

2.6 European Data Grid (EDG) Testbed Instance

The European Data Grid (EDG) testbed [16], considered also in [71], is used as the Grid environment in most of the simulations carried out in this work. The testbed has 11 resources as clusters of equally rated CPUs. The CPU rating is given in terms of MIPS. The CPU speed ranges from 1000 to 1330 MIPS. Table 2.1 shows further details of the testbed. In Figure 2.1, we show the modified topology of the network of EDG testbed resources that we use in our simulations. We have added some routers and links to the original topology to make it more challenging to find paths in the AR environment. We have inserted the router *r2* between routers *r1* and *r3*, and inserted the router *r8* between the routers *r9* and *r7*. We added a link between the routers *r2* and *r8* and also added a link between the routers *r8* and *r5*.

2.7 Workflow Instances

In this section we discuss the workflows that are mostly used in the experiments. The workflows shown in Figures 2.2, 2.3 and 2.4 are instances of scheduling problems given in the Project Scheduling Problem Library (PSPLIB) [47]. We refer to these workflows as PSPWF. There are many scheduling problems available in PSPLIB that are used

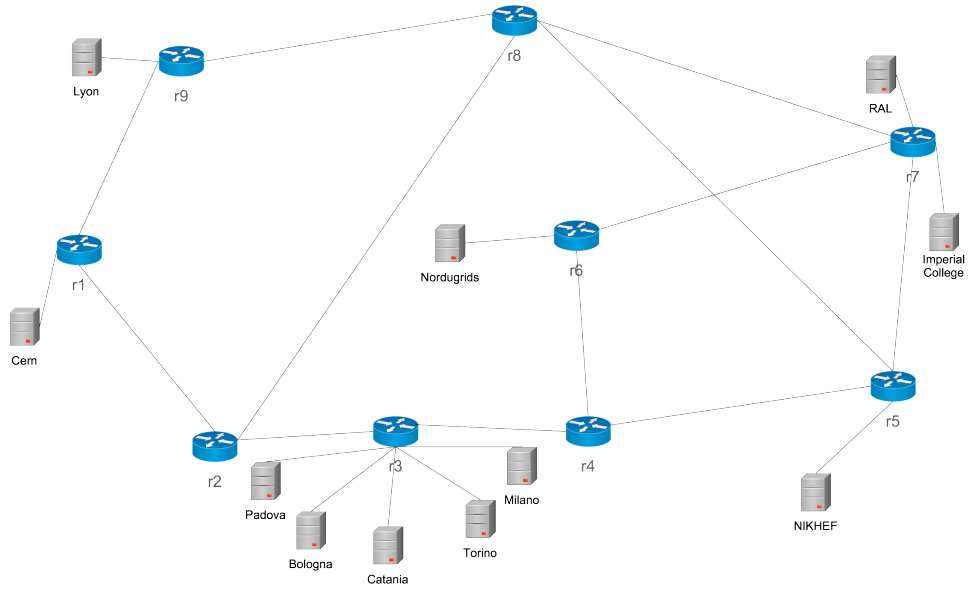
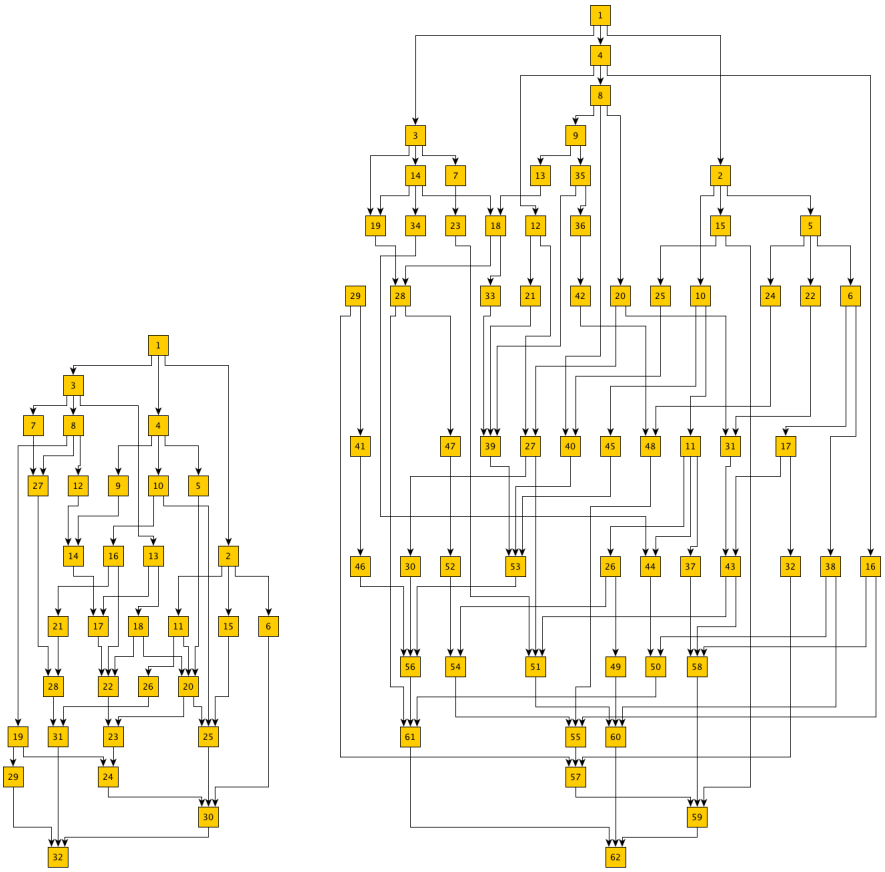


Figure 2.1. Modified network topology of EDG testbed resources.

as benchmarks by researchers to test scheduling heuristics. Furthermore, the chosen instances contain a fair number of jobs with challenging structure for scheduling heuristics. Along with the PSPWF we also used randomly generated workflows in which the number of parent jobs of a single job may vary from two to ten, so that the behaviour of heuristics could be studied for diverse workflows. As instance of a smaller size workflow we chose the eProtein project workflow shown in Figure 2.5, which has a structure that appears complicated enough to test a heuristic in a meaningful way.

To create a specific workflow to be scheduled, each node of the workflow DAG must be assigned information about the corresponding job such as job length, required input files, output files and their sizes. The incoming edges to a node are used to create a list of its required input files. The size of the job and its output file is generated randomly according to the required granularity. For each granularity we set the lower bound and upper bound for the job size in terms of MI. A random value between these bounds is then assigned to the jobs of the workflow. The same procedure is followed to generate the output sizes of the jobs. In all experiments, only one CPU is requested or assigned to each job, and the requested bandwidth for each output file is set to 10 Megabits per second (Mbps) throughout all the AR experiments. For each starting job, i.e., for each job without parent jobs, there exists one input file on resource r_0. The size of this file

is set to 1 MB and the requested bandwidth for this file is 1 Mbps in all AR related experiments.



(a) 30 jobs PSPLIB workflow

(b) 60 jobs PSPLIB workflow

Figure 2.2. Workflows from Project Scheduling Problem Library [47].

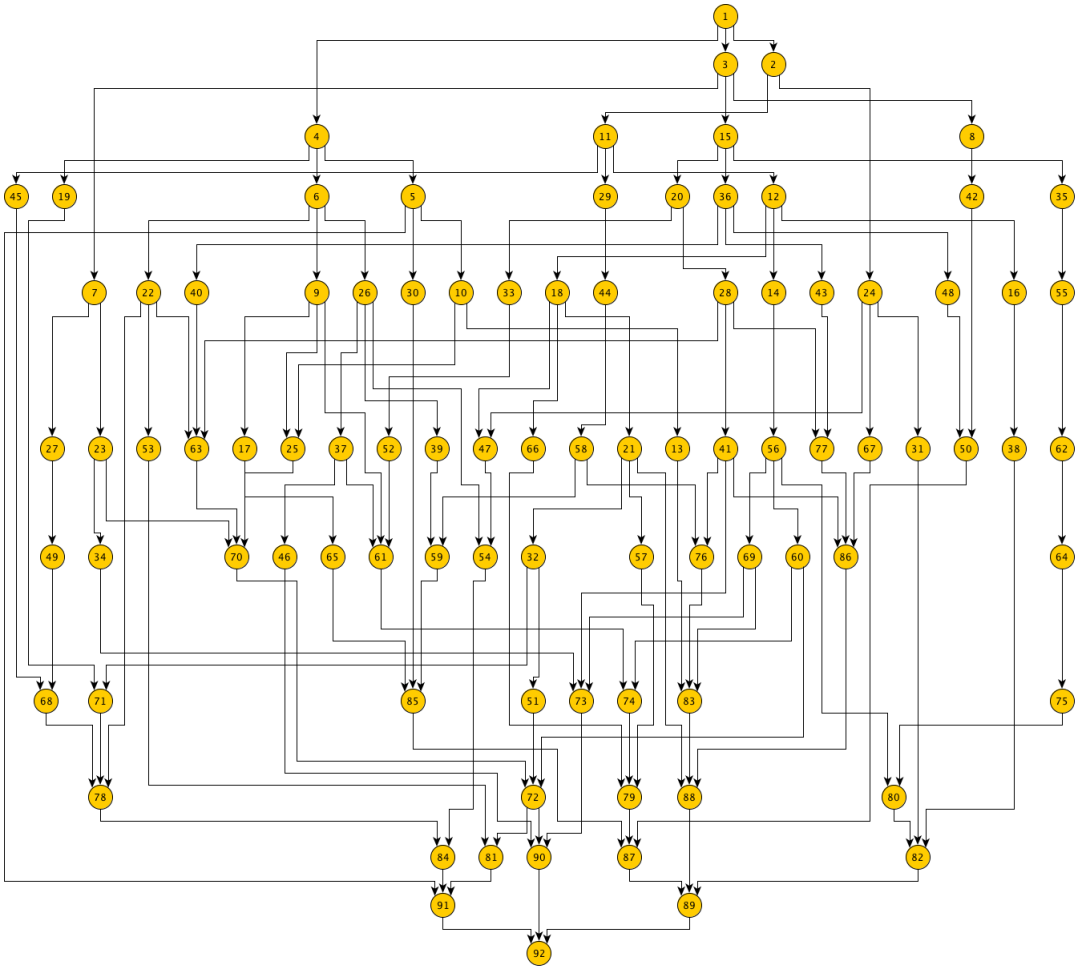


Figure 2.3. 90 jobs workflow from Project Scheduling Problem Library [47].

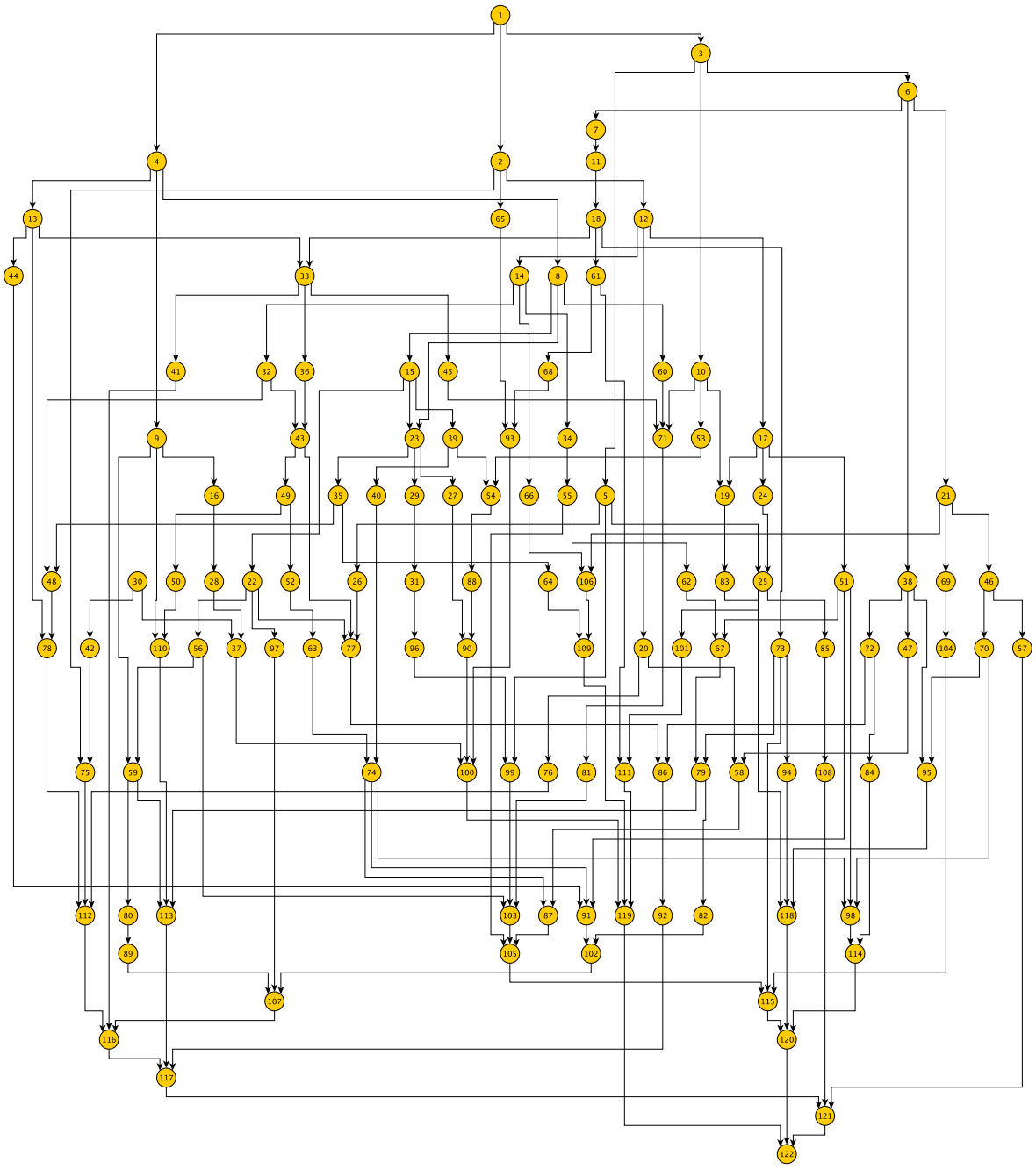


Figure 2.4. 120 jobs workflow from Project Scheduling Problem Library [47].

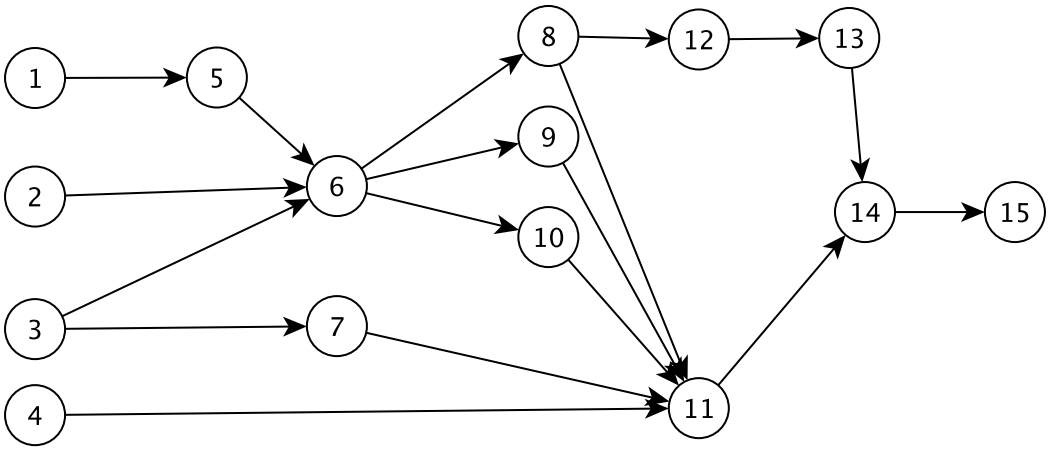


Figure 2.5. DAG of eProtein Project Workflow [57].

Chapter 3

Related Work

3.1 Workflow Scheduling Heuristics

Scheduling a DAG of jobs with unit execution time on a bounded number of processors of the same speed such that each job finishes before a time limit is an NP-complete problem [69]. For this reason, many heuristics have been proposed in this regard. Several heuristics have been proposed for homogeneous multiprocessor environments [49]. In homogeneous multiprocessor environments the architecture and the speed of the processors are the same. Many DAG scheduling heuristics have been proposed for heterogeneous and dynamic environments like Grids as well [13,81,84]. In heterogeneous multiprocessor environments, the architecture and/or the speed of the processors may differ. In many heuristics proposed for DAG scheduling, there are two main stages: the job selection stage and the resource selection stage. Different techniques and concepts were followed for these two stages to develop the heuristics. These heuristics can be divided into six categories [84, 81]:

- *Individual Task Scheduling*
- *List Scheduling*
- *Workflow Based Scheduling*
- *Clustering Based Scheduling*

- *Duplication Based Scheduling*
- *Meta-Heuristics*

In the remainder of this section, we briefly discuss the features of these categories and some of the heuristics that fall into each category. The heuristics that are going to be discussed are heuristics for deterministic scenarios. In a deterministic scenario the dependencies of workflow tasks, their execution time, sizes of input files and output files are known before the scheduling process.

3.1.1 Individual Task Scheduling

Heuristics in this category follow a very simple approach. No preference is assigned to any job. A job is selected randomly from the set of available jobs and sent to the resource which can finish it earliest. This process is repeated until all the jobs in the workflow are scheduled. Myopic [73] is one of the heuristics that falls under this category. This technique neither considers global nor local optimisation of the schedule. Here, by local optimisation we mean optimising the schedule of jobs while considering the set of all available jobs and by global optimisation we mean optimising the schedule of jobs while considering the whole workflow. This technique may yield a smaller schedule computation time but may not give a better schedule since it does not consider the optimisation at any level.

3.1.2 List Scheduling

One of the most widely adopted approaches for DAG scheduling in both homogeneous and heterogeneous systems is list scheduling [39]. This choice is due to its low running time and its tendency to produce good schedules. In this approach, the first task is to prioritise the jobs. The priority of a job may be assigned depending on its computation cost and/or the communication cost between the job and its successor jobs. The calculation of the computation cost and communication cost may vary in different heuristics.

Based on these costs, a rank or priority is assigned to the tasks in a workflow. This rank helps in selecting a task from multiple available tasks. The point that is important here is that in heterogeneous systems, unlike homogeneous systems, the capacity of the computational resources and the communication links between them varies, which affects the expected completion time of a task and the transmission time of data among tasks. Therefore, different approximate calculation methods are used to estimate the computation and communication costs. After a job is selected, it is sent to a resource which satisfies any objective(s) under consideration (e.g., having earliest completion time and/or satisfying budget constraints).

There are different ranking and resource selection methods which further divide list scheduling into four categories. These categories are discussed in the following.

Static Priority Based

For heuristics in this category, the process of assigning priorities to all the tasks in a workflow is carried out only once before scheduling. These priorities remain static throughout the remaining scheduling process. Some of the heuristics included in this category are Hybrid Balanced Minimum Completion Time (HBMCT) [60], Mapping Heuristic (MH) [30], Opportunistic Load Balancing (OLB) [36], Minimum Execution Time (MET) [23] and Heterogeneous Earliest Finish Time (HEFT) [68]. We discuss HEFT in more detail, as it is a well known scheduling heuristic and considered as a benchmark.

HEFT: HEFT uses an upward rank technique for assigning ranks to the jobs. In the upward rank technique, the workflow is traversed in upward direction and the rank assigned to a job based on its average execution time, the average communication time to its child jobs and the ranks of its child jobs. The rank of a job gives the critical path length from the job to the last job of the workflow. The purpose of assigning the rank to a job is to not only consider the available jobs while scheduling but also consider the impact of the scheduling of the job on future jobs. For scheduling purposes, jobs

are sorted in descending order of their rank values. Following this order, each job is assigned to a resource which is expected to finish it earliest. HEFT also considers the insertion of a job into the earliest available idle space between two already scheduled jobs. In [68], HEFT is compared with list heuristics, including MH and CPOP. The considered resource model in [68] contained heterogeneous computation resource with single processor, and different workflows, with varying degree of parallelism and depth, are generated for the test. HEFT outperformed all of the compared heuristic in terms of the ratio of obtained makespan and lower bound of the makespan. HEFT is also shown to be more cost effective with respect to the schedule computation time than all the other heuristics.

Even though HEFT has a very good reputation, in [58] it is shown to be outperformed by DCPG in scenarios where communication costs are greater than the computation costs when the workflow structures are parallel, fork-join and randomly generated and the resource model comprises of heterogeneous resources with multiple processors. HEFT is also shown to be outperformed by the heuristic Longest Dynamic Critical Path (LDCP) in case of greater communication cost than computation cost for some random and regular workflows from real world applications [29]. A similar worsening behaviour of HEFT against a duplication based heuristic for workflows with greater communication cost was noted in [15] as well. The higher communication cost in a workflow can greatly be reduced by assigning the interdependent jobs to the same resource, thus it can change the criticality of the jobs and may affect the makespan of the workflow.

Dynamic Priority Based

For heuristics in this category, the priority of a task is computed when it becomes available for scheduling. The priority of a task may change during the scheduling procedure. Some of the heuristics included in this category are Min-min [44], Max-min [44] and Duplex [23]. We discuss Min-min and Max-min next.

Min-min and Max-min: The Min-min algorithm gives priority to the smallest available tasks for scheduling. The expected completion times of available jobs are calculated on all resources to find their minimum expected completion time. The job with minimum expected completion time is given priority and scheduled on the resource which is expected to give it the minimum completion time. The same process is repeated at each iteration until all jobs in a workflow are scheduled. By giving priorities to the smaller jobs, the algorithm increases the chances of high throughput.

The difference between Min-min and Max-min is that jobs with longer execution time are given priority rather than the smaller jobs. For all unmapped jobs, the minimum expected completion time is calculated and the job with the longest completion time is scheduled on the resource which is expected to finish it earliest. In cases when there exist jobs with longer execution than many other jobs, the Min-min technique will cause the longer jobs to wait longer and run them when all the small jobs are executed. In such cases, the Max-min technique first maps the longer jobs, and then the other, smaller jobs can be run in parallel, which can improve the makespan.

Even though these techniques consider multiple jobs for the mapping decision, which works well for independent sets of jobs [23], in case of workflows these techniques do not take into account the dependent jobs, which may lead to a poor workflow makespan as compared to techniques that consider the dependent jobs as well, i.e., HEFT, DCPG [58, 68].

Critical Path Based

This category of heuristic computes a critical path based on the computation and communication cost of the jobs. At the time of selection, priority is given to the most critical job, the one which is on the critical path. In case no critical path job is available, the job which is considered most critical after that by the heuristic is chosen for scheduling. The idea behind the critical path technique is to give priority to the set of jobs which collectively may give a lower bound on the workflow makespan. Some of the heuristics

included in this category are Critical Path on Processor (CPOP), Modified Critical Path (MCP) [74], Dynamic Critical Path (DCP) [50] and Dynamic Critical Path for Grid [58]. There are heuristics in which the critical path remains the same throughout the scheduling process after it is computed in the beginning, e.g., CPOP, and heuristics where the critical path keeps changing after each scheduling step, like DCP and DCPG. As our work is closely related to DCPG, we have already discussed it in detail in Section 4.2. Here we discuss CPOP.

CPOP: CPOP computes upward and downward ranks, as done by HEFT, of all the jobs in a workflow. The maximum of the sum of the upward and downward ranks of a job gives the length of the critical path of the workflow, and all the jobs on the critical path have the same sum value. A processor is chosen for the critical path jobs, which is called *critical path processor*, to minimise the cumulative computation costs of the jobs on the critical path. If a critical path job is to be scheduled, it is scheduled on the critical path processor. For other jobs, the resource which is expected to finish it earliest is assigned to the job. In [68], it is shown that the CPOP stands second to the HEFT for the average of the ratio of obtained makespan and lower bound of the makespan when the communication cost of the workflow is greater its computation cost, the experiment settings are the same as discussed for the HEFT in the previous section. It showed poor performance as compared to the HEFT and two other heuristics for the same metric when the computation cost of the workflow is greater than its communication cost.

3.1.3 Workflow Based Scheduling

In this approach the schedule is optimised globally, i.e., at the workflow level [20]. Initially, multiple schedules for the complete workflow on the available resources are computed and one of them is chosen as a final schedule. In case of any changes in the environment, the jobs are reallocated. As shown in [20], this approach works very well in situations in which communication costs are higher than the computation costs in a workflow. This technique minimises the communication cost by scheduling the interdependent jobs on

the same resource, thus minimising the workflow makespan. But the better results come at the cost of higher running time for schedule computation, which may not make it a better choice if many workflows are being scheduled and quick schedule computation is needed.

3.1.4 Cluster Based Scheduling

The emphasis of this technique is to minimise the communication cost between interdependent jobs so that the makespan can be minimised. The approach adopted in [27] can be briefly explained as follows. Job clusters are constructed based on the dependencies between jobs and their communication and computation costs. In the next step, clusters may be merged conditionally. Clusters of resources are also built. The clusters of jobs are sorted in descending order with respect to their communication and computation costs. The clusters of resources are sorted in descending order with respect to their bandwidth capacity and computation capacity. Following the order, each cluster of jobs is allocated to a cluster of resources until the load limit is reached. Even though an improvement in the workflow makespan can be achieved by minimising the communication cost in this way, this approach incurs a computational overhead to achieve this.

3.1.5 Duplication Based Heuristics

In this technique the tasks are duplicated and run redundantly on multiple machines to minimise the communication cost between dependent jobs. The duplication is performed after an initial schedule is computed. Afterwards it is decided which task to duplicate on which resource. This technique may reduce the communication cost of the workflow but at the cost of execution overhead. This is suitable for situations where communication costs are high as compared to the computation costs and execution overhead can be afforded. In [40] a duplication based heuristic is proposed to solve the DAG scheduling problem in multiprocessor environments.

3.1.6 Meta-heuristics

Meta-heuristics provide both a general structure and strategy guidelines for developing a heuristic for solving computational problems. They are generally applied to large and complicated problems. They help in finding good solutions in a large solution space. Many meta-heuristics have been applied for solving workflow scheduling problems, including GRASP, Genetic Algorithms and Simulated Annealing.

Greedy Randomized Adaptive Search Procedure (GRASP): This is a search technique in which a scheduling solution is optimised in iterations [59]. For workflow scheduling, in each iteration a workflow schedule is computed based on the greedy approach [20]. Then the local search technique is used to find a better solution by swapping the allocation of jobs on the resources in the current workflow schedule. The best solution is updated if the new solution is a better one. The process stops when the maximum number of iterations are reached.

Genetic Algorithms (GAs): In Genetic Algorithms [38], the principle of evolution is used to find a better solution from a large search space. An initial population of randomly generated solutions is created. An individual solution is called a chromosome. Genetic operators including selection, crossover and mutation are applied to the chromosomes to generate new offspring (new solutions). The quality of offspring is measured using a fitness function. The best individuals are selected to be carried over to the next generation, and the same steps of applying operators are repeated. Later in the procedure, the fittest individuals from older generations are exploited and new generations of individuals are explored. The evolution process is repeated until a certain condition, like the number of iterations or a certain level of fitness value, is satisfied. GAs have been adopted for independent and inter-dependent tasks scheduling in homogeneous and heterogeneous environments. Some GAs for the DAG scheduling problem in a heterogeneous environment are proposed in [72, 63, 80]. In [72] and [63], the workflow schedule is the chromosome and the fitness value is the completion time of the workflow. In [80],

a budget constraint is also considered for the fitness of the schedule. The results show that a GA can find a better schedule as compared to the list scheduling heuristics but at the cost of execution time. For smaller workflows with certain assumptions, GA can give a schedule near the optimal schedule within polynomial time [63].

Simulated Annealing (SA): Simulated Annealing (SA) [53] is derived from the repeated process of heating and cooling down material such as glass or metal to remove internal stresses and toughen it. At the start, an initial solution is provided to a typical SA algorithm by assigning tasks randomly to resources. The temperature is then decreased at a specific rate. At each temperature level, multiple iterations are executed, and at each iteration a new solution is produced by applying random changes to the current solution. The new solution is definitely accepted if it is better than the current solution and with a certain probability (depending on the temperature) if it is worse than the current solution. At higher temperature the probability of the acceptance of a worse solution is also higher and the acceptance probability decreases as the temperature decreases. After a certain number of iterations, the temperature is decreased and the same process is repeated until the lower boundary of the temperature is reached. In [21], SA is shown to surpass many solutions to the benchmark project scheduling problems given in the Project Scheduling Problem Library [47] in terms of earliest completion time. For Grid workflow scheduling SA, is adopted in [18, 78, 76].

3.2 Advance Reservation

For the automation of AR in Grid environments Varvarigos et al. [70] proposed a mechanism to manage the utilisation information of network links over a period of time. This utilisation information is used to find the availabilities of resources in future time slots. The resource utilisation information is converted into binary information after checking it against the user's requested capacity. Based on the binary information of each network link, a new technique is proposed to find the availability of a path to a destination. This

AR technique is extended for the joint scheduling of communication and computation resources in [26] by adopting the same technique of maintaining and manipulating the utilisation information of the computing resources. It is shown that the joint scheduling gains improvements over separate scheduling for CPU and data intensive tasks in terms of earliest completion time.

3.3 Rescheduling in Failure Cases

The Grid is a dynamic environment in which the performance of resources fluctuates. The fluctuation could be because of resource failures, software faults or load from users. It also happens that resources may join or leave the Grid resource pool which if considered in time by scheduling algorithms can affect the performance of submitted/ready jobs. To enhance the performance of the Grid, fault detection and rescheduling techniques are developed.

To detect faults, push and pull methods are used [55]. In the push model, Grid components send messages to the failure detector component regarding liveness, while in the pull model the failure detector requests the status of liveness from Grid components. Fault tolerance approaches can be categorised into pro-active and post-active techniques [55]. Pro-active techniques consider the failure possibilities in the Grid before a scheduling decision is made. Post-active techniques, like *rescheduling*, take decisions after the occurrence of a failure. *Rescheduling* is the process of reallocating jobs after a failure occurrence. Which jobs are reallocated depends on the technique under consideration.

Rescheduling techniques for independent tasks are proposed in [19, 75]. For workflow applications, rescheduling mechanisms are given in [61, 82]. In [19], two rescheduling techniques are proposed: One technique stops and restarts the job and the other technique handles the job swapping for rescheduling. A self adaptive scheduling technique is proposed in [75]. Whenever an abnormal behaviour is detected, using an error threshold prediction in job processing, the affected job is rescheduled. The affected job is migrated

to the processor which gives the minimum completion time.

In [61], a selective rescheduling mechanism for rescheduling workflow tasks is proposed. The purpose of being selective is to lower the cost of rescheduling. Only those tasks are rescheduled whose difference between the expected actual start time and the expected start time of the initial schedule is larger than a maximum allowable delay. In [82], a HEFT based adaptive rescheduling mechanism is introduced. When a performance change event is triggered, the scheduler reassigns the tasks to the resources considering earliest finish time.

The above-mentioned rescheduling techniques were proposed for non-AR environments. For AR environments, a rescheduling policy for workflows was proposed in [24]. The approach is different from non-AR environments because not only currently running jobs are considered for rescheduling but also jobs which are scheduled on the faulty resources in the near future. A problem that arises here is the estimation of the length of the downtime, for which an adaptive dynamic load based approach is used. It is claimed that the adopted technique avoids over- or under-estimation of downtime, so that only jobs with a chance of abnormal termination are remapped.

3.4 Routing in Advance Reservation Environments

Advance reservation of bandwidth was initially considered for video conferencing between two studios to make sure its adequate availability. Similarly, advance reservation was needed for the use of transponders of satellite systems [41]. The main focus for such circuit-switched systems was on traffic modelling and call admission control. Later on, advance reservation related proposed work for circuit-switched systems was modified for packet switching networks.

In [41], the authors highlighted the importance of spatial and temporal routing algorithms in the AR environment. The impact of AR on path selection and the related computational complexity are discussed. It is shown that different requirements yield

different computational complexity. In one case, they considered a fixed start time for the connection, a requested bandwidth and a data transfer duration with the objective to find a simple path which has a bandwidth more than the requested bandwidth for the transfer duration from a single source to a single destination so that data can arrive before a defined time limit. The algorithm proposed for this scenario gave the computation complexity $\mathcal{O}(M \cdot \mathcal{D})$, where M is the total number of edges in the network and $\mathcal{D}$ is the duration of data transfer. In another case they made the starting time flexible and set the objective to find a simple path with the earliest starting time for the requested bandwidth. The proposed algorithm for this scenario has the computation complexity $\mathcal{O}(M \cdot (\mathcal{T} - \mathcal{D}))$, where $\mathcal{T}$ is the time limit. They also discussed a problem in which the requested bandwidth value is not specified and the objective is to find a simple path with the maximum available bandwidth for the data transfer. They proved this problem to be NP-hard. For these AR routing problems, they assumed that routing nodes can buffer the data, thus data that is transferred can wait at intermediate nodes. An important factor pointed out by the authors related to the computational complexity is the granularity of time slots for which a reservation is made. Higher granularity reduces the computational complexity at the cost of quality, because the allocated time slots may exceed the need. Thus, there exists a trade-off between performance and computational complexity.

In [41], the effect of the delay of links on the AR information has not been discussed. Later, in [70], it was pointed out that the delay of links may make some of the availability information outdated. Unlike [41], in [70] the authors considered the AR routing problem in OBS networks, in which routing nodes do not buffer data and the data flows from the source node to the destination node without waiting at intermediate nodes. Furthermore, it is discussed in [70] that the polynomial time solvable AR routing problems mentioned in [41] when considered in OBS networks can become intractable if the unit of the delay of the links and the smallest unit of AR are kept the same. Due to the intractability of the problem, they proposed three polynomial-time multicost routing algorithms to find a simple path with earliest arrival time, when the bandwidth for the data transfer is

specified. They introduced a technique to eliminate the expired information about the availability of links caused by the delay of links. As stated by the authors in [70], the polynomial time algorithms do not guarantee to give the optimal path.

In Chapter 8, we consider the routing problem addressed in [70] and propose a K-shortest path variant for the problem that follows the same path calculation method as given in [70]. We also present scenarios in which the polynomial time algorithms proposed in [70] may miss a better path while our K-shortest path variant may succeed in finding the better path.

Chapter 4

AR Framework, DCPG Scheduling and GridSim

This chapter discusses two techniques that have been introduced in previous work and that we have adopted throughout our work: The AR framework proposed by Varvarigos et al. [70] and Christodoulopoulos et al. [26] is explained in detail in Section 4.1, and the Dynamic Critical Path calculation technique proposed in [58] is presented in Section 4.2. We also explain the working mechanism of the Grid simulator GridSim [67] in the Section 4.3.

4.1 Resource Availability in Advance Reservation Environments

In AR environments, users can request for a resource to be reserved over a period of time. The request is checked against the future utilisation of the resource. A resource that is available for the requested period of time for the requested capacity is reserved for the user. Here, capacity can refer to the number of CPUs for computing resources or the bandwidth for network links. Varvarigos et al. [70] proposed a specific mechanism for maintaining resource utilisation information. A utilisation profile is maintained for

each resource, such as a network link or a computing resource, and updated whenever a new reservation is made. The data structure they use for the utilisation profile is a vector, which makes it easier for algorithms to check for available capacity. They also proposed a multicost routing and scheduling algorithm for advance reservation of network connections which is directly applicable to OBS networks. Christodoulopoulos et al. [26] extended this algorithm to take into account the resource availability on the destination cluster. We refer to their extension of the multicost routing and scheduling algorithm as *MRS* in the following. There are two major aspects of *MRS* to be discussed: The first is how the *CAV* (see Section 2.5) of a path in the network is calculated, which is explained in Section 4.1.1, and the other is how a path to a resource/cluster is selected, which is discussed in Section 4.1.2.

4.1.1 Path CAV Calculation

We discuss the calculation of the *CAV* of a path in the context of the example shown in Figure 4.1. The example shows a network with three nodes *S*, *I* and *D*. The node *S* is connected with the node *I* and the node *I* is connected with the node *D*. The delay of the link *SI*, measured in units of τ_l, is 2 and the delay of the link *ID* is 4. Considering the given *CAVs* of the links, the objective is to find the earliest arrival time of the data of duration 3 from the node *S* to the node *D*.

The routing and resource selection decision is made on node *S*, therefore the *CAVs* of all links and resources are made available at regular intervals to *S*. It is assumed that a data file is located on *S* and we want to transfer it to resource *D*, where it will be used as an input for a job execution. The *CAV* of a path is calculated by combining the *CAVs* of all the links in the path using the associative operator $\oplus$ (see (4.1) and Figure 4.1b). The *CAV* of path p_{SID} comprising of links *SI* and *ID* with delays $d_{SI} = 2$ and $d_{ID} = 4$, respectively, is calculated as follows:

$$\hat{C}_{SID} = \hat{C}_{SI} \oplus \hat{C}_{ID} = \hat{C}_{SI} \,\&\, LSH_{2 \cdot d_{SI}}(\hat{C}_{ID}), \tag{4.1}$$

where $\hat{C}_{SI}$ and $\hat{C}_{ID}$ are the *CAVs* of links *SI* and *ID*, respectively. $LSH_{2 \cdot d_{SI}}(\hat{C}_{ID})$ is the left shifting by $2 \cdot d_{SI}$ elements of $\hat{C}_{ID}$, that is 2 times the propagation delay of link *SI* measured in τ_l-time units. The first left shift is to eliminate the first d_{SI} elements from $\hat{C}_{ID}$ that have expired because of the delay of the link while transferring information from node *I* to *S*. The second left shift is to eliminate d_{SI} further elements, which corresponds to the delay the burst would incur when transmitted from *S* to *I*. It is assumed that the propagation delay in both directions is the same. The next step is to apply the bitwise AND operation $\&$ on the *CAVs* of *SI* and *ID* to get the binary availability vector of path p_{SID}. Let $b = 3$, then we have

$$ETT(p_{SID}, b) = 0,$$

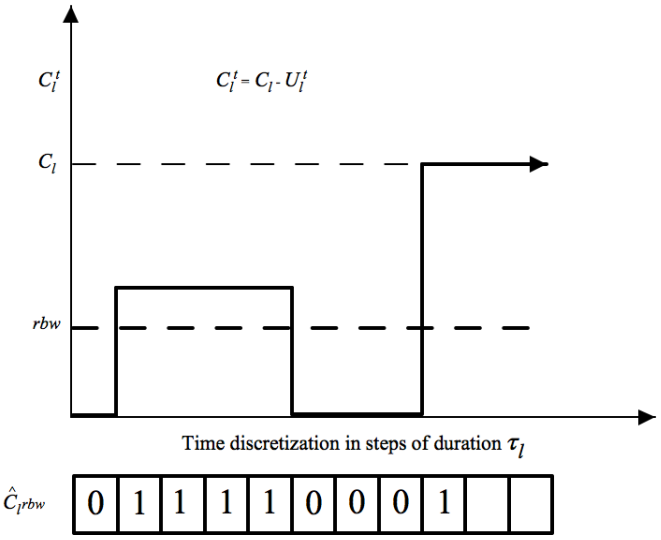
i.e., the first possible time slot at which $\hat{C}_{SID}$ has b consecutive ones is 0. In other words, the first time at which the transfer of data over path p_{SID} can start is time 0. The earliest arrival time of the data at cluster *D* over path p_{SID} is then

$$EAT(p_{SID}, b) = ETT(p_{SID}, b) + b + d_{p_{SID}} = 9.$$

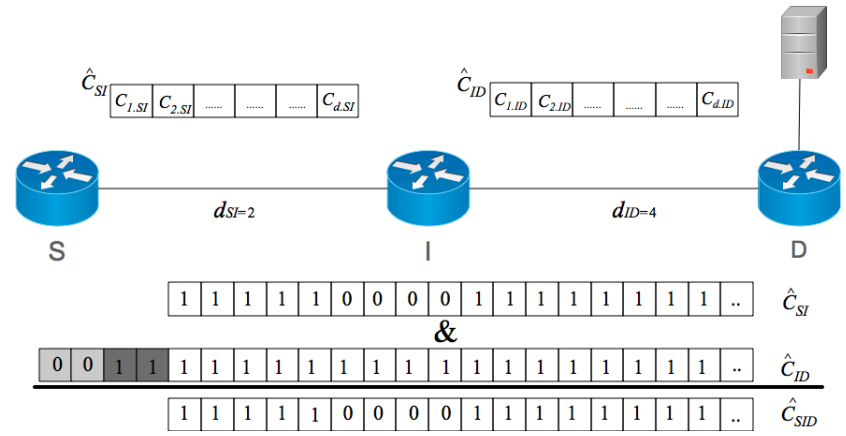
So far, the calculations are only concerned with routing and yield the earliest arrival time of data at *D*. Now, following [26], we consider additionally the resource availability of *D* after the arrival of the data. For this, the irrelevant information about the availability of *D* before the arrival of the data is removed from $\hat{W}_D$. To do this, an operation $MUV_k(\hat{W}_D)$ (Make UnaVailable) is performed, which makes the first k elements of vector $\hat{W}_D$ unavailable by setting them to 0. In this case, k is set to $EAT(p_{SID}, b) = 9$, and the new *CAV* for *D* becomes

$$\hat{W}_D(p_{SID}, b) = MUV_{EAT(p_{SID},b)}(\hat{W}_D) = MUV_9(\hat{W}_D) \text{ (see Figure 4.1c).}$$

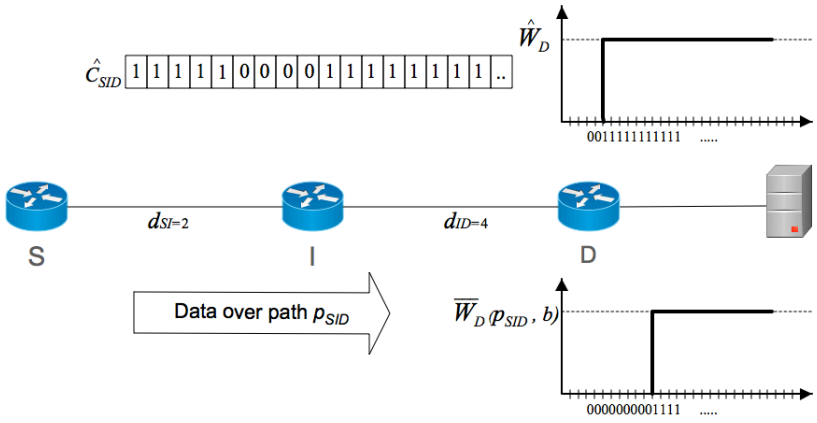
Thus the new vector gives the availability of *D* after the data arrives by following the path P_{SID}.



(a) The capacity profile C_l^t and the binary rbw-capacity availability vector $\hat{C}_{l^{rbw}}$ of a link l of capacity C_l. Similarly cluster availability profile W_r^t and the binary rcpu-capacity availability vector $\hat{W}_r^t$ is defined [70].



(b) Calculation of the path capacity availability vector $\hat{C}_{SID}$. Before applying the AND operation, $\hat{C}_{ID}$ is left-shifted twice by d_{SI} τ_l-time units [70].



(c) $b = 3$ is the data transfer duration. $EAT(p_{SID}, b)$ is the earliest arrival time of data at cluster D over path p_{SID}. $\overline{W}_D(p_{SID}, b)$ is the cluster availability vector after $EAT(p_{SID}, b)$. $\overline{W}_D(p_{SID}, b)$ is calculated by putting 0's into the first $EAT(p_{SID}, b) = 9$ elements [26].

Figure 4.1. Calculation of the path capacity availability vector $\hat{C}_{SID}$, and finding the first available place at cluster D.

4.1.2 Non-Dominated Set of Path and Cluster Pairs

Following the method for the calculation of the CAV of a path described in the previous section, a set of non-dominated paths $P_{n\text{-}d}$ to all resources is computed by *MRS*. A vector V_l for each link $l \in L$ is defined as follows:

$$V_l = (d_l, \hat{C}_l) = (d_l, \langle \hat{C}_{1,l}, \hat{C}_{2,l}, ..., \hat{C}_{z_l,l} \rangle).$$

The cost vector of path p is defined as

$$V_p = \underset{l \in p}{\odot} V_l \stackrel{def}{=} \left(\sum_{l \in p} d_l, \underset{l \in p}{\oplus} \hat{C}_l \right), \tag{4.2}$$

where $\oplus$ is the associative operator defined in (4.1). Based on the cost vectors of paths, a domination relation between paths with the same destination is defined as follows. Path p_1 dominates path p_2 if the delay of path p_1 is less than the delay of p_2 and p_1 is available at all times at which p_2 is available for the considered connection and source-destination pair. This is formally defined as

$$\begin{aligned} &p_1 \text{ dominates } p_2 (\text{notation: } p_1 > p_2) \text{ iff} \\ &\sum_{l \in p_1} d_l < \sum_{l \in p_2} d_l \text{ and } \underset{l \in p_1}{\oplus} \hat{C}_l \geq \underset{l \in p_2}{\oplus} \hat{C}_l \end{aligned} \tag{4.3}$$

where the inequality $\geq$ of vectors should be interpreted element-wise. $P_{n\text{-}d}$ is a set of paths such that no path in $P_{n\text{-}d}$ is dominated by any other path in $P_{n\text{-}d}$ [70]. After the set of non-dominated paths from a source to multiple destinations has been calculated, the set of non-dominated path-cluster pairs (p-c pairs) is computed [26]. The cost vector for a p-c pair for path p and its destination cluster r is defined as

$$\begin{aligned} V_{pr} &= \left(V_p, \overline{W}_r(p, b) \right) \\ &= \left(\sum_{l \in p} d_l, \underset{l \in p}{\oplus} \hat{C}_l, \overline{W}_r(p, b) \right), \end{aligned} \tag{4.4}$$

where $\overline{W}_r(p,b) = \mathit{MUV}_{\mathit{EAT}(p,b)}(\hat{W}_r)$ is the binary cluster availability vector as described in the previous section. The p-c pair p_1r_1 dominates p_2r_2 if p_1 dominates p_2 (see (4.3)) and cluster r_1 is available for job execution after the arrival of input data over path p_1 at least at all times the cluster r_2 is available for execution of the same job after the arrival of input data over path p_2. This can be formally described as:

$$p_1r_1 \text{ dominates } p_2r_2 \text{ (notation: } p_1r_1 > p_2r_2\text{) iff}$$
$$p_1 > p_2 \text{ and } \overline{W}_{r_1}(p_1,b) \geq \overline{W}_{r_2}(p_2,b), \tag{4.5}$$

where vector inequality $\geq$ is interpreted component-wise. The set of non-dominated p-c pairs $PR_{n\text{-}d}$ is computed by applying (4.5) to the set $P_{n\text{-}d}$. There is no pair in $PR_{n\text{-}d}$ that dominates another and $PR_{n\text{-}d} \subseteq P_{n\text{-}d}$. The optimal p-c pair for a given optimisation criterion can then be computed by evaluating the optimisation criterion on all p-c pairs in the set $PR_{n\text{-}d}$.

A drawback of *MRS*, as mentioned by its authors, is that the number of non-dominated paths can be exponential. As the algorithm is not guaranteed to run in polynomial time, two alternative polynomial-time algorithms have also been proposed, which compute a set of *non-pseudo-dominated* paths. One of the algorithms, which we refer to as *AW*, considers the number of $1s$ in the *CAV* of a path. For a link l, $w_l = weight(\hat{C}_l)$ denotes the number of $1s$ in its *CAV*. The pseudo-domination relation for the path pruning is defined as:

$$p_1 \text{ pseudo-dominates } p_2 \text{ (notation: } p_1 >_{ps} p_2\text{) iff}$$
$$\sum_{l\in p_1} d_l < \sum_{l\in p_2} d_l \text{ and } weight(\bigoplus_{l\in p_1} \hat{C}_l) \geq weight(\bigoplus_{l\in p_2} \hat{C}_l) \tag{4.6}$$

That is, the path p_1 pseudo-dominates path p_2 if the delay of p_1 is less than the delay of p_2 and the weight of the *CAV* of p_1 is greater or equal to the weight of the *CAV* of p_2.

The metric used for pseudo-domination in the second algorithm is the number of times at which a sequence of consecutive $1s$ equal to the length of the requested connection

duration is available. This algorithm will be referred to as *CS*. For link l we denote by $L_l(b, \hat{C}_l)$ the total number of runs of consecutive $1s$ in $\hat{C}_l$ that have length equal to the connection duration b. The pseudo-domination relation for the path pruning is defined as:

$$p_1 \text{ pseudo-dominates } p_2 \text{ (notation: } p_1 >_{ps} p_2\text{) iff}$$
$$\sum_{l \in p_1} d_l < \sum_{l \in p_2} d_l \text{ and } L(b, \bigoplus_{l \in p_1} \hat{C}_l) \geq L(b, \bigoplus_{l \in p_2} \hat{C}_l) \tag{4.7}$$

An example for the calculation of the weight and the total number of runs of consecutive $1s$ in a *CAV* can be given as follows. Given the *CAV*

$$\hat{C}_l = (00111101001100011100011), \tag{4.8}$$

we have $w_l = 12$, $L_l(3, \hat{C}_l) = 3$, and $L_l(2, \hat{C}_l) = 7$.

The polynomial-time algorithms resulting from the use of non-pseudo-dominated paths are heuristics that do not guarantee to find an optimal path. After we present the pseudocode for the algorithm proposed in [70], the computational complexity will be discussed.

Notation

The network is defined as a directed graph $G = (N, M)$: N is the set of n nodes, and M is the set of m links. The connection request is of size b in slots and has source node $S \in N$ and destination node $D \in N$. V_m is the cost vector of link $m \in M$. Each path p is represented by a label that includes the cost vector V_p associated with it and the first hop to the source using that path. L_i is the set of labels of the paths from node S to a node $i \in N$, and $L = \cup_{i \neq S} L_i$ is the set of all labels. $L^f \subset L$ is the subset of all final labels for all the nodes, and L_i^f is the set of final labels for the node i.

Algorithm 4.1 Computing set of non-dominated paths to all nodes in a network.

function ComputeSetofNonDominatedPaths($G, S, V_m$of all links)
  Initialization(G, S, V_m)
  while $L \neq L^f$ **do**
    i = ChooseOptimumLabel(L, L^f)
    ObtainNewLabelsAndDiscardDominatedPaths(G, i, L, V_m)
  end while
  return ($P_{\text{n-d}} = L_E^f$)
end function

function Initialization($G, S, V_m$of all links)
  $L^f = \{\,\}$
  for each $m \in M$ that starts from S **do**
    $L = L \cup V_m$
  end for
end function

function ChooseOptimumLabel(L, L^f)
  find the path $p_i \in L$ with minimum delay
  i = ending node of path p_i
  $L^f = L^f \cup V_{p_i}$
  return(i)
end function

function ObtainNewLabelsAndDiscardDominatedPaths(G, i, L, V_m)
  1:
  for each $j \in N$ neighbour of i connected through link l **do**
    $V'_{p_j} = V_{p_i} \odot V_l$
    for all $V_{p_j} \in L$ **do**
      if $V_{p_j} > V'_{p_j}$ **then**
        goto 1 (check the next neighbour)
      else
        if $V'_{p_j} > V_{p_j}$ **then**
          $L = L - V_{p_j}$
        end if
      end if
    end for
    $L = L \cup V_{p_j}$
  end for
end function

Running Time

The outer while loop in the first function computes Y final non-dominated paths to n nodes. During the process X paths are stored for each node, from which Y paths are selected. The size of the utilisation profile is u and Δ is the maximum degree of the network. The size of the input for the graph is $m + n$, whereas the size of the input for utilisation profiles is $m \cdot u$. We calculate the running time of the algorithm as follows: $\mathcal{O}(n \cdot Y(\log(n \cdot X) + \Delta \cdot u \cdot X))$, where $n \cdot Y$ is the number of iterations of the while loop, $\log(n \cdot X)$ is the running time for choosing the optimum label from a priority queue of $n \cdot X$ elements and $\Delta \cdot u \cdot X$ is the running time for finding and discarding the new label. In the worst case scenario there can be u paths stored at each node for the path weight from 0 to u, thus X can be replaced by u. The same can be the case for Y final paths. For the worst case Δ can be replaced by m links. Thus, the running time can be bounded as follows: $\mathcal{O}(n \cdot u(\log(n \cdot u) + m \cdot u^2))$.

4.2 Dynamic Critical Path for Grids (DCPG)

DCPG [58] is a list scheduling heuristic that was proposed for non-AR environments and is an extension of the Dynamic Critical Path method [50]. In each step, an available job is selected and assigned to a resource. Priority for selection is given to the jobs on the *critical path*, which is the longest path from the entry node to the exit node of a workflow, calculated on the basis of the (estimated) costs of nodes (computation time) and edges (data transfer time). As the name suggests, in DCPG the critical path keeps changing during the scheduling process. For each job, estimates *AEST* and *ALST* are calculated, whose difference determines the level of criticality of a job. The jobs with the minimum difference of *ALST* and *AEST* are the most critical ones. The jobs on the critical path have a difference of 0. The calculation is done as follows [58].

$$\overline{estET(j)} = \frac{JS_j}{\underset{r \in R}{max}\ (PS_r)},$$

$$ADTT_j = \frac{DS_{f_j}}{\max\limits_{r \in R} (BandWidth(r))},$$

where f_j is the output file of job j and *BandWidth(r)* is the maximum capacity of any link incident with $r \in R$. *AEST* of job j is recursively defined as,

$$AEST(j) = \max_{1 \le i \le x} (AEST(pj_i) + \overline{estET(pj_i)} + Comm_{j,pj_i}(r_j, r_{pj_i})), r_j, r_{pj_i} \in R,$$

where j has x parent jobs and pj_i is its ith parent job. r_j and r_{pj_i} are the resources on which job j and pj_i are considered, respectively. After a job is scheduled, $\overline{estET}$ and *ADTT* are updated accordingly. Thus *AEST* changes for the scheduled jobs. $AEST(j) = 0$ if j is the entry job.

$$Comm_{j,pj_i}(r_j, r_{pj_i}) = 0, \text{if } r_j = r_{pj_i},$$

$$Comm_{j,pj_i}(r_j, r_{pj_i}) = ADTT_{pj_i}, \text{ if } j \text{ and } pj_i \text{ are not scheduled.}$$

$Comm_{j,pj_i}(r_j, r_{pj_i})$ is assigned the actual communication time of the path which gives the earliest arrival time between pj_i and j, if these jobs have already been scheduled on different resources. Using this definition, *AEST* of all jobs is calculated by traversing the DAG in breadth-first manner starting from the entry job. On the basis of *AESTs* of all jobs, the dynamic critical path length *(DCPL)* is defined as

$$DCPL = \max_{1 \le i \le v} (AEST(j_i) + \overline{estET(j_i)}),$$

where v is the total number of jobs. After *DCPL* is calculated, *ALST* is computed for all jobs in breadth-first manner in reverse direction. Thus, *ALST* for job j is defined as

$$ALST(j) = \min_{1 \le i \le y} (ALST(cj_i) - \overline{estET(j)} - Comm_{cj_i,j}(r_{cj_i}, r_j)),$$

where y is the total number of child jobs of job j and cj_i is its ith child job,

$ALST(j) = DCPL - \overline{estET(j)}$, if j is the exit job.

With the aim of reducing the overall length of the workflow schedule, a job on the critical path is selected for scheduling if all its parent jobs are scheduled. If there is no critical job ready then the job with minimum difference of *ALST* and *AEST* is chosen. The selected job is mapped to the resource which gives the minimum completion time for the job as well as the earliest possible start time for its most critical child job, that is the earliest combined start time of the job and its critical child. Here, the critical child is the child with minimum difference of *ALST* and *AEST*.

4.2.1 DCPG Example

We explain the DCPG algorithm with the help of the example given in Figure 4.2. This example also illustrates how the critical path can change during the scheduling process. The sample workflow consists of 6 jobs J_1, J_2, J_3, J_4, J_5 and J_6. The length of jobs and the size of output of each job is measured in MI and Giga Bytes (GB) (see Figure 4.2a). The jobs are to be mapped on two resources R_1 and R_2, each with a single processing unit. Their processing speed and bandwidth capacity is shown in Figure 4.2i. First, the *estET* and *ADTT* are calculated, shown in Figure 4.2a. The next step is to calculate the *AEST* and *ALST* for all jobs and the *DCPL* of the workflow using the *estET* and *ADTT* values (see Figure 4.2b), as mentioned in the Section 4.2. The jobs which are on the current critical path are J_1, J_2, J_4 and J_6. Since the job J_1 is the first available critical job, it is allocated to the resource R_1, which gives it the minimum combined start time.

The DCPL remains unchanged with this mapping, i.e., 760 (Figure 4.2c). The next available critical job J_2 is also mapped to the resource R_1 because of the minimum start time, which is 100. After this mapping the *DCPL* changes to 730 and the critical path changes as well. The job J_4 is excluded from the critical path while J_3 and J_5 are included into it, thus J_3 becomes the next available critical job (Figure 4.2d). Job J_3 is mapped to the resource R_2, which gives the earlier starting time of 180 as compared to 300 on the resource R_1. After mapping J_3, the *DCPL* changes to 820 and J_5 becomes

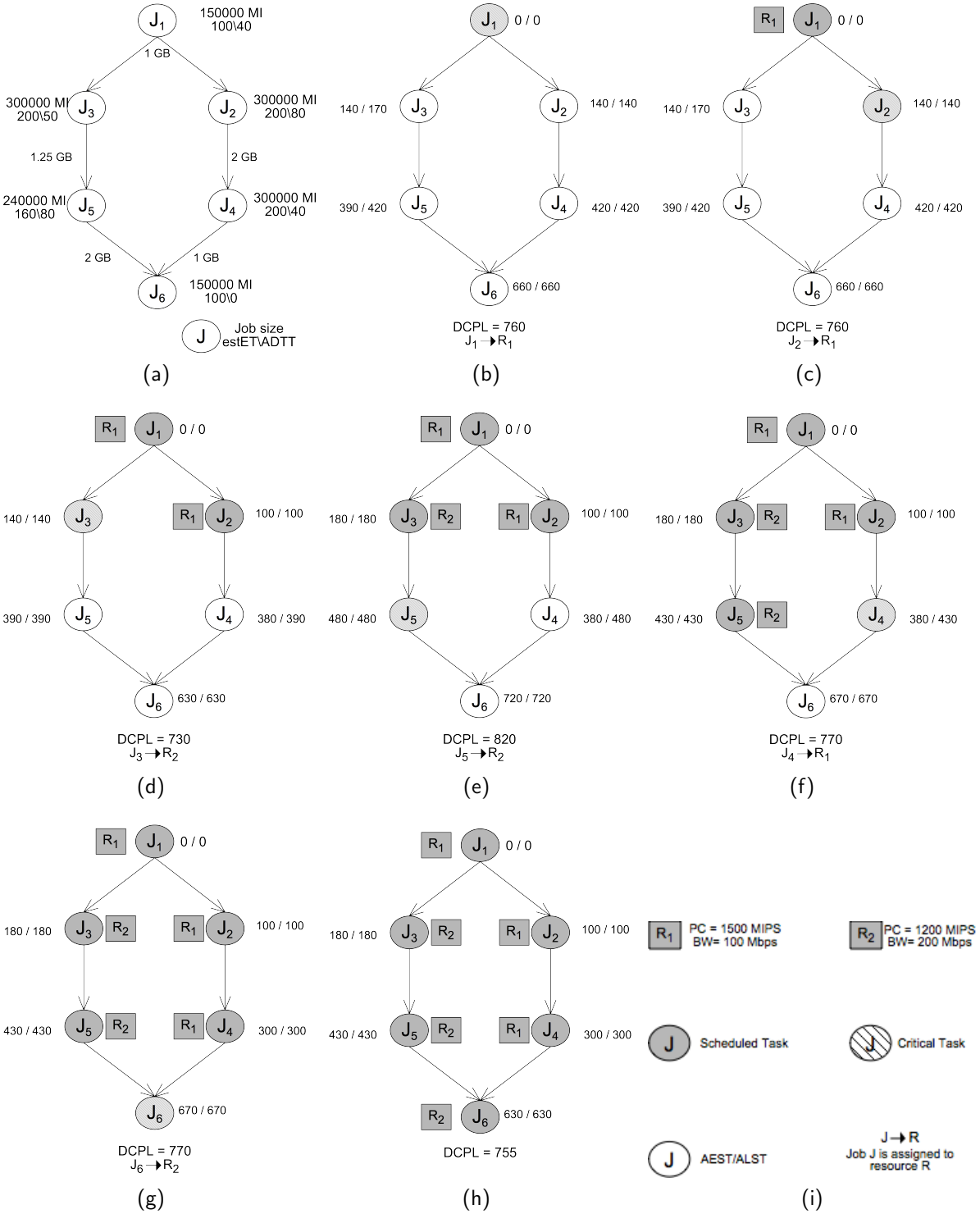


Figure 4.2. DCPG example.

Jobs	Resources	Start	End
J_1	R_1	0	100
J_2	R_1	100	300
J_3	R_2	180	430
J_4	R_1	300	500
J_5	R_2	430	630
J_6	R_2	630	755

Table 4.1. Final Schedule for the example in Figure 4.2 .

the available critical job (Figure 4.2e). J_5 is mapped to the same resource on which its parent job is scheduled, i.e., R_2, to get the minimum combined start time of J_5 and its critical child job J_6. After this step the *DCPL* changes to 770 and J_6 becomes the only unscheduled critical job. To meet the dependency requirement for the job J_6, its parent job J_4 is scheduled on the resource R_1, which gives the minimum starting time of 300 (Figure 4.2g) and the minimum combined start time as well. Finally, the last job J_6 is scheduled on R_2, with 630 as the minimum starting time, and the final *DCPL* becomes 755 (Figure 4.2h). The final schedule of the workflow is given in the Table 4.1.

4.3 GridSim: A Simulator for the Grid Environment

As different fields of science are getting more attracted to utilise the distributed computing technologies, more work needs to be done to tweak them. The community which develops and maintains distributed computing technologies needs to pace up as well for their solutions to be validated. For this purpose, one approach is to test the solutions on a dedicated deployed distributed system or a system which is under utilisation for other purposes. The former approach is very expensive for many, and the latter one makes it hard for researchers to configure the system according to a variety of experimental requirements. To deal with such problems, simulators are developed.

The GridSim [67] simulator is among the well known simulators for P2P, Cluster, Grid and now Cloud computing environments. Other simulators worth mentioning include Bricks [12], SimGrid [25] and MicroGrids [65]. GridSim is an effort to provide a number of features, especially those related to scheduling activities, that are lacking in the above-mentioned simulators, and to make them all available in one place. The features provided

by GridSim, as presented in [67], are as follows;

- It allows modelling of heterogeneous types of resources, like PC and cluster.
- Resources can be modelled as operating in space- or time-shared mode.
- Resource capabilities can be defined (in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark).
- Resources can be located in any time zone.
- Weekends and holidays can be mapped depending on the local time of a resource to model non-Grid (local) workload.
- Resources can be booked for advance reservation.
- Applications with different parallel application models can be simulated.
- Application tasks can be heterogeneous. They can be CPU or I/O intensive.
- There is no limit on the number of application jobs that can be submitted to a resource.
- Multiple user entities can submit tasks for execution simultaneously in the same resource, which may be time-shared or space-shared. In time-shared resources, jobs can be pre-empted before completion, while in the case of space-shared resources, jobs are allowed to complete their execution. This feature helps in building schedulers that can use different market-driven economic models for selecting services competitively.
- The network speed between resources can be specified.
- It supports simulation of both static and dynamic schedulers.
- Statistics of all or selected operations can be recorded and analysed using GridSim statistics analysis methods.

GridSim is Java based and extends SimJava [31], which is a discrete event based simulator. We have chosen GridSim for two main reasons: First, it is Java based; second,

a built-in AR feature for resources was available. The following section gives a brief description of SimJava followed by the detailed working of GridSim. Appendix A gives the details regarding the modifications and additions we made in GridSim for our work.

4.3.1 GridSim Entities and their Interactions

SimJava is a Java-based discrete event simulation application. Entities are created in SimJava which can run in parallel in their own threads. The behaviour of an entity is available in the body() method. The body method is also used to communicate with other entities. First, all entities are created and their body() methods are put in the run state. To handle simulation events, SimJava uses a Sim_system object to create future events sequentially. Sim_system maintains a timestamped ordered queue for this purpose. After an entity causes an event to occur, the Sim_system object pauses the thread of the event-triggering entity and places the event in the future queue. All the entities are halted, then events are popped out of the event queue and the simulation time is forwarded according to the occurring event and the relevant entities are restarted. This process is repeated until all the events in the event queue are generated. GridSim uses this event generation mechanism of SimJava. GridSim creates entities extended from the Sim_entity class. The entities include *user, broker, resource, grid information service and router*. Furthermore, each entity has its own threaded input and output entity to communicate with other entities and to simulate a communication delay, if any. Brief descriptions of the functionalities of the entities are as follows.

- **User** entities can create jobs, schedule these jobs, and set different optimisation parameters like budget and time. Users can be created in different time zones.
- Each user is connected to a **Broker** entity. The broker is the entity which implements the scheduling algorithm. The user submits a job to its broker which schedules it, according to the user's scheduling policy, on a resource. Before this, the broker entity receives a list of available resources from the Grid Information Services (GIS) entity. Because of the broker entity, users can schedule jobs ac-

cording to individual needs. Overall this creates a real Grid environment where different types of requests with different optimisation functions are handled.

- The **Resource** entity, a computing entity, can have different configuration parameters like number of machines with different number of processors with different speeds. The time zone, load on different days of the week and dates of the year can be set for each resource. The internal process scheduling policy, i.e., time shared or space shared, can also be set.
- The **Grid information service (GIS)** provides resource registration services. It also responds to queries regarding resource configurations and status.
- The **Router** entity connects resources and users. At the start of the simulation, routers advertise their availability and the routing tables are populated. For routing of data, the RIP routing protocol is available by default.
- **Input and Output** entities are connected to each entity as an independent entity, having their own body method to handle events while running in their own threads. All the networked entities communicate with each other through their input and output entities. Having separate input and output channels helps simulating duplex and multi-user parallel communications.
- A **Gridlet** is an entity which contains information about a job. The information is about the owner (a user) of the job, job length in MI, input and output file sizes. A gridlet is created and submitted by a user to a grid resource.

GridSim entities interact with each other using events. An entity which needs a service generates a service request event and sends it to an entity which can deliver the service. Upon receiving a service request event the entity generates an event (or several events) in response to it and sends it to one or more entities, depending on the type of request. The provider entity may wrap the data, in response to the service request, inside the event. The requesting entity receives the response from the provider and extracts the information stored in the wrapped data inside the event object. The sender and receiver of the events have to agree upon a protocol, like what data is stored in the event and

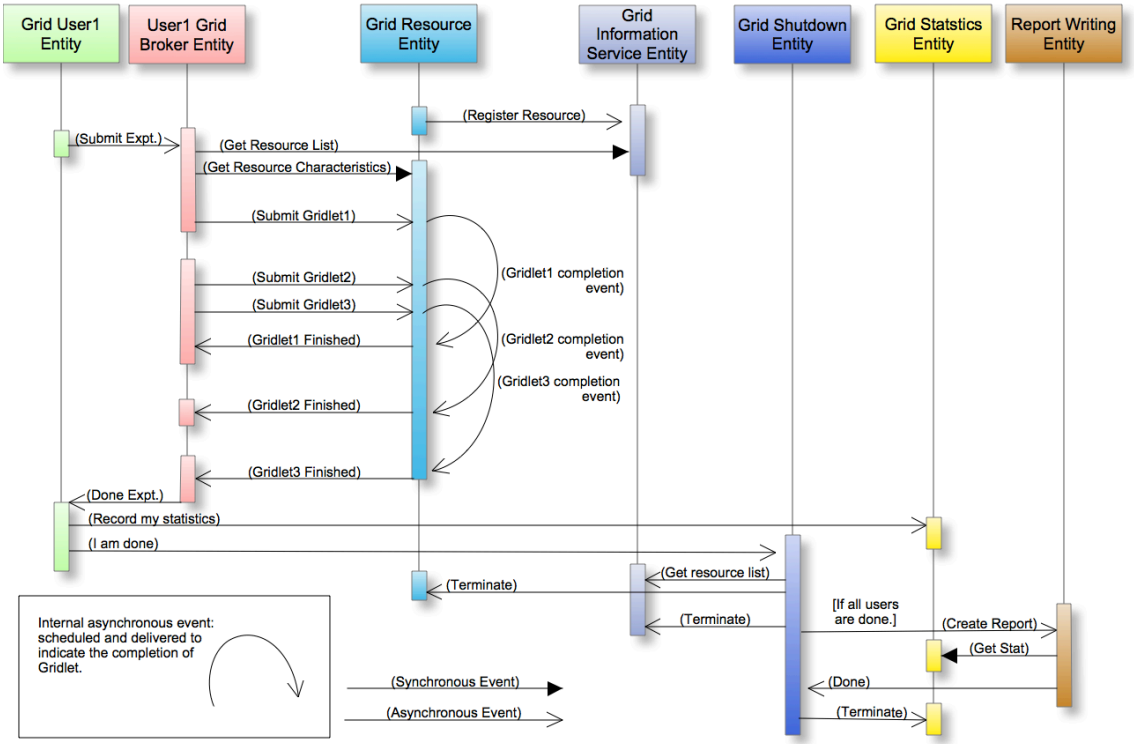


Figure 4.3. An event diagram for the interaction between a space-shared resource and other entities [67].

how to extract it. All the event processing is done in the body() method.

Events can be internal or external. Internal events are generated by an entity for itself while external events are sent by other entities. An entity can differentiate between the events by checking its originator. There is another classification of events on the basis of the type of the service the events are related to. The classes are *synchronous* and *asynchronous* events. An event is synchronous if the originator of it waits until the response from the target entity of the event is received. In case of asynchronous events, the originator of the event does not wait for the response from the targeted entity and continues with other activities.

We further explain the interaction between the different entities from the start to the end of a simulation using the diagram in Figure 4.3. At the start of the simulation, the entities are created, and the resource entities are registered with GIS. The user entity submits the experiment (a set of three jobs) to the broker entity. Before submitting the jobs, the broker entity requests the list of available resources from GIS and waits for the response (synchronous event). After the retrieval of the resource list, the broker requests the resource configurations like processor speed and idle processors (synchronous event).

After the required information is retrieved, the jobs are submitted according to the user's scheduling policy. A resource entity receives the job submission event and starts executing it. To execute a job, the resource entity generates an internal asynchronous event for job completion. After receiving the job completion event, it updates the remaining length of the job. A job is returned to the user when it is finished. When all the jobs are returned to the user, it sends a user shutdown request event to the Grid Shutdown entity. In case of multiple users, if all the users are finished the Grid Shutdown entity shuts down all the entities in response to this event. Statistics and reports can be requested before finishing the simulation.

Chapter 5

Partner Based Dynamic Critical Path for Grids (PDCPG)

PDCPG is an extension of DCPG [58], which was proposed for non-advance reservation environments while considering the heterogeneity of the Grid. PDCPG considers partner jobs while scheduling a workflow. Recall that jobs in a workflow are said to be partner jobs if they have at least one common child job. This approach is adopted with the aim of minimising the communication costs between parent and child jobs, thus minimising the workflow makespan. For advance reservation, PDCPG adopts the technique proposed in [26], i.e., it uses a joint advance reservation technique for communication and computation task scheduling for Grid resources connected by an *Optical Burst Switching* (OBS) [77] network. It is shown that for better performance of computation and communication intensive tasks in the Grid, the jointly optimised utilisation of communication and computation resources can be used. For PDCPG we assume that the Grid resources are connected using the OBS network architecture. We study the behaviour of PDCPG in AR environments in scenarios with and without resource failures. The environment in which failures do not occur is referred to as *static environment* and the environment in which failures can occur is referred to as *dynamic environment*. The performance of PDCPG is compared with an AR version of DCPG, whose original version has been shown to outperform other well known Grid workflow scheduling heuristics in [58]. Our

results show that PDCPG performs well especially when the granularity of the workflow is low.

The rest of this chapter is organised as follows. Section 5.1 presents the details of PDCPG. In Section 5.1.1 the scheduling problem under consideration is made precise. The routing technique used for PDCPG is described in Section 5.1.2, and our novel resource selection technique is discussed in Section 5.1.3. The experimental setting used for the evaluation of PDCPG in static AR environments is described in Section 5.1.4, and the results of the performance evaluation are discussed in Section 5.1.5. The scheduling problem for dynamic AR environments and a modification of PDCPG for such environments are presented in Section 5.2. The setting used for performance evaluation as well as the experimental results are discussed in Section 5.2.1. Conclusions for this chapter are given in Section 5.3.

5.1 PDCPG Heuristic

In this section we propose and evaluate a new heuristic for workflow scheduling in static AR environments. The proposed PDCPG heuristic consists of two main parts: the selection of a job from the workflow to be scheduled next, and the mapping of the selected job to a resource (which includes the routing of any input data files to that resource). For job selection, a modified DCPG technique is used. For resource selection, a novel partner-based technique is applied where the algorithm attempts to schedule partner jobs on the same resource on the basis of the condition given in (5.1) and briefly described as follows. If the sum of the expected completion time of the job to be scheduled on the resource allocated to the partner job and the transmission time of its output is less than or equal to the sum of the completion time of the partner job and the transmission time of its output, then the job is allocated to that resource and the further exploration of other resources is skipped, which saves time during the schedule computation. In the following section, the problem at hand is specified. In Section 5.1.2, the routing technique used for PDCPG is described, and the resource selection technique

is explained in detail in Section 5.1.3.

5.1.1 Problem Description

For the problem under study, the workflow model is as given in Section 2.2 and the model of the Grid environment is as given in Section 2.1. In addition, it is assumed that in the given Grid environment communication and computation resources support AR and the network architecture is OBS. Further, it is assumed that resources do not fail and that the user, who is the owner of the workflow, requests one CPU for each job and requests a certain bandwidth for each file transfer. For AR, the framework proposed by Varvarigos et al. [70] (see Section 4.1) is used. The objective is to schedule the workflow submitted by the user so that its makespan is minimised.

5.1.2 Proposed Routing Method

As a subproblem of the resource selection step, the problem of transferring the input data file(s) of a job to a resource (cluster) needs to be considered. We refer to this subproblem as the routing problem. A variant of the technique to compute path availability in an AR environment proposed in [70] and discussed in Section 4.1 is adopted for the proposed work. The variation is in the way the set of path-cluster pairs (p-c pairs) is maintained. Our set of p-c pairs contains only one pair for each cluster and is computed by using a variant of Dijkstra's algorithm for shortest paths (see Algorithm 5.1). A path is paired with the cluster that gives the minimum *EAT* for the considered connection. The shortest path is computed on the basis of the following measure for the length of a path:

$$EAT(p_r, b) = d_{p_r} + ETT(p_r, b) + b,$$

where p_r is the path to resource $r \in R$, d_{p_r} is the delay of path p_r, b is the duration of the data transfer when the requested bandwidth is available, and $ETT(p_r, b)$ is the earliest possible time at which a transfer of duration b on path p_r can be started. The

drawback of this approach is that the computation of an optimal path (with earliest arrival time) is not guaranteed unless all links have the requested bandwidth available at all times. The pseudo-code of our variant of Dijkstra's Algorithm for AR environments is shown in Algorithm 5.1. The relevant notation is as follows.

Notation

A graph G has a set of V vertices and a set of E edges. *rbw* is the requested bandwidth, b is the duration of the transfer of data when *rbw* bandwidth is utilised. Each $e \in E$ has a *CAV* vector, which gives the availability of the edge for *rbw*. The weight of each edge $e \in E$ is the earliest arrival time *EAT*(e, b) which is obtained by finding the first position in *CAV*$_e$ at which data of duration b can arrive from the start vertex u to the end vertex v of edge e. The weight of a path P is *EAT*(P, b), which is obtained by performing the associative operation between all edges $e \in P$ as mentioned in Section 4.1. The algorithm computes the shortest paths in terms of earliest arrival time from the *source* vertex to all vertices $v \in V$.

EAT$[v]$ stores the minimum earliest arrival time to vertex v, *pathCAV*$[v]$ stores the *CAV* of the shortest path and *previous*$[v]$ stores the node previous to v on the shortest path. *LinkCAV*$[u][v]$ stores the *CAV* of the edge connecting vertices u and v. For pseudo code see Algorithm 5.1.

Execution Example

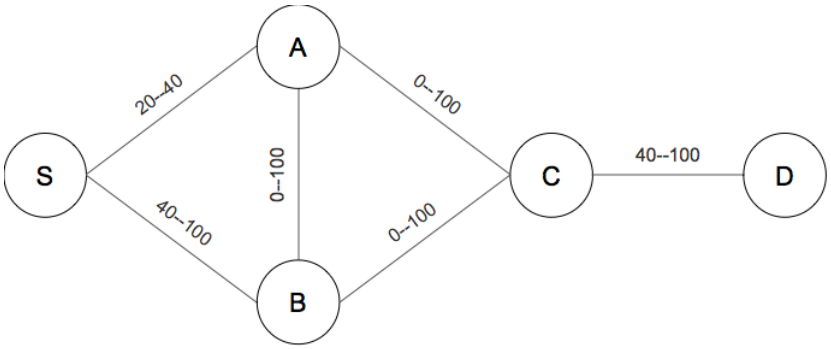
We give an example of the execution of the modified Dijkstra's algorithm till the first relaxation step in Figure 5.1. We have a network consisting of *S*, *A*, *B*, *C* and *D* nodes (Figure 5.1a). For the sake of simplicity, delays of the edges are assumed to be zero and the status of the utilisation profiles of the edges is not shown, but the availability of the edges for the requested bandwidth is displayed, instead. The availability is computed after going through their utilisation profiles. The duration of transfer is assumed to be 20. The source node is *S* and we want to find the path to all nodes with earliest arrival

Algorithm 5.1 Variant of Dijkstra's Algorithm for Path Finding in AR Environment

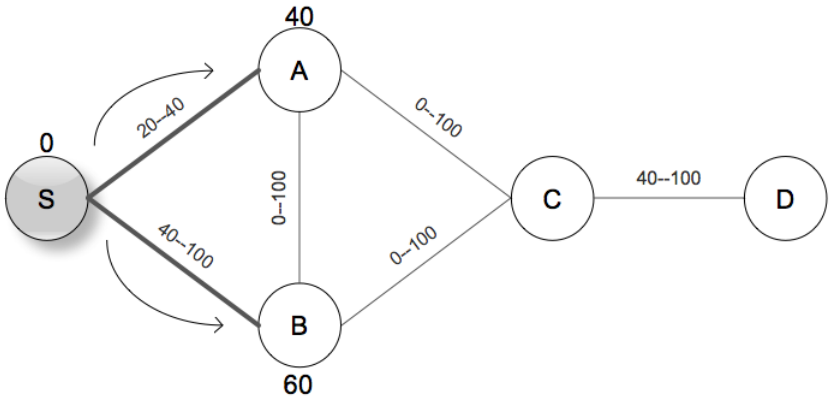
```
function DijkstraVariant(G, source, b)
    for each vertex v in V do
        EAT[v] ← infinity
        previous[v] ← undefined
    end for
    EAT[source] ← 0
    Q ← the set of all nodes in V
    while Q is not empty do
        u ← vertex in Q with with smallest earliest arrival time in EAT[ ]
        remove u from Q
        if EAT[u] =infinity then
            break
        end if
        for each neighbour v of u do
            newPathCAV[v] ← ComputeNewPathCAV(pathCAV[u],LinkCAV[u][v])
            temp ← ComputeEAT(newPathCAV[v], b)
            if temp < EAT[v] then
                EAT[v] ← temp
                previous[v] ← u
                pathCAV[v] ← newPathCAV[v]
            end if
        end for
    end while
end function

function ComputeNewPathCAV(pathCAV[u],LinkCAV[u][v])
    Computes and returns new CAV as explained in Section 4.1.
end function
function ComputeEAT(pathCAV[v], b)
    Finds and returns the first position in pathCAV[v] at which data of duration b
can arrive at vertex v
end function
```

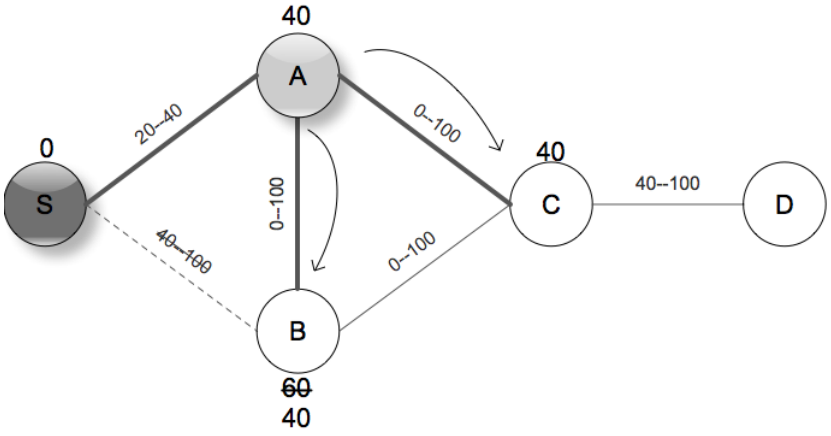
time. Starting from the node *S*, the arrival time of the path *S*-*A* to its neighbour node *A* is 40 and to the node *B* is 60 by following the path *S*-*B* (Figure 5.1b). In the next step, the neighbours of the node *A* are explored. The new path *S*-*A*-*B* to the node *B* gives the arrival time 40, which is better than the previous path *S*-*B*. Thus, the previous path is replaced by the new one and the earliest arrival time is updated as well.



(a)



(b)



(c)

Figure 5.1. Execution till a better path to the node B is found by the variant of Dijkstra's algorithm.

5.1.3 Resource Selection

Before discussing the resource selection method in detail, we explain the modifications of some concepts from [58] (see Section 4.2). The estimated execution time $\overline{estET}$ of job j is calculated as

$$\overline{estET(j)} = \frac{\sum_{1 \leq i \leq m} \frac{JS_j}{PS_{r_i}}}{m},$$

i.e., as the average execution time of the job on all m resources, whereas in [58] it is calculated as the execution time of the job on the fastest resource. The calculation of *ADTT* is modified as follows because of the AR environment:

$$ADTT_j = \frac{DS_{f_j}}{rbw_{f_k}},$$

where rbw_{f_k} is the requested bandwidth for the output file f_k of job j. As the considered network is *OBS*, the propagation delay is considered negligible and ignored for *ADTT*.

The above-mentioned modifications are used in calculating *ALST* and *AEST* of jobs. The difference between these values determines how critical a job is and whether it is given priority for selection. The jobs with the same *ALST* and *AEST* are considered the most critical jobs.

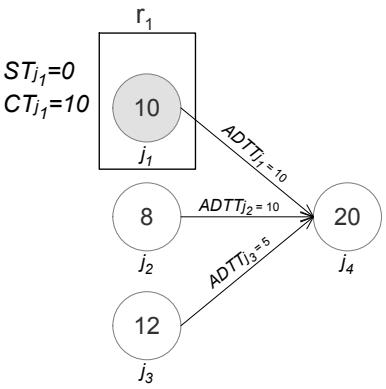
A job to be scheduled j' is selected from among the available jobs of the workflow using the same criteria used by DCPG. For resource mapping, it is checked whether one or more of the partner jobs of j' have already been scheduled or not. If no partner job has been scheduled, then the resource that gives the earliest completion time is chosen. If one or more of the partner jobs of j' have already been scheduled, then the availability of the resources on which those partner jobs are scheduled is checked first. The job j' is mapped to the resource of its ith partner $ptr_i \in PTR_{j'}$ if the condition

$$expCT_{j'}^{r_{ptr_i}} \leq CT_{ptr_i} + ADTT_{ptr_i} - ADTT_{j'} \tag{5.1}$$

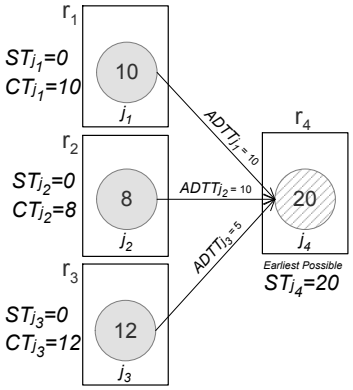
is satisfied, where

$$expCT_{j'}^{r_{ptr_i}} = \max_{1 \leq k \leq h} (EAT^{f_k}(p_{r_{ptr_i}}^{f_k}, b)) + estET_{j'}^{r_{ptr_i}},$$

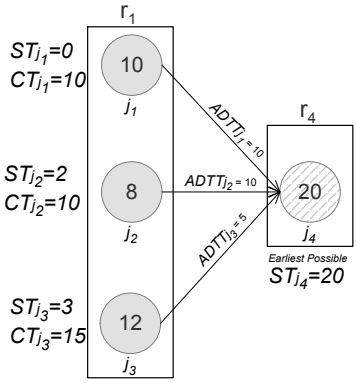
where $(EAT^{f_k}(p_{r_{ptr_i}}, b))$ is the earliest arrival time of input file f_k to the resource r_{ptr_i} when path $p_{r_{ptr_i}}^{f_k}$ is used. This condition is checked against all scheduled partner jobs one by one in descending order of criticality. The first resource that satisfies the condition is selected for scheduling j', and thus any further availability checking is skipped. If the condition is not met for any of the partner jobs, then the resource which gives the earliest completion time among all resources is selected. The motivation for this technique is that the job that is scheduled first is a more critical job, and further improvement can only be possible by reducing the communication duration to the critical child job. Even though the job may finish later on the partner's resource as compared to other resources, this allocation increases the probability of reducing the communication duration between all parent jobs and a critical child. Another point that can be noted is that the condition (5.1) allows only an affordable delay in the completion time of j', as illustrated by the following example. First, the case in which all jobs are scheduled according to their earliest possible completion time is discussed, and then it will be shown that in spite of delayed scheduling of jobs the earliest possible start time of the critical child job will not be affected. The example is as follows. j_1, j_2, j_3 are three partner jobs having execution duration $10, 8, 12$ seconds (seconds will be abbreviated as sec in the following), respectively (see Figure 5.2a). These are critical in the same order as shown in the figure from top to bottom, and j_4 is their common child. The resources on which these jobs are to be scheduled have three CPUs each. Only one CPU on each resource is available at time $0sec$. The most critical job j_1 is scheduled on resource r_1 having start time $ST_{j_1} = 0$ and completion time $CT_{j_1} = 10sec$. In Figure 5.2b it is shown that j_2, j_3 are mapped to resources r_2 and r_3, respectively, with start times $ST_{j_2} = 0$, $ST_{j_3} = 0$ and completion times $CT_{j_2} = 8sec$, $CT_{j_3} = 12sec$. Assuming that only r_4 is available for the job j_4, the earliest possible start time for it would be at $20sec$, which is the time at which output data of all the parents will be available on r_4. Now consider another scenario as shown in Figure 5.2c. At the time of scheduling j_2, j_1 is the only scheduled



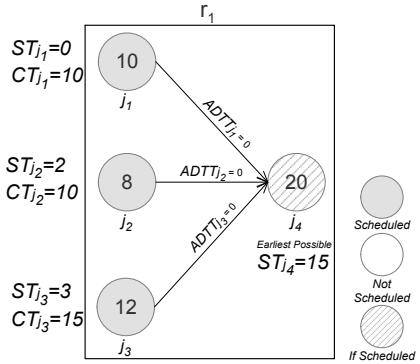
(a) Most critical job j_1 is scheduled on resource r_1.



(b) Earliest possible start time of j_4 on only available resource r_4 is 20, when all jobs are scheduled on different resources with earliest possible completion time.



(c) Using (5.1), j_2 and j_3 are scheduled on the same resource as j_1 with later completion time as compared to Figure 5.2b, but still the earliest possible start time for j_4 on only available resource r_4 is 20.



(d) The mapping technique used in Figure 5.2c helps to increase the chance for parent and child jobs to be scheduled on the same resource.

Figure 5.2. Partner-based scheduling example

partner job. By (5.1), the affordable delayed completion time for j_2 on r_1 is

$$expCT_{j_2}^{r_1} \leq CT_{j_1} + ADTT_{j_1} - ADTT_{j_2} = 10 + 10 - 10 = 10.$$

Assume that before time $2sec$, there are not enough CPUs available on r_1 to run j_1 and j_2 in parallel, and the same assumption is made for j_3 on r_1 before time $3sec$. j_2 is scheduled on r_1 with $ST_{j_2} = 2sec$ and $CT_{j_2} = 10$, which is $2sec$ later than in the scenario shown in Figure 5.2b. At the time of scheduling j_3, there are two scheduled partner jobs j_1 and j_2. First the condition (5.1) is checked for j_1, which is

$$expCT_{j_3}^{r_{j_1}} \leq CT_{j_1} + ADTT_{j_1} - ADTT_{j_3} = 10 + 10 - 5 = 15.$$

If it is not satisfied, then the next partner job is considered,

$$expCT_{j_3}^{r_{j_2}} \leq CT_{j_2} + ADTT_{j_2} - ADTT_{j_3} = 10 + 10 - 5 = 15,$$

There are two reasons to check the condition 5.1 against all partner jobs instead of only against the most critical job. It is possible that a less critical job may finish later than the most critical job, or partner jobs might be scheduled on different resources. After jobs j_1, j_2 and j_3 are scheduled on r_1, if j_4 is mapped to the only available resource r_4, still the earliest possible start time for j_4 would be at $20sec$. Thus, the delayed scheduling of parent jobs would not affect the earliest possible starting time of the child job. Moreover, this mapping technique increases the chance of reducing the communication duration between parent and child jobs if r_1 is available for j_4 (see Figure 5.2d). Thus it can improve the workflow makespan, and it can reduce the running time of the algorithm as well.

5.1.4 Performance Evaluation

We evaluate the performance of PDCPG by comparing it to an AR version of DCPG in a simulation environment for different workflows, with makespan as the optimisation

criterion. Since it was shown in [58] that DCPG outperforms HEFT, Min-Min, Max-Min, Myopic and several meta-heuristic scheduling algorithms in terms of workflow makespan in non-AR environments, and because we are not aware of any well known scheduling heuristic for AR environments, we chose to compare our work with an AR version of DCPG. The only difference between the original version and the AR version of DCPG is the way in which *ADTT* is calculated. In the AR version, the calculation reflects the influence of the AR environment and is done in the same way as for PDCPG (see Section 5.1.3). In this section, first the environment for simulation experiments is described, and then the results are presented.

Experiment Environment: GridSim [67] is used to simulate a Grid environment that supports AR for both communication and computation resources. A single user submits a workflow, and each job of the workflow requests a single processor. The requested bandwidth for each file transfer is also specified.

The platform details of the simulation system on which GridSim was executed are as follows: 2.4 GHz Core 2 Duo processor, Mac OSX operating system and 4 GB RAM.

Resource Model: The European Data Grid (EDG) testbed (see Section 2.6) is used as the model of Grid resources with interconnecting OBS network in the simulation experiments. The experiments are conducted for a high load of 90% or above on the resources, and for workflows with high and low granularity. Load refers to the percentage of CPUs of a resource under utilisation at any time. The CPU load in each interval of 5–10 consecutive time slots is generated by generating a uniform random value between 0 and the total number of CPUs, and taking the maximum of that value and the target CPU load (e.g. 90% of the total number of CPUs). The granularity range for low granularity is 0.1 to 0.5, and for high granularity it is 1.3 to 1.8. The load on the network links is set to zero so that enough capacity is always available for the requested bandwidth, in order to check the performance of the proposed algorithm when the bottleneck lies in the computation resources. Utilisation profile vectors of resources and links are used to

store their instantaneous load. For each simulation run, randomly generated values are assigned to the utilisation profiles.

Workflows: The details of the workflows that have been selected for experimentation are given is this section. The e-Protein workflow of 15 jobs is taken from a real world application [57], three workflows of 30, 60 and 90 jobs are taken from j301_1, j601_1, j901_1 in the Project Scheduling Problem Library (PSPLIB) [47], respectively, and four random workflows comprising 50, 100, 150, and 200 jobs are generated using the technique described in [58]. The simulation is run approximately 100 times for each of the workflows. For workflows other than random workflows, in each run, jobs are assigned lengths ranging from 40,000 MI to 110,000 MI randomly according to a uniform distribution, and their output file size is also generated randomly according to a uniform distribution on the basis of the granularity under consideration. For low granularity, the output file size ranges from 220 to 250 Megabytes, and for high granularity it ranges from 40 to 60 Megabytes. For random workflows, the job length ranges from 10,000 MI to 30,000 MI, and the output file size ranges from 100 to 280 Megabytes. This is done to minimise the size of the required utilisation vector for larger workflows, because for larger workflows the larger utilisation vector computation could exceed the available main memory of our simulation system. It should be noted that the size of the utilisation vector determines the limit of the time within which reservations can be made.

5.1.5 Results

The results for low granularity are presented first, then the results for high granularity will be discussed.

The main aim of PDCPG is to minimise the communication duration between the parent and child jobs, and scheduling these jobs on the same resource can help to achieve this. This is the reason why the improvement obtained by PDCPG is more noticeable for low granularity than for high granularity. In Table 5.1, the average makespan of 100

	PDCPG	DCPG	Improvement %age
eProtein	656.89	666.6	1.46
J301_1PSPLIB	1018.9	1034.9	1.55
J601_1PSPLIB	1149	1174	2.13
J901_1PSPLIB	1370	1401	2.21
50Random	441.9	451.4	2.08
100Random	722.4	745.8	3.14
150Random	646	691.8	6.62
200Random	847.4	902.4	6.09

Table 5.1. Average makespans generated by heuristics, and improvement percentage of PDCPG.

	PDCPG	DCPG
eProtein	1.4	2.5
J301_1PSPLIB	2.9	3.9
J601_1PSPLIB	5.3	7
J901_1PSPLIB	8	10
50Random	4.5	5.9
100Random	8.1	12
150Random	12.2	19.3
200Random	16.6	29

Table 5.2. CPU time in second(s) for scheduling algorithms for each workflow for low granularity.

simulation runs for each workflow by both heuristics is shown, as well as the percentage improvement achieved by PDCPG. With an increase in the number of jobs in a workflow, the percentage of improvement for PDCPG also increases.

Although the improvement obtained by our heuristic is not by a very large margin for all workflows, PDCPG is able to achieve relatively significant improvements by up to 6.6% for the larger random workflows. It should also be noted that DCPG is known to outperform many other popular heuristic scheduling algorithms, and PDCPG achieves the improved results with significantly lower running time for the schedule computation (see Table 5.2). Table 5.2 shows that PDCPG is approximately two times faster than DCPG for larger random workflows on our simulation platform. The running time difference tends to increase with the increase in the size of the utilisation profile and computing resources in the network. It should be noted, however, that we have not optimised or tuned the code of the two heuristics in our implementation, so the running times reported in Table 5.2 should only be taken as an indication of the computational efficiency of the heuristics. Nevertheless, both algorithms have been implemented in similar style using similar data structures, so it is unlikely that the faster running times observed for PDCPG are only due to coding issues.

PDCPG does not always outperform DCPG, but the number of times DCPG improves on

	PDCPG	DCPG	Imp\Worsening %age
eProtein	548.2	549	0.15
J301_1PSPLIB	733.8	734	0.03
J601_1PSPLIB	850	850.4	0.05
J901_1PSPLIB	937.8	943.1	0.57
50Random	366.1	365.4	-0.2
100Random	686	679	-1.03
150Random	717.5	718.5	0.14
200Random	801.9	803.1	0.15

Table 5.3. Average makespans generated by heuristics for high granularity.

	PDCPG	DCPG
eProtein	1.83	2.41
J301_1PSPLIB	3.04	3.87
J601_1PSPLIB	5.49	6.37
J901_1PSPLIB	7.4	8.74
50Random	4.9	5.28
100Random	9.17	11.21
150Random	13.66	18.19
200Random	23	29

Table 5.4. CPU time in second(s) for scheduling algorithms for each workflow for high granularity.

PDCPG is smaller than the number of times PDCPG surpasses it, especially for random workflows. Furthermore, the margins with which PDCPG has been outperformed are smaller, again especially for random workflows. In this sense, we can say that our proposed heuristic performs better for random workflows.

For high granularity workflows, the average makespan obtained by both algorithms is similar (see Table 5.3), but again the faster running-time of the proposed algorithm could make it an attractive alternative in practice (see Table 5.4 for the running times measured on our simulation platform).

The experiments show that the proposed technique for resource mapping can improve the makespan for larger workflows with low granularity. Furthermore, the results indicate that the proposed technique has a fast running-time. To study PDCPG in more detail, further variants of the simulation settings were considered and will be discussed in Chapter 6. These variants concern the way utilisation profiles are populated (e.g., choosing the number of consecutive slots that are assigned the same random load value), setting the resource load so that the number of CPUs available per resource conform to the trend of the number of partner jobs in a workflow, and reducing the workflow granularity to 0.01. For these variations of the simulation setting, it was observed that PDCPG achieves further improvements.

5.2 PDCPG in Dynamic Advance Reservation Environments

PDCPG was introduced in Section 5.1 for static AR environments. To analyse the performance of PDCPG, simulation experiments were carried out for a Grid environment where no failures occur. Real Grid environments are usually dynamic in nature, and resources may become unavailable during or before the execution of a job for various reasons. For a scheduling algorithm to be adopted in real Grid environments, it is necessary to observe its performance also in such dynamic situations. For this purpose, we now consider PDCPG in a dynamic Grid environment that supports AR.

In a dynamic Grid environment, it is possible that a resource fails after jobs have been allocated to it. In that case, it becomes necessary to allocate these jobs (and possibly other jobs that depend on these jobs) to a different resource. *Rescheduling* refers to the process of reallocating jobs after the occurrence of a failure. The decision which jobs to reallocate depends on the specific rescheduling technique. A number of different rescheduling techniques are available (see Section 3.3) for dealing with resource failures. For simplicity, we adopt a very simple rescheduling technique, in which three categories of jobs and their descendants are rescheduled. One category consists of the jobs which were scheduled on the failed resource but whose execution has not started yet. The second category of jobs comprises the jobs that were in running state on the failed resource when it failed. The third category of jobs consists of those jobs that were scheduled on the failed resource and whose execution has finished before the failure occurrence but whose output file(s) are not accessible by their child jobs.

To evaluate the performance of PDCPG, we again compare PDCPG to the AR version of DCPG. It is observed that in a dynamic AR environment, PDCPG again performs better for low granularity but also, unlike in the other scenarios discussed in Section 5.1.4 and 7.2, for high granularity.

Next we describe the problem under consideration. In Section 5.2.1, the simulation

environment and the results are discussed.

Problem Description

The problem is the same as described in Section 5.1.1, except that it is now assumed that Grid resources may fail. When a resource fails during the execution of a workflow, the scheduling algorithm is required to reschedule the affected jobs of the workflow. The objective is to find a schedule (and to react to resource failures by rescheduling affected jobs) such the that actual makespan of the workflow is minimised.

5.2.1 Evaluation

The GridSim simulator is used to simulate the Grid resource model described in Section 2.6 which supports AR. Computing resource failures may occur. The utilisation profiles of the resources and the links are populated at the start of a simulation run. Two users are simulated. One user employs the PDCPG scheduling approach and the other user the DCPG scheduling technique. The two users submit an identical workflow. Only one user can submit the workflow at one time, the other user can submit it after the workflow of the first user has completed. Once the workflow of the first user is completed, for the second user the utilisation profiles of the computing resources and links are restored to the state they were in at the time of the simulation start, so that the workflow of the second user is scheduled in an identical Grid environment. Regarding resource failures, only one fastest resource is made to fail, as this is expected to affect the schedule of the workflow in the most significant way. The start time of the failure and its duration is randomly generated, and it is kept the same for both users. The start time of the failure is generated in such a way that it occurs after some of the jobs of the workflow have already been scheduled and executed.

Simulation Settings: For the simulation we modified the European Data Grid (EDG) testbed (see Section 2.6) by changing the number of CPUs of the fastest resource Torino

(Italy) from 5 to 52, and this resource is also the one that is chosen for the resource failure. The purpose of this modification is to encourage the heuristics to schedule jobs on that resource, so that when the resource fails it will cause both heuristics to reschedule a number of jobs. The results are only considered for those simulation runs in which both scheduling heuristics are affected by the resource failure. The minimum CPU load is set to 85%. To assign the load to a slot in an utilisation profile, a random value is generated as follows:

$$\mathit{CPULoad} = \mathit{TotalCPU} - \mathit{Random}(\mathit{TotalCPU} - 0.85 * \mathit{TotalCPU}).$$

Here, $\mathit{Random}(x)$ is a uniformly distributed random value in the interval from 0 to x. The CPU load values generated in this way are assigned to intervals of 10 consecutive time slots for one experiment and intervals of 2 consecutive time slots for all other experiments. The number of consecutive slots to which the same random load value is assigned affects the fragmentation of the resource capacity in the utilisation profile. Bandwidth utilisation is again kept at zero in order to check the performance of the proposed algorithms when the bottleneck lies in the computation resources.

Workflows: The details of the workflows that have been selected for experimentation are as follows. Four workflows of 30, 60, 90 and 120 jobs are taken from j301_1, j601_1, j901_1 and j1201_1 in the Project Scheduling Problem Library (PSPLIB) [47], respectively, and four random workflows comprising 50, 70, 100, and 150 jobs are generated using the technique described in [58]. The simulation is run approximately 100 times for each of the workflows.

Experiments are done for a high granularity of 1.2 and low granularities of 0.1 and 0.02. Job sizes are generated randomly between 10,000 MI and 15,000 MI using uniform distribution for the granularities 0.1 and 0.02, and between 25,000 MI and 30,000 MI for granularity 1.0. Job output file sizes are varied and generated randomly using uniform distribution to achieve different granularities. For granularity 1.2, job output file sizes are chosen between 10 and 20 MB, for granularity 0.1 between 100 and 150 MB, and for granularity 0.02 between 320 and 370 MB.

Results: PDCPG is compared with DCPG regarding the workflow makespan. Since the resource failure causes the heuristics to reschedule the workflow, the makespan is calculated as the difference between submission time of the workflow (before the failure occurs) and the completion time of the last job of the workflow (after the failure has occurred and part of the workflow has been rescheduled). In the tables with results, we show the makespan of the initial schedule (i.e., the schedule computed for the workflow before resource failure) and the makespan of the actual schedule (i.e., the schedule in which the jobs affected by resource failure have been rescheduled).

For all granularities, but for different workflows, the performance of PDCPG is significantly better than that of DCPG. The best performance is observed for granularity 1.2, where the minimum improvement is 21.49% and the maximum is 39.27% (see Table 5.7 and Figure 5.5). In case of granularity 0.02 and 0.1 (see Tables 5.5 and 5.6, and Figures 5.3 and 5.4), the improvements are also very significant, ranging from 6% to 24.5% for most workflows, with just four cases where the improvement is less than 6%.

It can be observed from the results that in many cases the improvement in the makespan of the actual schedule of PDCPG compared to DCPG is greater than the improvement in the makespan of the initial schedule. There could be various reasons for this in each experiment. A common reason is that for the initial schedule both heuristics tend to schedule many jobs on the fastest of the resources (which is Torino (Italy), see Table2.1) and the overall schedule is affected by this decision. We enforce the failure of the fastest resource, after which the heuristics have multiple options to choose for rescheduling. DCPG always prefers the fastest resources, while PDCPG also considers the resources on which partner jobs have been scheduled. Thus there is an increased chance that DCPG may distribute the workflow jobs over multiple fast available resources, while PDCPG keeps jobs on fewer available fast resources, thus reducing the communication cost. Other factors that can affect the improvement of PDCPG over DCPG are the granularity, the structure of the workflow, the available resources, and the amount of fragmentation in the resource capacity. The latter effect is analysed further as follows. In the case of the experiments for granularity 0.02 (Table 5.5), the improvement of PDCPG

	PDCPG Act	DCPG Act	Improv %age	PDCPG Init	DCPG Init	Init Improv %age
J301_1PSPLIB	655	694	5.71	408	450	9.31
J601_1PSPLIB	1207	1267	4.73	972	1051	7.52
J901_1PSPLIB	1420	1567	9.39	1188	1280	7.25
J1201_1PSPLIB	1590	1661	4.26	1348	1400	3.75
50Random	725	961	24.51	553	588	6.01
70Random	972	1269	23.38	782	831	5.89
100Random	1368	1685	18.84	1139	1204	5.44
150Random	1931	2349	17.80	1660	1794	7.46

Table 5.5. Initial and actual average makespans generated by heuristics, and improvement percentage of PDCPG for granularity 0.02.

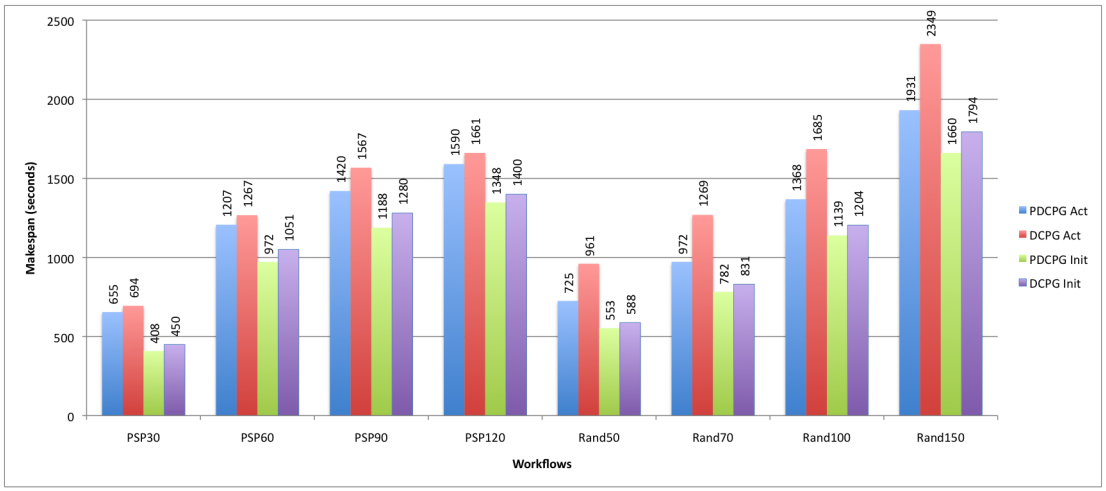


Figure 5.3. Initial and actual average makespans generated by heuristics for granularity 0.02.

	PDCPG Act	DCPG Act	Improv %age	PDCPG Init	DCPG Init	Init Improv %age
J301_1PSPLIB	505	539	6.29	340	367	7.26
J601_1PSPLIB	602	640	5.99	433	455	4.95
J901_1PSPLIB	658	721	8.74	473	499	5.18
J1201_1PSPLIB	779	866	10.06	598	621	3.71
50Random	602	650	7.47	430	441	2.52
70Random	721	789	8.60	542	554	2.16
100Random	1009	1086	7.09	816	825	1.13
150Random	1411	1534	8.01	1179	1199	1.70

Table 5.6. Initial and actual average makespans generated by heuristics and improvement percentage of PDCPG for granularity 0.1.

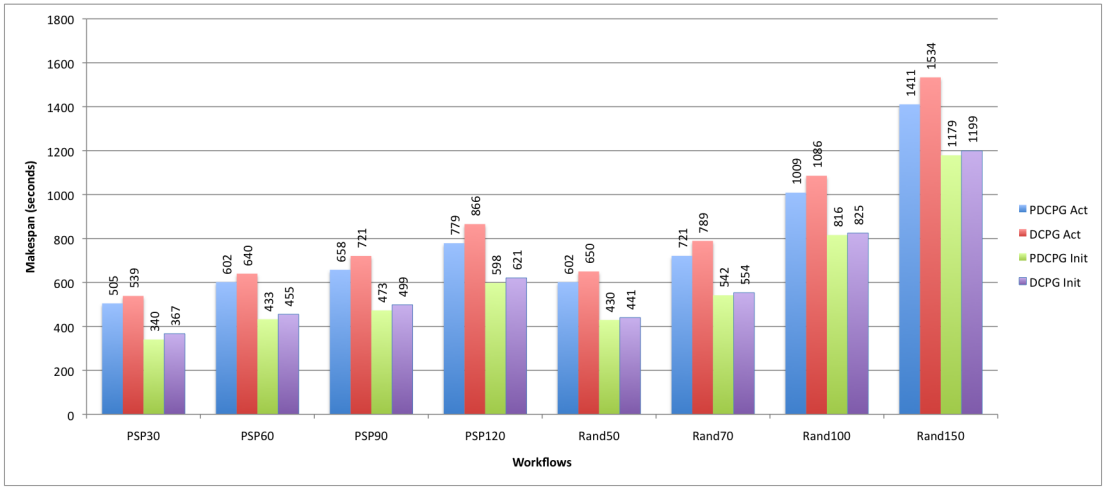


Figure 5.4. Initial and actual average makespan generated by heuristics for granularity 0.1.

	PDCPG Act	DCPG Act	Improv %age	PDCPG Init	DCPG Init	Init Improv/ Worsening %age
J301_1PSPLIB	630	802	21.49	464	491	5.55
J601_1PSPLIB	868	1279	32.13	670	684	2.11
J901_1PSPLIB	1103	1740	36.64	902	922	2.14
J1201_1PSPLIB	1351	2224	39.27	1143	1160	1.41
50Random	845	1145	26.18	655	650	-0.77
70Random	1086	1515	28.29	889	874	-1.62
100Random	1481	2101	29.51	1269	1251	-1.43
150Random	2159	3072	29.72	1905	1877	-1.48

Table 5.7. Initial and actual average makespans generated by heuristics and improvement/worsening percentage of PDCPG for granularity 1.2.

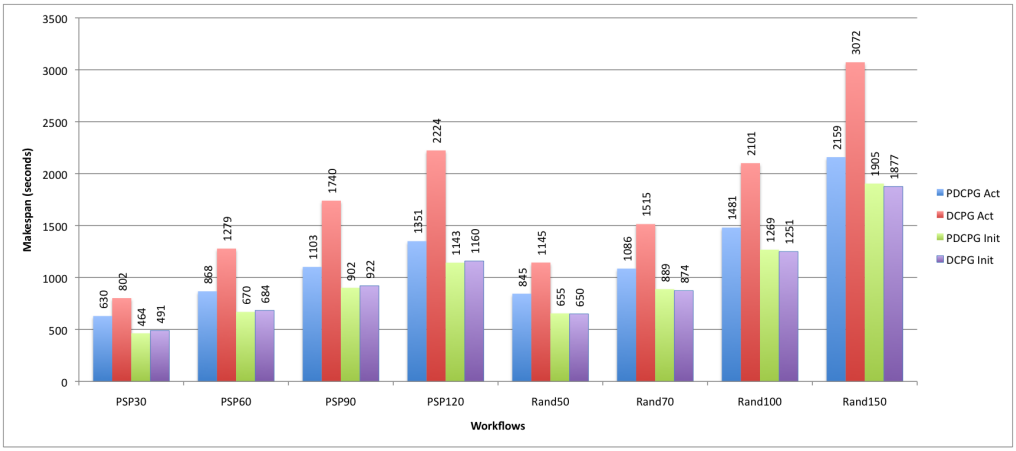


Figure 5.5. Initial and actual average makespan generated by heuristics for granularity 1.2.

	PDCPG Act	DCPG Act	Improv %age	PDCPG Init	DCPG Init	Init Improv/ Worsening %age
J301_1PSPLIB	526	573	8.18	371	402	7.76
J601_1PSPLIB	647	713	9.26	475	517	7.99
J901_1PSPLIB	621	730	14.95	444	475	6.58
J1201_1PSPLIB	872	1010	13.69	677	731	7.37
50Random	813	856	4.98	634	633	-0.12
70Random	813	856	5.07	634	632	-0.38
100Random	1114	1171	4.87	922	911	-1.20
150Random	1552	1628	4.63	1316	1291	-1.95

Table 5.8. Initial and actual average makespans generated by heuristics and improvement/worsening percentage of PDCPG for granularity 1.2, when the number of consecutive slots is set to 10.

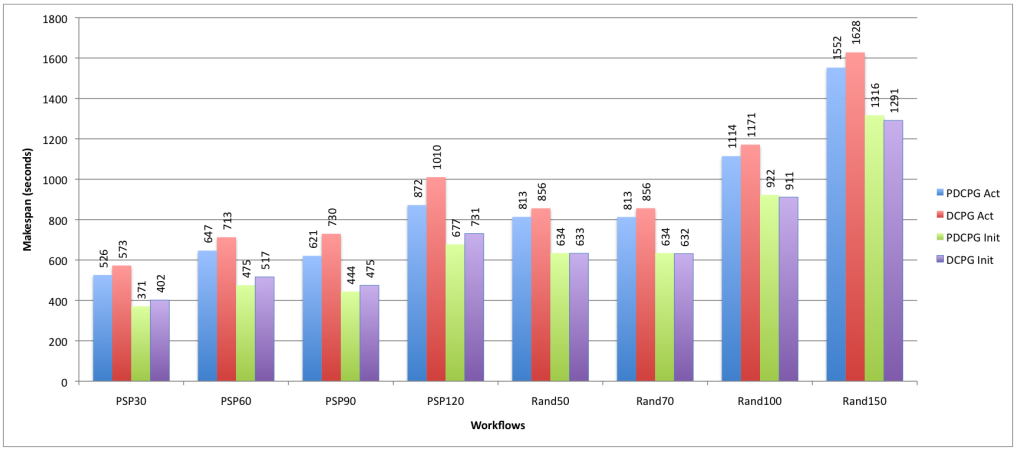


Figure 5.6. Initial and actual average makespan generated by heuristics for granularity 1.2, when the number of consecutive slots is set to 10.

increased hugely for random workflows, which seems to be the combined effect of low granularity, workflow structure and the pattern of available resources conforming to the workflow structure. Here, by *conforming* we mean that the CPUs on the resources are available in such a pattern that in the workflow the partner jobs and their child jobs can be assigned to them without degrading the performance. The improvement of PDCPG becomes smaller in the case of granularity 0.1 (Table 5.6), which seems to be the effect of an increase in granularity. For granularity 1.2, however, the improvement of PDCPG increases by a great margin (Table 5.7). This is because the job length for this experiment was increased from 10,000 MI to 25,000 MI while the size of the resource capacity fragments remained the same. The smaller the resource capacity fragments in the utilisation profile, the harder it is to find a place for the job (i.e., a time interval during which a CPU is available). In this tighter situation, PDCPG accepts an affordable delay in the completion time of partner jobs, allowing it to fit the jobs into the available fragments, which helps getting better results in the end. To support this argument, we show in Table 5.8 the results for an experiment where the resource capacity fragment size was increased. All the settings for this experiments were the same as for Table 5.7, except that the number of consecutive slots that receive the same random load value when the utilisation profiles are populated was changed from 2 to 10. With this change, the improvement percentage of PDCPG decreases, which confirms the above-mentioned effect of capacity fragmentation on the relative performance of PDCPG and DCPG.

Overall the results show that the performance of PDCPG in scenarios where resource failure may occur is encouraging.

5.3 Conclusion

We have presented PDCPG, a new technique for mapping jobs to resources after jobs are selected using the selection method of DCPG, in an advance reservation environment where resources do not fail. The technique used for dealing with advance reservation is a variant of the technique proposed in [70]. For route selection, a modification of Dijk-

stra's algorithm is used. It is shown that PDCPG has improved the workflow makespan significantly and quickly especially for larger workflows with low granularity. The lower execution time of PDCPG is obtained by direct investigation and selection of relevant resources for jobs with partner jobs that are already scheduled.

We also investigated the behaviour of PDCPG in an advance reservation environment where resources may fail and the schedule of the workflow may be affected. Results are presented for failure scenarios where jobs of affected workflows are rescheduled. It is observed that PDCPG performs very well against DCPG for both low and high granularities, which makes PDCPG a potential candidate for deployment in a dynamic Grid environment.

It is also observed that the performance of PDCPG is significantly influenced by the structure of the workflow, conformance of the pattern of resource availability with the structure of the workflow, granularity of the workflow and the amount of capacity fragmentation in the utilisation profiles of the resources.

Chapter 6

Hybrid Dynamic Critical Path for Grids (HDCPG)

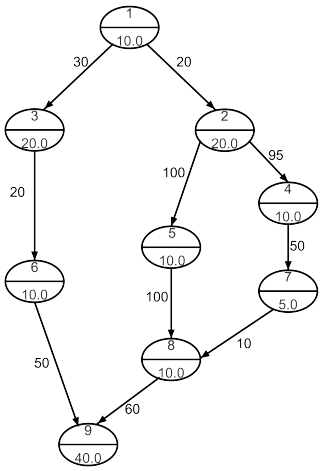
In Chapter 5 we presented PDCPG, a workflow scheduling heuristic for advance reservation environments. Even though PDCPG outperformed DCPG in the low granularity scenario, it was observed that neither of the two algorithms is consistently better than the other in all cases. This inconsistency shows that different heuristics may perform better than others in different situations. Based on this observation, in this chapter we propose a combination of multiple heuristics. We refer to this combination as Hybrid Dynamic Critical Path for Grids (HDCPG). Our results show that HDCPG performs very well against DCPG, PDCPG and HEFT for low granularity, while for high granularity HEFT seems the better option over HDCPG because of faster execution time and only a marginal difference in performance.

The scheduling problem under consideration in this chapter is the same as the one that was discussed in Section 5.1.1. In Section 6.1 we present a novel job selection technique that is used in HDCPG. In Section 6.2, we present the details of HDCPG and its performance evaluation. Conclusions are given in Section 6.3.

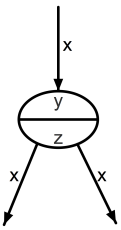
6.1 A Novel Job Selection Technique

Before introducing a new critical path method, a problem that is encountered in the general critical path approach is described. The problem occurs in multiprocessor environments, as discussed in [52]. It is pointed out that wrong decision making by a heuristic at any stage of selecting a job from the workflow may result in a makespan that is far from optimal. The reason is the inability of the heuristic to see the workflow globally. A modified version of an example given in [52] to illustrate this is shown in Figure 6.1. To schedule the workflow, we have two fully available processors of the same speed. The execution time of each job is shown as the label z, shown in the corresponding node of the DAG. The duration of the transmission of the output file of a job to a child job on another processor is shown as the label x of the corresponding edge between these two jobs. For this example we present a hypothetical critical path based algorithm, it will be referred to as *CPA*. It computes the critical path by adding the computation and communication costs of the jobs in the workflow, selects the most critical job to be assigned to the processor which can finish it earliest.

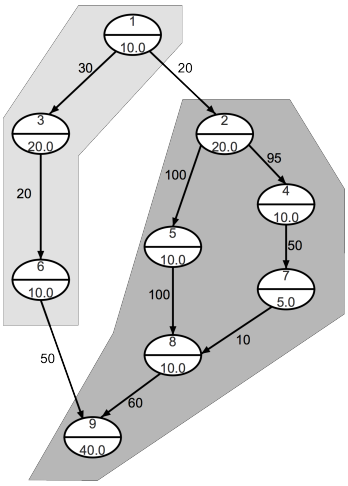
If the workflow in Figure 6.1a is scheduled on a single processor, it can be completed in 135 units. The optimal schedule of this workflow on two processors has makespan 125 (see Figure 6.1d). For the same workflow CPA computes the critical path (1,2,5,8,9) of length 370 units, its schedule will take 145 units to complete (see Figure 6.1c), which is worse than the serial (sequential) schedule. In this scenario, the jobs (1,2,4,5,7,8,9) will be executed on one processor and the jobs (3,6) on the other processor because of the chosen critical path. Even though dynamic critical path methods may change the critical path after scheduling a job, in this case the change comes too late. Had the critical path been (1,3,6,9) from the start, the schedule produced would have been optimal. The main difficulty in calculating the right critical path is that the communication costs between parent and child job may be reduced to 0 in the actual schedule if the jobs are scheduled on the same resource, but initially it is not known which jobs will be scheduled on the same resource.



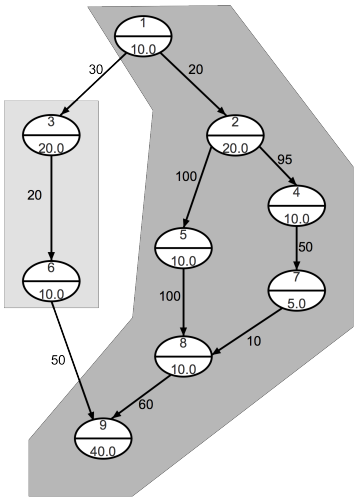
(a) Serial Schedule=135, Critical Path Length(1,2,5,8,9)=370



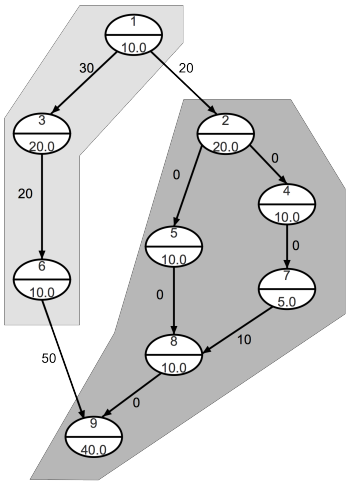
(b) x = Communication Delay, y=Node Identifier, z=Node Weight



(c) Optimal Schedule=125



(d) Critical Path Length (1,2,5,8,9)=370, Schedule=145



(e) New Critical Path Length (1,3,6,9)=180, Schedule=125

Figure 6.1. Modified example of scheduling failure [52].

It is very difficult to tell in advance the choice of which path as critical path can give a better schedule, and the best solution to one problem may not work well for other problems. The factors which can be considered while designing a critical path technique are the structure of the workflow, the relationship between computation and communication costs of jobs in the workflow and the available resources. To achieve a better critical path for the problem under consideration, we try to convert the communication costs between jobs to zero before the scheduling process starts which may become actually zero after the jobs are scheduled on the same resource because of the higher communication cost between them. The criterion to convert communication cost to zero is based on the relationship of the ratio of the cost of a job and the cost of an incoming edge to the job with the granularity of the workflow.

We address the problem illustrated in the example as follows. The workflow is represented as a DAG $G = (V, E)$, where V is the set of n nodes and E is the set of m edges connecting these nodes. $IE_v \subseteq E$ denotes the set of incoming edges to the node $v \in V$. The cost of the node $v \in V$ in duration of execution is denoted by $cost(v)$ and the cost of incoming edge $ie \in IE_v$ in duration of data transfer is denoted by $cost(ie)$. The time unit for both $cost(v)$ and $cost(ie)$ is the same. We consider the following modified definition of workflow granularity:

$$Granularity = \frac{1}{m}\left(\sum_{ie=e(u,v)\in E} \frac{cost(v)}{cost(ie)}\right). \tag{6.1}$$

The $Granularity$ of a workflow is the average, taken over all m edges of the workflow DAG, of the cost of the job at the head of the edge, divided by the the cost of the edge. In our modified critical path method, we convert the cost of an edge $ie \in IE_v$ to zero if

$$\frac{cost(v)}{cost(ie)} < Granularity.$$

In Figure 6.1e, it is shown that the cost of some edges is converted to zero, which changes the critical path to (1,3,6,9), leading to an optimal schedule.

The reason for considering the cost of an incoming edge relative to the cost of the target node in the calculation is its influence on the decision where to schedule its target node.

By comparing the ratio with the $Granularity$ of the workflow (calculated according to (6.1)), we take into account its relationship with the remaining edges. We believe that there is room to develop a more intelligent way of deciding when to convert the cost of an edge to zero by considering a combination of workflow-level metrics (in our case, $Granularity$), the variation in the computation and communication ratios, the relationship between a single edge cost and the costs of other edges, and possibly some other factors.

Following the above-mentioned new technique, we calculate *AEST* and *ALST* of the jobs (an in the critical path calculation by DCPG) in the workflow. The conversion of certain edge costs to zero is only for the purpose of calculating the criticality of the nodes, which affects job selection. During the resource selection step, the actual communication cost is considered.

6.2 HDCPG

We propose a technique in which we compute five different schedules in advance and select the best among them for the actual schedule. The resource selection mechanism of PDCPG is combined with three job selection techniques. The first is the job selection method of DCPG; this combination will be referred to as P. The second job selection method is the new one described above in Section 6.1, which we refer to as *CTZ* (Communication to Zero); the resulting combination will be referred to as Pz. The third job selection method is the ranking-based job selection method of HEFT; this combination will be referred to as Ph. The remaining two algorithms are obtained by combining the resource selection method of HEFT with two job selection methods: The first combination is with its own job selection method , which will be referred to as H. The second combination is with a modified version of the HEFT job ranking technique, which will be referred to as Hz. This modified version of the HEFT job selection technique is obtained by using the same technique we used to obtain *CTZ*. Before assigning the ranks to the jobs, the cost of edges is reduced to zero on the basis

of the granularity based criterion described in Section 6.1. The five algorithms P, Pz, Ph, H and Hz are combined to produce the hybrid algorithm HDCPG. Furthermore, the job ranking technique of H and Hz is influenced by the AR. The communication cost of the jobs is calculated based on the requested bandwidth not the average/maximum bandwidth.

The hybrid algorithm computes five different schedules using the five algorithms, and then uses the one with smallest makespan as the actual schedule. We remark that the execution time overhead of hybrid scheduling can be minimised if the five algorithms are run in parallel.

6.2.1 Evaluation

HDCPG is compared with respect to workflow makespan with PDCPG, DCPG and HEFT in a simulation environment for different workflows. Before presenting the analysis, the experimental environment is described. The GridSim [67] simulator is used for the simulation of the Grid environment for a single user. The selected resource model is described below.

Resource Model

A subset of the European Data Grid (EDG) testbed (see Section 2.6) is chosen for the simulation. 11 resources are simulated as clusters of equally rated CPUs, the rating is in terms of *MIPS*. CPU speed ranges from 1000 to 1330 MIPS. Experiments are conducted for a high load of 90% or above on resources and for workflows of high and low granularity. Load is the percentage of CPUs of a resource under utilisation at any time. The CPU load in each interval of 5–10 consecutive slots of the utilisation profile vector is generated by generating a uniform random value between 0 and the total number of CPUs, and taking the maximum of that value and the target CPU load (e.g., 90% of the CPUs).

The range for low granularity is 0.01 to 0.06, and for high granularity it is 0.8 to 1.5. To specifically check the performance of scheduling algorithms (rather than the routing

	DCPG	PDCPG	HEFT
J301_1PSPLIB	22.03	16.29	17.18
J601_1PSPLIB	19.84	13.35	15.85
J901_1PSPLIB	19.60	13.67	13.44
J1201_1PSPLIB	20.04	13.81	17.05
50Random	16.75	8.95	13.25
70Random	19.08	13.96	13.76
100Random	20.50	21.35	15.30
150Random	25.50	19.89	17.32

(a) Improvement percentage of HDCPG over DCPG, PDCPG and HEFT for low granularity.

	DCPG	PDCPG	HEFT
J301_1PSPLIB	5.11	3.21	3.70
J601_1PSPLIB	9.08	7.75	2.08
J901_1PSPLIB	10.26	9.51	1.66
J1201_1PSPLIB	8.33	7.67	2.77
50Random	8.46	8.95	0.99
70Random	6.93	7.76	0.67
100Random	7.94	9.44	0.32
150Random	8.29	9.79	0.61

(b) Improvement percentage of HDCPG over DCPG, PDCPG and HEFT for high granularity.

Table 6.1. Improvement of HDCPG.

aspect for data transmissions), sufficient bandwidth is made available on all network links. For each simulation run, new randomly generated values are assigned to all utilisation profiles.

Workflows

The workflows chosen for the simulation experiments are as follows. Four workflows of 30, 60, 90 and 120 jobs are taken from j301_1, j601_1, j901_1, j1201_1 in the Project Scheduling Problem Library (PSPLIB) [47], respectively. These workflows will be referred to as *PSP* workflows. Four random workflows comprising 50, 70, 100, and 150 jobs are generated using the technique described in [58]. For each of the workflows, the simulation is run approximately 100 times. For low granularity, the jobs are assigned lengths ranging from 10,000 MI to 15,000 MI randomly according to a uniform distribution, and their output file sizes range from 150 to 270 Megabytes. For high granularity, the jobs are assigned lengths ranging from 30,000 MI to 40,000 MI randomly according to a uniform distribution, and their output file sizes range from 30 to 50 Megabytes.

Results

It is obvious that HDCPG cannot be outperformed by DCPG, PDCPG and HEFT. One of the objectives of the study is to find the margin of improvement. The other objective is to observe the behaviour of the different algorithms for different workflows with different granularity. From the experiments it can be noted that the good instances and the bad

	P	Pz	Ph	H	Hz
J301_1PSPLIB	6.86	0.48	8.31	5.85	-0.45
J601_1PSPLIB	7.49	4.75	8.44	4.74	1.86
J901_1PSPLIB	6.86	1.41	2.85	7.12	-1.02
J1201_1PSPLIB	7.23	1.41	1.82	3.61	4.92
50Random	8.56	1.95	11.02	4.03	1.68
70Random	5.95	-3.07	11.77	6.17	-0.08
100Random	-1.08	-6.17	8.33	6.15	-4.64
150Random	7.01	2.90	10.33	9.89	4.85

(a) Improvement percentage of individual algorithms over DCPG for low granularity.

	P	Pz	Ph	H	Hz
J301_1PSPLIB	1.97	0.44	2.59	1.47	1.57
J601_1PSPLIB	1.44	0.11	7.88	7.15	7.99
J901_1PSPLIB	0.82	-0.09	9.80	8.74	8.48
J1201_1PSPLIB	0.72	-1.12	7.92	5.72	5.94
50Random	-0.53	-2.57	5.64	7.55	7.70
70Random	-0.90	-3.09	4.30	6.30	5.82
100Random	-1.66	-3.68	4.99	7.64	7.58
150Random	-1.66	-2.94	5.33	7.73	8.00

(b) Improvement percentage of individual algorithms over DCPG for high granularity.

Table 6.2. Improvement of individual algorithms over DCPG.

	P	Pz	Ph	Hz
J301_1PSPLIB	1.06	-5.71	2.61	-6.70
J601_1PSPLIB	2.88	0.00	3.88	-3.03
J901_1PSPLIB	-0.27	-6.15	-4.59	-8.76
J1201_1PSPLIB	3.76	-2.28	-1.86	1.36
50Random	4.72	-2.16	7.28	-2.45
70Random	-0.24	-9.86	5.96	-6.67
100Random	-7.70	-13.12	2.33	-11.50
150Random	-3.20	-7.76	0.49	-5.59

(a) Improvement percentage of individual algorithms over HEFT for low granularity.

	P	Pz	Ph	Hz
J301_1PSPLIB	0.51	-1.04	1.14	0.10
J601_1PSPLIB	-6.15	-7.58	0.79	0.91
J901_1PSPLIB	-8.68	-9.67	1.16	-0.28
J1201_1PSPLIB	-5.30	-7.26	2.34	0.23
50Random	-8.74	-10.95	-2.07	0.16
70Random	-7.68	-10.02	-2.14	-0.51
100Random	-10.07	-12.26	-2.88	-0.07
150Random	-10.18	-11.57	-2.61	0.29

(b) Improvement percentage of individual algorithms over HEFT for high granularity.

Table 6.3. Improvement of individual algorithms over HEFT

	P	Pz	Ph	H	Hz
J301_1PSPLIB	35	22	38	19	16
J601_1PSPLIB	25	22	25	23	21
J901_1PSPLIB	24	26	22	17	13
J1201_1PSPLIB	28	12	16	22	28
50Random	31	19	40	13	13
70Random	30	13	27	22	16
100Random	29	17	24	18	18
150Random	30	15	22	10	24

(a) Count of contribution to best schedule for low granularity.

	P	Pz	Ph	H	Hz
J301_1PSPLIB	53	38	57	35	33
J601_1PSPLIB	13	10	67	37	53
J901_1PSPLIB	0	0	68	28	21
J1201_1PSPLIB	3	1	79	18	22
50Random	3	3	35	62	67
70Random	10	4	29	56	57
100Random	1	0	12	62	59
150Random	0	0	12	51	67

(b) Count of contribution to best schedule for high granularity.

Table 6.4. Count of contribution to best schedule out of 100 by individual algorithms in HDCPG.

instances are different for the different individual algorithms. Thus, the hybridisation of these algorithms leads to a significant improvement over all the individual algorithms.

HDCPG gives a huge improvement over the three other algorithms for low granularity, and a considerable improvement is obtained over DCPG and PDCPG for high granularity (Table 6.1). HEFT performed very well against HDCPG in the high granularity scenario. The improvement of HDCPG over HEFT is small especially for random workflows. This strong performance of HEFT may make it a better choice than HDCPG for high granularity scenarios.

If the performance of the individual algorithms that make up HDCPG against DCPG is analysed (Table 6.2), then P, Ph and H show a good improvement for low granularity for all workflows except for the 100 jobs random workflow where P is outperformed by a small margin. Pz and Hz shown a mixed performance against DCPG for low granularity. These two gave improvements for some workflows and are outperformed for others. For high granularity, Ph, H and Hz clearly performed well against DCPG. P gave an improvement for PSP workflows but fell short for random workflows. The difference in both cases is very small, so we can say that the performance of P and DCPG was almost the same. Pz is outperformed by DCPG by small margins for most of the workflows.

The performance of the individual algorithms against HEFT is less clear-cut than the performance against DCPG (Table 6.3). For low granularity, P has shown mixed performance by giving an improvement for half of the workflows and worse performance for the other half of the workflows. Pz and Hz have shown a considerable worsening of the makespan for most of the workflows. Only Ph outperformed HEFT for most of

the workflows by considerable margins. For high granularity, P and Pz are outperformed by HEFT by bigger margins, especially for random workflows. Ph has shown a better performance here, by giving a small improvement for PSP workflows and only a small worsening for random workflows. The performance of Hz is almost the same as that of HEFT.

In Table 6.4, we present the count (in percent) of the number of times individual algorithms produced the best schedule. There are cases in which several algorithms gave the best schedule at the same time, so the percentage values can add up to more than 100%. For low granularity, the contribution by P and Ph clearly dominates. The contribution by Pz, H and Hz is still considerable and cannot be ignored. For high granularity, Ph has contributed very dominantly as compared to all other algorithms for PSP workflows. Its contribution is between 57 and 79 percent. H and Hz have contributed dominantly for the random workflows. The contribution by P and Pz for most of the workflows is zero or very small.

We observe that the HEFT job selection method has a great influence, which makes Ph perform very well especially for the high granularity scenarios where DCPG, P and Pz do not fare well. Furthermore, on the basis of these results the following suggestions can be made. For the low granularity case, the hybridisation is worth adopting as it can produce significant improvements in workflow makespan compared to the schedules produced by any single algorithm. For high granularity scenarios, however, using only one of the algorithms among Ph, H and Hz could be the better option rather than hybridisation.

6.3 Conclusion

The huge improvements obtained by the hybrid method show that the good instances and the bad instances for the individual algorithms are different. None of the five algorithms consistently gives better schedules. By running all five algorithms, there is a good chance that one of them produces a good schedule, and thus the hybrid method gives a strong improvement over the individual algorithms. In the future, it may be worth exploring

new combinations, which might improve the results further.

Chapter 7

PDCPG for Non-Advance Reservation Environments

In Section 5.1 a partner-based resource mapping technique, Partner-based Dynamic Critical Path for Grids (PDCPG), was proposed for scheduling Grid workflows in advance reservation (AR) environments. The PDCPG decision rule for selecting a resource for a job is helped by the AR facility, and it was shown that PDCPG performs better than two other well known scheduling heuristics, Dynamic Critical Path for Grids (DCPG) and Heterogeneous Earliest Finish Time (HEFT) (see Sections 4.2 and 3.1), in terms of workflow completion times when the granularity of certain workflows is low. Since many workflow scheduling heuristics have been proposed for non-AR environments, this chapter focuses on the study of the behaviour of PDCPG in an environment which does not support AR. PDCPG is modified according to the non-AR situation and compared again with DCPG and HEFT. It is shown that PDCPG performs even better in terms of workflow completion time in scenarios where the granularity of workflows is low.

In the next section, PDCPG for non-AR environments is explained. Its performance is evaluated in Section 7.2, and conclusions are drawn in Section 7.3.

7.1 PDCPG for Non-AR Environments

For the problem under study, the workflow model is the same as described in Section 2.2 and the model of the Grid environment is the same as described in Section 2.1. Furthermore, it is assumed that Grid resources do not support AR and they do not fail. The objective is to schedule a workflow of jobs so that its makespan is minimised.

For job selection, the same dynamic critical path method is used, but with a little modification. The ADTT calculated for AR environments is modified for non-AR environments by replacing the requested bandwidth with the average bandwidth, which was also used in [58]. ADTT for non-AR environments is calculated as follows:

$$ADTT_j = \frac{JS_j}{Average\_Bandwidth}.$$

For resource mapping, the same partner-based approach is used, but with a modification in the condition used for the resource selection. The modified resource selection condition is as follows:

$$minExpCT_{j'}^{RP} \leq minExpCT_{j'}^{R \setminus RP} + ADTT_{j'}, \tag{7.1}$$

where

$$minExpCT_{j'}^{RP} = \min\{expCT_{j'}^{rp} \mid rp \in RP_{j'}\},$$

is the minimum expected completion time of the job j' on the resources $rp \in RP_{j'}$ (where $RP_{j'}$ denotes the set of resources on which partner jobs of job j' have already been scheduled) and

$$minExpCT_{j'}^{R \setminus RP} = \min\{expCT_{j'}^{r} \mid r \in R \setminus RP_{j'}\}$$

is the minimum expected completion time of the job j' on the resources $r \in R \setminus RP$. If condition (7.1) is satisfied, the resource of a partner job that gives the minimum expected completion time is selected for the scheduling of job j'. Otherwise, any resource which gives the minimum expected completion time is chosen.

The reason for the modification of the condition from (5.1) to (7.1) is as follows. In

non-AR environments, it is comparatively difficult to predict the completion time of a job as compared to the AR environment. One of the reasons of uncertainty in non-AR environments is the dynamic load on the executing system. Based on this factor, the completion time of the partner job is removed from the resource selection condition and the competition is simply between the expected completion time of the job to be scheduled on resources in $R \setminus RP$ and in RP. Priority is still given to the resources in RP by allowing a delay in the completion time within a limit equal to the communication cost of the job to be scheduled plus its minimum expected completion time on a resource in $R \setminus RP$.

7.2 Evaluation

In the following, we refer to the modification of PDCPG for non-AR environments simply as PDCPG. HEFT and DCPG are the heuristics against which PDCPG is compared. Workflow makespan is used as the evaluation criterion. Recall that workflow makespan is the difference between the submission time of the workflow and the completion time of the last job. Eight workflows are selected for comparison purpose. Four workflows of 30, 60, 90 and 120 jobs are taken from j301_1, j601_1, j901_1 and j1201_1 in the Project Scheduling Problem Library (PSPLIB) [47], respectively. The other four workflows are random workflows comprising 50, 100, 150 and 200 jobs as in [58]. To generate these random workflows, the number of levels is set to 10, so that with an increase in the number of jobs in the workflow, the number of jobs per level also increases, which increases the number of partner jobs per level as well.

The non-AR environment is simulated using the GridSim [67] simulator. The resource model of a subset of the European Data Grid (EDG) [16] is chosen (see Table 2.1). Experiments are conducted for three different granularities, which are 0.01, 0.1 and 1.0. The ranges of job sizes and file sizes, generated using uniform distribution, for the different granularities are as follows.

- Job size from 60000 to 65000 MI and file size from 400 to 450 MB for granularity

	DCPG	HEFT
30PSP	19.79%	33.53%
60PSP	20.93%	24.38%
90PSP	15.18%	18.44%
120PSP	15.77%	18.21%
50Rand	8.70 %	10.17%
100Rand	8.22%	22.99%
150Rand	7.72%	12.35%
200Rand	2.24%	9.05%

Table 7.1. Improvement of PDCPG over heuristics for granularity 0.01

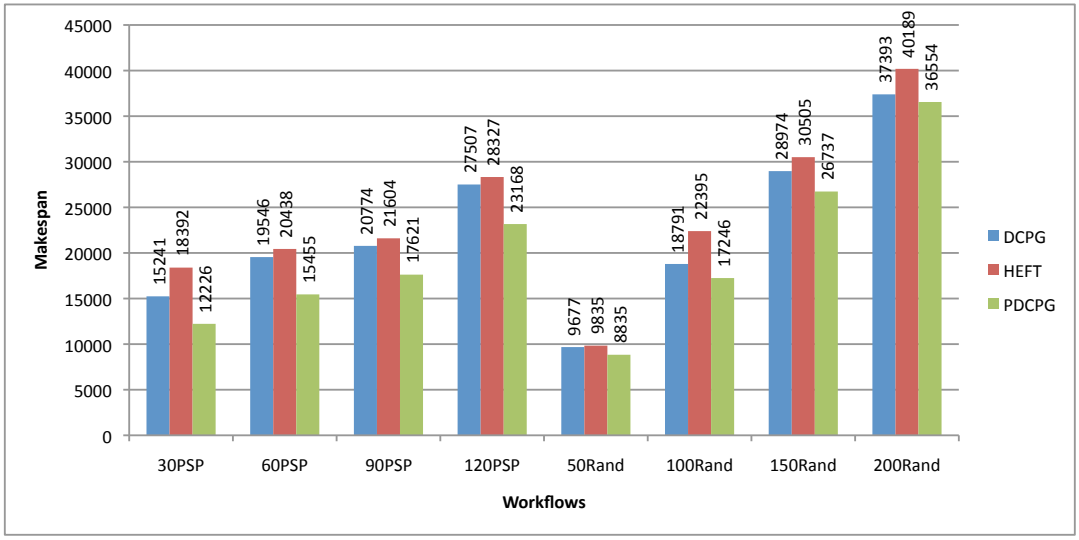


Figure 7.1. Makespan (in seconds) by heuristics for granularity 0.01.

	DCPG	HEFT
50Rand	12.96%	24.75%
100Rand	9.70%	15.05%
150Rand	7.03%	13.03%
200Rand	8.40%	9.64%

Table 7.2. Improvement of PDCPG over heuristics for granularity 0.01. Random workflows have maximum 8 jobs per level.

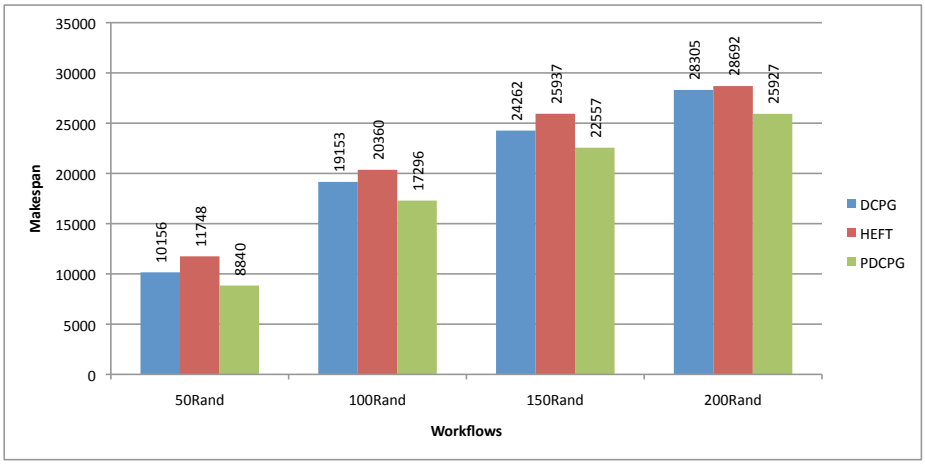


Figure 7.2. Makespan (in seconds) by heuristics for granularity 0.01. Random workflows have maximum 8 jobs per level.

	DCPG	HEFT
50Rand	12.25%	19.46%
100Rand	11.09%	11.96%
150Rand	8.21%	9.40%
200Rand	3.15%	12.98%

Table 7.3. Improvement of PDCPG over heuristics for granularity 0.01. Random workflows have maximum 6 jobs per level.

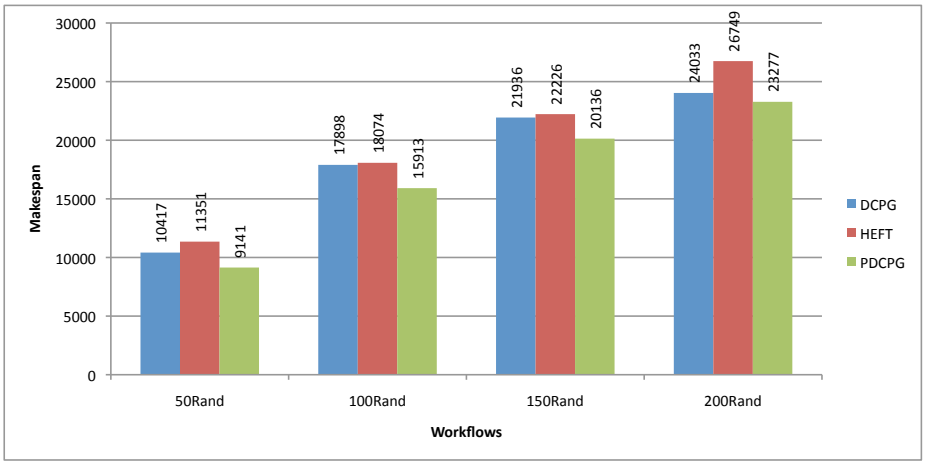


Figure 7.3. Makespan (in seconds) by heuristics for granularity 0.01. Random workflows have maximum 6 jobs per level.

	DCPG	HEFT
30PSP	10.81%	12.56%
60PSP	16.33%	20.04%
90PSP	12.30%	12.28%
120PSP	9.44%	13.49%
50Rand	0.72 %	6.51%
100Rand	3.96%	8.26%
150Rand	2.69%	8.24%
200Rand	1.84%	7.76%

Table 7.4. Improvement of PDCPG over heuristics for granularity 0.1.

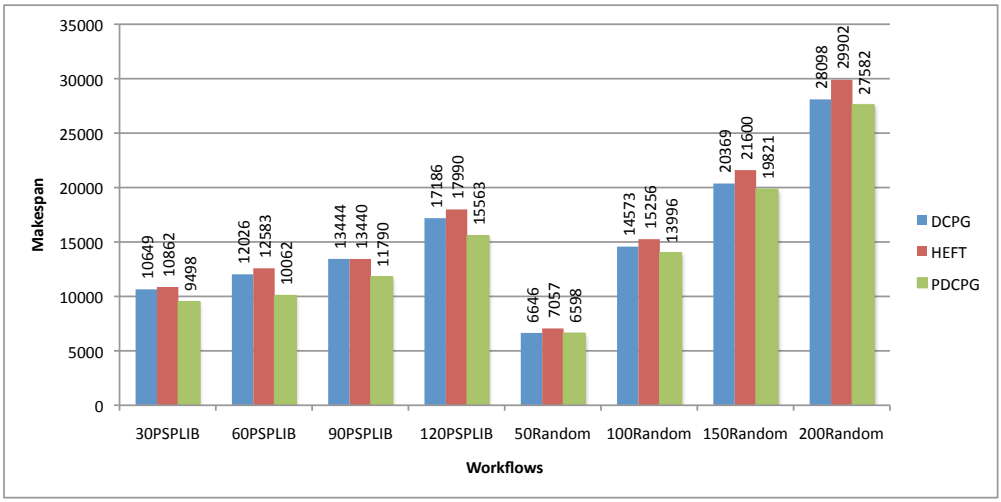


Figure 7.4. Makespan (in seconds) by heuristics for granularity 0.1.

	DCPG	HEFT
30PSP	0.82%	-0.03%
60PSP	3.75%	3.34%
90PSP	3.22%	3.06%
120PSP	1.49%	0.12%
50Rand	1.79%	-3.57%
100Rand	1.81%	-2.56%
150Rand	-0.45%	-3.18%
200Rand	-0.77 %	-4.45%

Table 7.5. Improvement/Worsening of PDCPG over heuristics for granularity 1.0

Figure 7.5. Makespan (in seconds) by heuristics for granularity 1.0.

0.01.

- Job size from 60000 to 65000 MI and file size from 150 to 270 MB for granularity 0.1.
- Job size from 1500000 to 1505000 MI and file size from 100 to 150 MB for granularity 1.0.

The simulations are run 100 times for each experiment, and the average makespan is shown in the results. The results are discussed in order of ascending granularity.

Figure 7.1 shows the average makespan produced by the heuristics and Table 7.1 shows the percentage of improvement in makespan of PDCPG over the two other heuristics for granularity 0.01. The improvements are significant, especially over HEFT, where they range from 9% to 33.5%. In this comparison, HEFT is the least efficient among the three heuristics for all workflows. This is because of the static ranking technique for job selection, as the bigger difference between the communication and computation cost

in the low granularity case may significantly change the critical path in the scheduling process. The dynamic critical path selection method kept both PDCPG and DCPG ahead of HEFT, which is not the case for high granularity experiments (as shown later). This behaviour of HEFT is similar to the ones noted in [58, 29, 15] and mentioned in Section 3.1.2. Another reason for the huge improvement over HEFT seems to be related to the structure of the workflows, especially PSPLIB workflows, and the resource model having multiprocessors per resource for which dynamic critical path methods seem more suitable. Against DCPG, the performance of PDCPG is significantly better for PSPLIB workflows as compared to random workflows. For PSPLIB workflows the improvement ranges from 15% to 20%. For random workflows, the maximum improvement is 8.7%. A factor which influences the performance of PDCPG is whether the number of available CPUs on a faster computing resource matches the number of partner jobs in a workflow. The PDCPG resource selection technique can work better if it can schedule as many clusters of partner jobs on the same resource as possible. The structure of PSPLIB workflows and the simulated resource models constitute a better combination for PDCPG, whereas the way random workflows are generated makes them less suitable than the PSPLIB workflows. To study this relationship between workflow structure and the resource model, we also generated random workflows differently from the above ones. We generated random workflows in such a way that at each level the maximum number of jobs remains 8 and 6, respectively. The average makespan given by the heuristics is shown in Figure 7.2 and Figure 7.3, and the improvement achieved by PDCPG is shown in Table 7.2 and Table 7.3. The improvement of PDCPG over both heuristics is increased as compared to the results shown in Table 7.1, which supports the hypothesis about the influence of the relationship between workflow structure and the number of available CPUs on computing resources on the performance on PDCPG.

Figure 7.4 gives the average makespan produced by the heuristics, and Table 7.4 gives the improvement obtained by PDCPG, for granularity 0.1. The improvement decreases as the granularity is increased. This is also noticeable in Table 7.5, which gives the results for granularity 1.0 (the average makespans are shown in Figure 7.5). The improvement

or worsening of PDCPG for granularity 1.0 is marginal. Based on these results we can say that PDCPG exhibits better performance for low granularity and performs almost the same as other popular heuristics for the high granularity case.

7.3 Conclusion

PDCPG was originally proposed for the AR environment and has been studied here for the non-AR environment. The simulation experiments show that PDCPG performs better than two well known heuristics, especially when the granularity is low. These results further strengthen the potential of PDCPG for being adopted by the Grid community along with other popular heuristics.

Chapter 8

K-Shortest Path Routing Technique in Advance Reservation Environment

In this chapter we consider the routing problem in AR environments, which occurs as a subproblem in workflow scheduling when a data file needs to be transmitted from one resource to another. In Section 8.1 we analyse the complexity of different variants of the routing problem in the AR environment. Section 8.2 discusses shortcomings of Dijkstra's greedy algorithm and two polynomial time algorithms proposed by Varvarigos et al. [70] for finding a path with earliest arrival time in the AR environment. A K-Shortest Path *(KSP)* variant aimed at addressing these shortcomings is proposed in Section 8.3, and an evaluation of the newly proposed routing algorithm is performed in Section 8.4. Conclusions are given in Section 8.5.

8.1 Complexity Analysis of Routing Problems in the Advance Reservation Environment

First, the general routing problem in the AR environment is defined, then a number of specific cases of the general problem will be analysed. We denote the general Advance Reservation Routing Problem by *ARRP*. Given is a directed graph $G = (V, E)$, where

V is the set of vertices/nodes and E is the set of edges/links connecting the nodes. Each edge $e \in E$ has a delay d_e and a capacity/bandwidth denoted by C_e. Each edge $e \in E$ also has a *utilisation profile vector* U_e which stores the information regarding the reserved capacity at any future time $\theta \in \{0, \cdots, T\}$, where T is the maximum time for which utilisation information is available and thus the dimension of U_e. The unit of the link delay and the smallest time for which a reservation can be made is assumed to be the same. $S \in V$ is the source node from where data is to be transferred to the destination node $D \in V$. For the data transfer, *rbw* bandwidth is requested, and b is the duration of the transfer if *rbw* bandwidth is utilised. It is assumed that d_e, C_e, *rbw* and $U_e(\theta)$ are integers. An edge is said to be *feasible* for *rbw* bandwidth at time θ if $C_e - U_e(\theta) \geq$ *rbw*, and it is unfeasible otherwise. Note that an instance of *ARPP* has size $\mathcal{O}(|V| + |E|T)$, as each edge has an utilisation profile of size $\mathcal{O}(T)$.

The network is assumed to be an OBS network, in which data is sent in a single burst from the source node to the destination node, without any queueing at intermediate nodes. The data burst can enter a link e at time t if e is feasible for *rbw* bandwidth at times $t, t+1, \ldots, t+b-1$. In that case, the first bit of data will arrive at the endpoint of the link at time $t + d_e$. A path $\mathcal{P}$ from S to D is said to be feasible at time t if each edge e on $\mathcal{P}$ is feasible at times $A_e, A_e + 1, \ldots, A_e + b - 1$, where A_e is the time when the first bit of data enters the link e, i.e., A_e is equal to t plus the sum of the delays of all edges that precede e on the path $\mathcal{P}$. If the path $\mathcal{P}$ enters the same edge e multiple times, the available capacity of the edge must be at least the appropriate multiple of *rbw*. Formally, if the edge e is entered k times and the entry times are $A_1, A_2, \ldots, A_k$, then we must have

$$C_e - U_e(\theta) \geq \textit{rbw} \cdot |\{1 \leq i \leq k \mid \theta \in [A_i, A_i + b - 1]\}|$$

for all $\theta \in \{0, \ldots, T\}$.

If the path $\mathcal{P}$ is feasible at time t, the data can be transmitted over path $\mathcal{P}$ at time t and will be completely received at the destination at time $AT(\mathcal{P}, t) = t + b + d_{\mathcal{P}}$, where

$d_{\mathcal{P}} = \sum_{e \in \mathcal{P}} d_e$ is the total delay of path $\mathcal{P}$. The objective is to find a path $\mathcal{P}$ from S to D and a start time t such that $\mathcal{P}$ is feasible at time t and has the minimum arrival time $AT(\mathcal{P}, t)$. Only paths with arrival time at most T are considered.

For a path $\mathcal{P}$, we say that the arrival time of $\mathcal{P}$ is $AT(\mathcal{P}, t)$, where $t \geq 0$ is the first time at which path $\mathcal{P}$ is feasible.

Different variants of *ARRP* are characterised by the type of path that is allowed (i.e., whether the path is required to be simple or whether repetition of vertices or even edges is allowed) and by restrictions on the range of edge delay values or on the duration of the transmission. A *simple path* is a path in which neither vertices nor edges can be repeated. An *edge-simple path* is a path in which vertices can be repeated but edges cannot be repeated. A *non-simple path* is a path in which vertices and edges can be repeated. S^b_λ denotes the variant of *ARRP* where only simple paths are allowed and the delay of links is λ and the duration of the transfer is b. Similarly, ES^b_λ denotes the corresponding variant of *ARRP* where edge-simple paths are allowed, and NS^b_λ denotes the corresponding variant of *ARRP* where non-simple paths are allowed. Both b and λ can be specified as arb, which stands for arbitrary values ≥ 1.

Definition 8.1.1. (S^1_1 problem) Given the problem *ARRP*, we restrict the path to be a simple path from S to D, $\lambda = 1$ and $b = 1$.

We define the directed Hamiltonian Path Problem *(HPP)*. After that, it is shown that the *HPP* can be reduced to S^1_1.

Definition 8.1.2. (Directed Hamiltonian Path Problem (HPP)) Given a directed graph $G' = (V', E')$, where V' is the set of vertices/nodes and E' is the set of edges/links, and S', $D' \in V'$. Find a path from source S' to destination D' that visits each vertex in the graph exactly once. (Such a path is called a Hamiltonian path.)

HPP is NP-hard [37].

Theorem 8.1.3. S_1^1 *is NP-hard.*

Proof. We reduce the *HPP* to the decision version of S_1^1. The reduction can be explained with the help of Figure 8.1. Let an instance of *HPP* be given by a directed graph $G' = (V', E')$ and S', $D' \in V'$. Let G' have n nodes. Construct an instance of S_1^1 as follows. Obtain the graph $G = (V, E)$ from G' by adding two new vertices S and D and two new edges, one from S to S' and one from D' to D. Set all the edges of G that correspond to edges from G' to be feasible at all times. (For the sake of simplicity we only mention the availability of edges after calculation rather than specifying in detail their capacities, utilisation profiles and the requested bandwidth for the data transfer.) The delay of all edges of G is set to 1. The edge from S to S' is only feasible at time 0 and the edge from D' to D is only feasible at time n. The objective is to decide if there is a simple path from S to D for data of duration $b = 1$ so that the data arrives at D at time $n + 1$.

We claim that G' has a Hamiltonian path from S' to D' if and only if there is a path from S to D in G with arrival time at most $n + 1$. The proof in both directions is as follows.

(a) G' has a Hamiltonian path from S' to $D' \Rightarrow$ there is a path from S to D in G with arrival time at most $n + 1$.

Let H be a Hamiltonian path from S' to D' in G'. Construct $\mathcal{P}$ by taking the edge S-S' followed by H and then D'-D. $\mathcal{P}$ is feasible at time 0 as it enters the edge S-S' at time 0 and the edge D'-D at time n. Thus, $\mathcal{P}$ is a path with arrival time $n + 1$.

(b) There is a path from S to D in G with arrival time at most $n + 1 \Rightarrow G'$ has a Hamiltonian path from S' to D'.

Let $\mathcal{P}$ be a path from S to D with arrival time at most $n + 1$. It must start with S-S' and end with D'-D. As the edge D'-D is only feasible at time n, the path must have arrival time equal to $n + 1$. Let H be the subpath of $\mathcal{P}$ from S' to D'. The data leaves S' at time 1 and reaches D' at time n. Therefore it must traverse

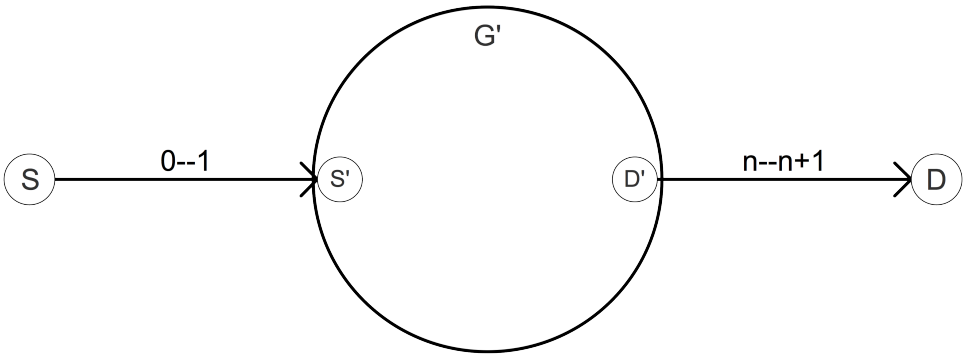


Figure 8.1. A graph G' with n nodes. Source node S and destination node D are connected to the graph. Edge labels show the time at which the edges are feasible for data transmission.

$n-1$ edges between S' and D'. As it is a simple path, it is a Hamiltonian path in G'.

Hence S_1^1 is NP-hard. □

Definition 8.1.4. (S_{arb}^{arb} problem) Given the problem *ARRP*, we restrict the path to a simple path from S to D. It is assumed that $\lambda \geq 1$ and $b \geq 1$ are arbitrary.

Corollary 8.1.5. *S_{arb}^{arb} is NP-hard.*

Proof. The S_1^1 problem is a special case of the problem S_{arb}^{arb}. Theorem 8.1.3 proves that the S_1^1 problem is NP-hard. Hence the S_{arb}^{arb} problem is also NP-hard. □

Before analysing other variants of *ARRP* in which edge-simple and non-simple paths are allowed, we present some examples for the purpose of motivation. Figure 8.2 shows a situation in which the destination node is only reachable through an edge-simple path. The directed graph G in the example has vertex set $V = \{A, B, C, D, E\}$, A is the source node, and E is the destination node. The duration of the data transmission is assumed to be 1. Due to the gap between the availability periods of edges A-D and D-E, no simple path is feasible for data transmission to the destination E. Here, the edge-simple path A-D-C-B-D-E can be used to reach the destination.

Similar to the situation shown in Figure 8.2, there can also be situations in which only non-simple paths can reach the destination. Such a situation is shown in Figure 8.3. The directed graph G in this example has vertex set $V = \{A, B, C, D, E\}$, A is the source

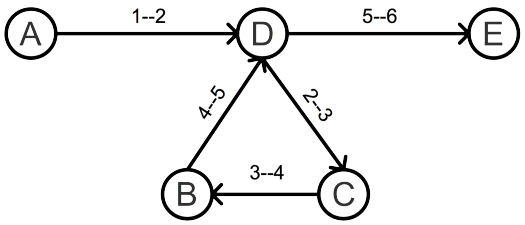


Figure 8.2. Example where the only feasible path from A to E uses node D twice.

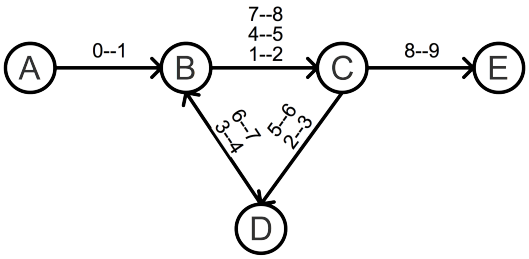


Figure 8.3. Example where the only feasible path from A to E uses edges B-C, C-D and D-B more than once.

node, and E is the destination node. The labels of the edges show their availability periods. The duration of the data transmission is assumed to be 1. Due to the mistimed availabilities of the edges, simple and edge-simple paths do not work. Here, the non-simple path A-B-C-D-B-C-D-B-C-E can be used to reach the destination.

The above mentioned scenarios draw attention to studying problems related to edge-simple and non-simple paths. We next define the ES_1^1 problem and prove it to be NP-Hard by a reduction from a variant of the $3SAT$ problem, which is defined as follows.

Definition 8.1.6. (At-Most-3SAT Problem)

Given a Boolean formula ϕ, which is the conjunction of m clauses over n boolean variables and where each clause is a disjunction of exactly 3 literals, where each literal can be a variable or its negation. Furthermore, each variable is restricted to appear at most three times and each literal at most two times. Find a truth assignment to variables such that ϕ evaluates to *true*. We denote this problem by At-Most-3SAT.

At-Most-3SAT is NP-hard [37].

Definition 8.1.7. (ES_1^1 problem)

Given the problem *ARRP*, we restrict the path to be an edge-simple path from S to D, $\lambda = 1$ and $b = 1$.

Theorem 8.1.8. *ES_1^1 is NP-Hard.*

Proof. To prove that ES_1^1 is NP-Hard we present a reduction from the $\mathrm{At\text{-}Most\text{-}3SAT}$ problem. For a given instance $\mathcal{I}$ of the $\mathrm{At\text{-}Most\text{-}3SAT}$ with m clauses and n variables, we construct a graph as follows. All edges of the constructed graph have delay 1. For each $variable\ x_i$, $1 \leq i \leq n$, we create a gadget of 10 connected nodes. Each such gadget consists of two parallel directed paths of 4 nodes together with an $entry$ node that is connected to the start nodes of the two paths and an $exit$ node to which the end nodes of the two paths are connected. We call the two paths the $upper$ and $lower$ paths. For the sake of simplicity, we only mention and show the availability periods of edges, which means that the available capacity of the edges is more than the requested bandwidth for the respective periods. The idea of the gadget is that a path that traverses the gadget from the $entry$ node to the $exit$ node must use either the upper path (which corresponds to setting the variable to false) or the lower path (which corresponds to setting the variable to true). This leaves the edges on the other path available, and some of these edges can then be used by the path at a later time, which corresponds to satisfying a clause.

Based on the restriction on the occurrence of literals to at most two times in the Boolean formula, only two special edges, called *literal edges*, of each of the two paths in the variable gadget are made feasible at all times so that they can be used in correspondence to a literal occurrence in a clause. The availability of the edges other than the *literal edges* is set in such a way that the data can enter the gadget for variable x_i at time $5(i-1)$ and reach the exit node of the gadget at time $5i$. The *literal edges* are separated by edges with limited availability, so that a path arriving at the variable gadget from a *clause* gadget (to be defined below) can only use one *literal edge* of the *variable* gadget. To achieve this, the availabilities of the edges other than *literal edges* of the gadget are

set as shown in Figure 8.4: For the x_i gadget, set the interval of availability of the edges from $5 \cdot (i-1) + c$ to $5 \cdot (i-1) + c + 1$, where $c \in \{0, 2, 4\}$ is the index of the edge. The edges on each of the two paths in the gadget are indexed by consecutive numbers starting with 0. The index of the outgoing edge from the $entry$ node of the variable gadget is 0, and the indices of the *literal edges* are 1 and 3. The *literal edges* in the upper path can be used for positive occurrences of the variable in clauses while the *literal edges* in the lower path can be used for negative occurrences. For $1 \leq i \leq n-1$, the gadget for variable x_i is connected to the gadget for variable x_{i+1} by identifying the $exit$ node of the x_i gadget with the $entry$ node of the x_{i+1} gadget. The $entry$ node of the first variable gadget is the source node S.

For each clause $\mathcal{C}_j$, $1 \leq j \leq m$, we create a $clause$ gadget with two nodes. We call these nodes the *entry* and *exit* nodes of the clause gadget. The $exit$ node of the nth variable gadget is connected to the $entry$ node of the first $clause$ gadget via an edge. The only feasible paths from the $entry$ node of a clause gadget for clause $\mathcal{C}_j$ to the $exit$ node of the same clause gadget go via a *literal edge* corresponding to one of the literals in the clause, i.e., a *literal edge* of the $upper$ or $lower$ path of the gadget of a variable that occurs in the clause. For each of the three literals in the clause, we choose a *literal edge* of the corresponding *variable* gadget in such a way that no two clauses choose the same *literal edge* (which is possible since each literal occurs at most twice in the given Boolean formula). For each literal in the clause, we add an edge from the $entry$ node of the clause gadget to the start node of the chosen *literal edge* that represents that literal in the corresponding variable gadget, and an edge from the end node of that *literal edge* to the $exit$ node of the clause gadget. This enables a path of three edges that starts at the $entry$ node of the clause and goes through the start node and end node of a *literal edge* and ends at the $exit$ node of the clause. The availability of an edge related to the clause gadget for clause $\mathcal{C}_j$ is set to the interval from $5 \cdot n + 4 \cdot (j-1) + c$ to $5 \cdot n + 4 \cdot (j-1) + c + 1$, where $c \in \{0, 1, 3\}$ is the index of the edge. Here, the index is 0 for the incoming edge to the $entry$ node of the clause, 1 for the edges from the $entry$ node to the start node of a *literal edge*, and 3 for the edges from the end node of

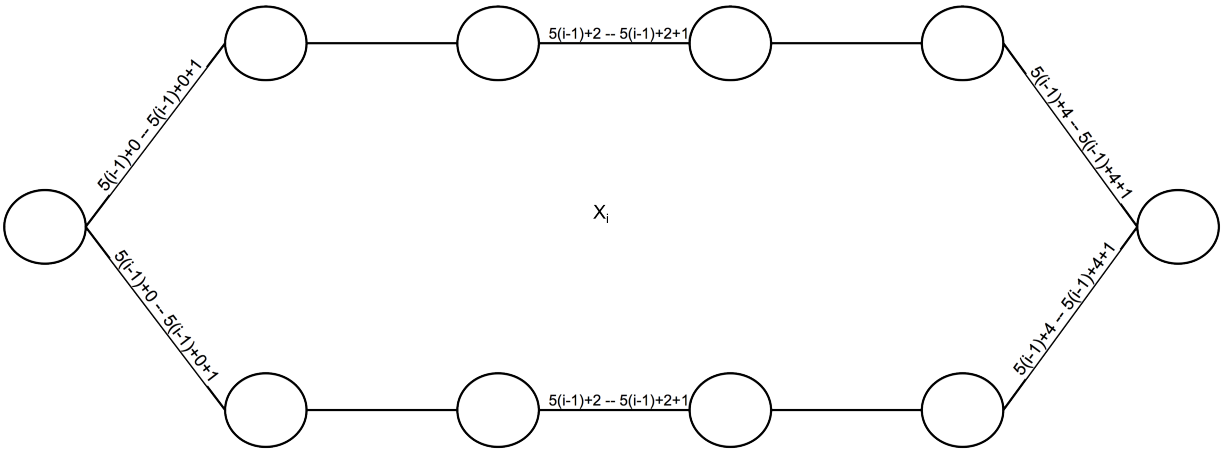


Figure 8.4. General settings for the availabilities of the edges in the variable gadget.

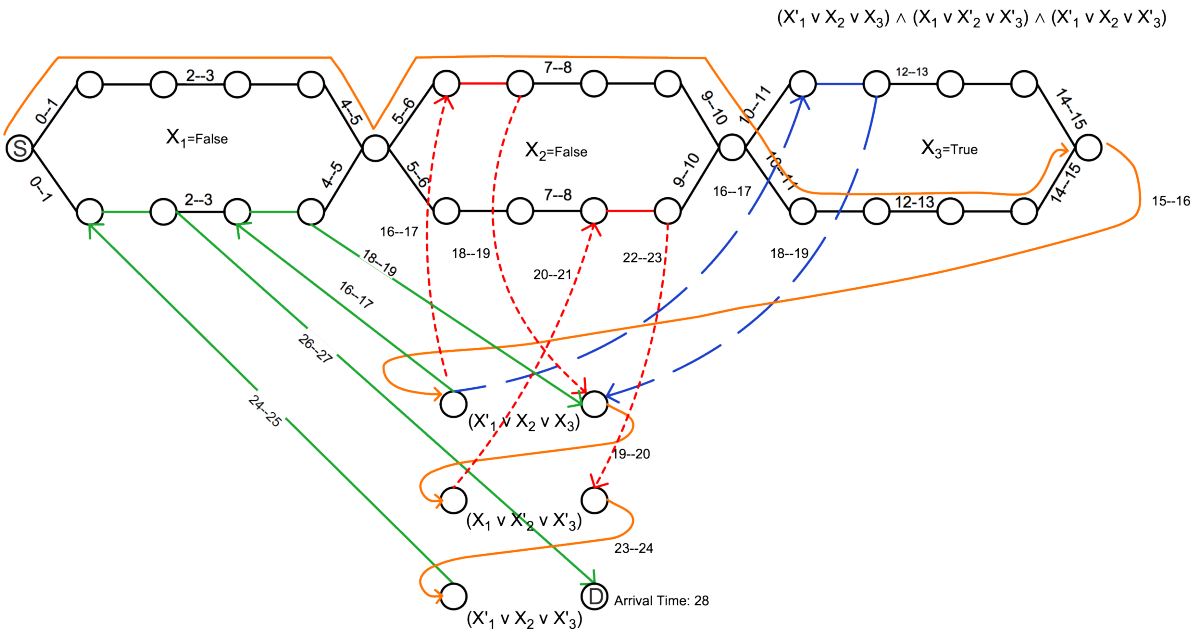


Figure 8.5. Example of the reduction from At-Most-3SAT to ES_1^1.

a *literal edge* to the $exit$ node of the clause gadget. For $1 \leq j \leq m-1$, the $exit$ node of the clause gadget for $\mathcal{C}_j$ is connected to the $entry$ node of the clause gadget for $\mathcal{C}_{j+1}$ by an edge. The $exit$ node of the mth clause gadget is set to be the destination D.

The idea of the construction is that the path from S to the $exit$ node of the nth variable gadget corresponds to a truth assignment for the n variables. A path following the upper edges of a variable gadget sets the variable to *false*, a path following the lower edges sets it to *true*. A clause gadget admits a feasible path from its $entry$ node to its $exit$ node through a *literal edge* only if the truth assignment makes that literal true.

We illustrate the above graph construction with the help of an example. Given the

Boolean formula $(\neg X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee X_2 \vee \neg X_3)$ with $m = 3$ clauses and $n = 3$ variables, we produce the graph shown in Figure 8.5. The graph consists of a gadget for each $variable$ and a gadget for each $clause$. For the ease of understanding of the figure we keep it less denser by omitting several edges of the second and third clause gadget. The labels of the edges give their availability for data transfer and the edges without label are feasible at all times.

Denote by $\mathcal{I}'$ the constructed instance of ES_1^1. Recall that $\mathcal{I}$ denotes the given instance of At-Most-3SAT. We claim that $\mathcal{I}$ is satisfiable if and only if $\mathcal{I}'$ admits a feasible path with arrival time at most $5n + 4m + 1$. The proof in both directions is as follows.

(a) $\mathcal{I}$ is satisfiable $\Rightarrow$ $\mathcal{I}'$ has a feasible path with arrival time at most $5n + 4m + 1$.

Consider a truth assignment τ that satisfies the instance $\mathcal{I}$ of At-Most-3SAT. Use the truth assignment τ to construct a path $\mathcal{P}$ from S to D as follows. First, the path goes from S to the $exit$ node of the X_n gadget by choosing in each variable gadget the upper edges if the variable is *false* in τ and the lower edges if the variable is *true* in τ. Denote this initial part of $\mathcal{P}$ by $\mathcal{P}'$. For each clause C_j, let l_j be a literal of C_j that is true in τ. Continue the path from the $exit$ node of the X_n gadget using the following edges for each clause C_j, $1 \leq j \leq m$: Use the incoming edge of the $entry$ node of the C_j gadget, then the edge to the start node of the *literal edge* chosen for l_j during the construction, then that *literal edge* (which is feasible as it is not used by $\mathcal{P}'$ nor by another clause), then the edge from the end node of that *literal edge* to the $exit$ node of the C_j gadget. The resulting path $\mathcal{P}$ consists of $5n + 4m$ edges. Each edge has delay 1, and it is easy to verify that by construction the ith edge of the path is feasible at time i. Hence, $\mathcal{P}$ is a feasible path with arrival time $5n + 4m + 1$.

(b) $\mathcal{I}'$ has a feasible path with arrival time at most $5n + 4m + 1$ $\Rightarrow$ $\mathcal{I}$ is satisfiable.

Let P be a path from S to D with arrival time at most $5n + 4m + 1$. The way in which the availability periods of the edges have been set, $\mathcal{P}$ must first go from S to the $exit$ node of the X_n gadget by traversing the upper or lower path of each

of the n *variable* gadgets. Let $\mathcal{P}'$ denote this initial part of $\mathcal{P}$. After that, $\mathcal{P}$ must pass through all the *clause* gadgets, and in each *clause* gadget it must use a *literal edge* that is not used by $\mathcal{P}'$. This means that the truth assignment corresponding to $\mathcal{P}'$ must satisfy at least one literal in each clause, meaning that $\mathcal{I}$ is satisfiable.

Hence the above reduction shows that ES_1^1 is an NP-hard problem. □

To further illustrate that a satisfying truth assignment gives a feasible path from S to D, we again refer to the Figure 8.5. We consider the following satisfying truth assignment: $X_1 =$ *true*, $X_2 =$ *false* and $X_3 =$ *true* for the boolean formula mentioned above and shown in the Figure 8.5. The shown path from S to the $exit$ node of the X_3 gadget corresponds to the truth assignment. The $exit$ node of the X_3 gadget and the $entry$ node of the first clause are directly connected. For the clause $(\neg X_1 \vee X_2 \vee X_3)$, the $entry$ and the $exit$ nodes are connected through three paths: For literal $\neg X_1$ one of the two lower *literal edges* of the X_1 gadget is used, for the positive literal X_2 one of the two upper *literal edges* of the X_2 gadget is used, and for $\neg X_3$ one of the two lower *literal edges* of the X_3 gadget is used.

The $exit$ node of the first clause gadget is reachable only through two feasible paths, one path for the literal $\neg X_1$ and the other one for the literal X_3. For literal X_2, the path via the corresponding *literal edge* is not feasible since all the upper edges of the X_2 gadget have already been used by the path from S to the $exit$ node of the X_3 gadget (and only edge-simple paths are allowed).In the same manner, feasible paths through the second and third clause gadget can be found: Each of these clauses has a satisfied literal, and the corresponding literal edge can be used for the path (we have shown one of the feasible paths for the remaining $clause$ gadgets). When the data is transferred from S and reaches the $exit$ node of the last $variable$ gadget, its outgoing edge becomes available at the appropriate time for appropriate duration, i.e., from 15 to 16. In the same manner the path goes through the clause gadgets by following feasible edges with appropriate timing to D. Complete data arrives at time 28 at D after going through $5n + 4m = 27$ edges, where 5 is the number of edges in a $variable$ gadget that must

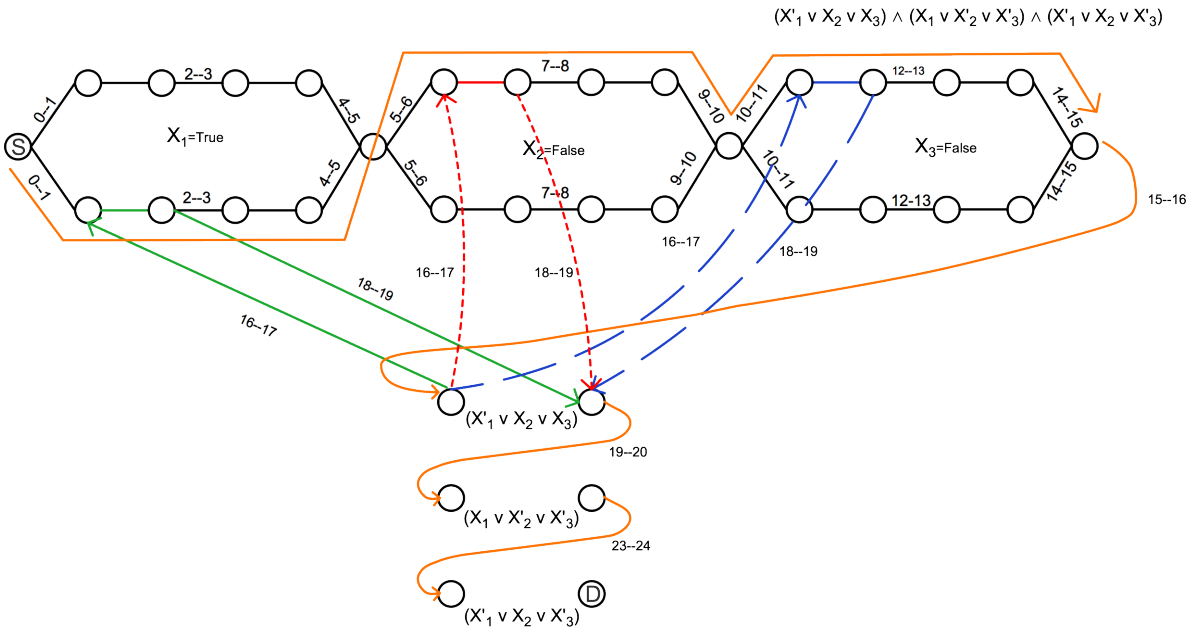


Figure 8.6. An unsatisfying truth assignment does not give a feasible path from node S to node D.

be followed by a path, whereas 4 is the number of edges related to a $clause$ gadget that must be used by a path. Thus, a satisfying truth assignment gives a feasible path to the destination node D.

Now we give an example, with the help of Figure 8.6, to show that an unsatisfying truth assignment does not lead to a feasible path from S to D. We consider the following unsatisfying truth assignment: $X_1 = \textit{true}$, $X_2 = \textit{false}$ and $X_3 = \textit{false}$ for the same boolean formula discussed above. The path from S to the $exit$ node of the X_3 gadget corresponds to the considered truth assignment. Three paths, corresponding to the literals in the $clause$, from the $entry$ node to the $exit$ node of the first $clause$ gadget are drawn by following the same technique mentioned above (the paths for other $clause$ gadgets are not shown in the diagram). Here none of the paths for first $clause$ gadget is feasible, since all the needed $literal$ edges have already been used by the path from S to the $exit$ node of the X_3 gadget. Thus, the $exit$ node of the first $clause$ gadget becomes unreachable from its $entry$ node, and therefore D becomes unreachable. Thus, an unsatisfying truth assignment does not give a feasible path to the destination node.

Definition 8.1.9. (ES^{arb}_{arb} problem)

Given the problem *ARRP*, we restrict the path to be an edge-simple path from S to D.

It is assumed that $\lambda \geq 1$ and $b \geq 1$ are arbitrary.

Corollary 8.1.10. *ES^{arb}_{arb} is NP-Hard.*

Proof. ES^1_1 problem is a special case of the problem ES^{arb}_{arb}. Theorem 8.1.8 proves that the ES^1_1 problem is NP-hard. Hence the ES^{arb}_{arb} problem is also NP-hard. □

Definition 8.1.11. (NS^1_1 problem)

Given the problem *ARRP*, we allow the path from S to D to be a non-simple path, $\lambda = 1$ and $b = 1$.

Theorem 8.1.12. *NS^1_1 is polynomial-time solvable.*

Proof. This problem is related to a network flow problem discussed in [32], in which the goal is to maximise the amount of flow from a source node that can reach a destination node in the network within a given time limit, where each link has a traversal time and a capacity that limits the rate at which flow can enter the link. The problem NS^1_1 also asks for data to be transmitted from a source to a destination within minimum time, but the capacities of the edges are time-dependent and the data must be transmitted along a single path and in the form of a burst with fixed rate *rbw* and duration b.

The NS^1_1 problem can be solved using the Time-Expanded Network (TEN) construction as done in [32]. A TEN is a network that contains copies of the original nodes of the network for each discrete time step and edges connecting the nodes in different time layers. The TEN for a given graph $G = (V, E)$ of an instance of the problem NS^1_1 is denoted by $G(\tilde{T})$ and can be constructed as follows. $G(\tilde{T})$ contains $T + 1$ copies of V, denoted by $V_{\tilde{T}}$. The copy of node $v \in V$ for time $\theta \in \{0, 1, \ldots, T\}$ is denoted by v_θ. If nodes $u, v \in V$ are connected by an edge $e(u, v) \in E$, edges $e(u_\theta, v_{\theta'})$ are added to $G(\tilde{T})$, where $\theta' = \theta + d_{e(u,v)}$, for all times θ at which edge $e(u, v)$ is feasible. The path with earliest arrival time from S to D can be found in polynomial time by finding the copy D_θ for minimum θ that is reachable from a copy of S. To allow a more efficient calculation of which copies of D are reachable from some copy of S, an additional node

S' is added to $G(\tilde{\mathcal{T}})$ and connected to all copies of S. It then suffices to determine the nodes that are reachable from S' using, for example, a single depth-first search. Among all copies D_θ that are reachable from S', let D_θ be the one with minimum index θ. The path from S' to D_θ then corresponds to a minimum arrival path.

The construction of the TEN for solving the problem NS_1^1 can be illustrated with the help of Figures 8.7 and 8.8. A sample instance of NS_1^1 is shown in Figure 8.7, with edge labels representing availability periods. The time horizon is $T = 13$. The TEN for the graph given in Figure 8.7 is shown in Figure 8.8. It contains $T + 1 = 14$ copies of all nodes. The nodes are connected based on the availability of the links between them at different times in the original graph. The paths from copies of S to copies of D correspond to feasible paths along which data can be transmitted from S to D. There are two paths through which D is reachable from S in this way, one is S_6-a_7-b_8-c_9-a_{10}-b_{11}-D_{12} and the other is S_{10}-a_{11}-b_{12}-D_{13}. The path reaching D_{12} is an earliest arrival time path.

As mentioned earlier, S' is added to the TEN in order to improve the efficiency of finding the path that starts at some copy of S and reaches a copy D_θ of D with minimum index θ. This can be explained further as follows. In the original graph G, let the total number of nodes and edges be n and m, respectively. The graph $G(\tilde{\mathcal{T}})$ has $(T + 1) \cdot n$ nodes (without S') and $T \cdot m$ edges (without the $T+1$ edges from S' to copies of S). A naive algorithm that checks for each pair of a copy S_θ of S and a copy $D_{\theta'}$ of D whether $D_{\theta'}$ is reachable from S_θ in $G(\tilde{\mathcal{T}})$ has running time $\mathcal{O}(T \cdot (n + m))$ for each pair and thus $\mathcal{O}(T^2 \cdot T \cdot (n + m)) = \mathcal{O}(T^3 \cdot (n + m))$ in total. By connecting S' to all copies of S and running a single depth-first search from S', we can solve the problem in time $\mathcal{O}(T \cdot (n + m))$. As the input has size $\mathcal{O}(n + m \cdot T)$, the running-time is polynomial in the size of the input. □

Definition 8.1.13. (NS_1^{arb} problem)

Given the problem *ARRP*, we allow the path from S to D to be a non-simple path with $\lambda = 1$ and $b \geq 1$.

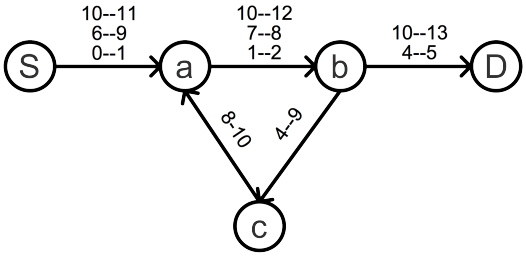


Figure 8.7. Example instance of NS_1^1.

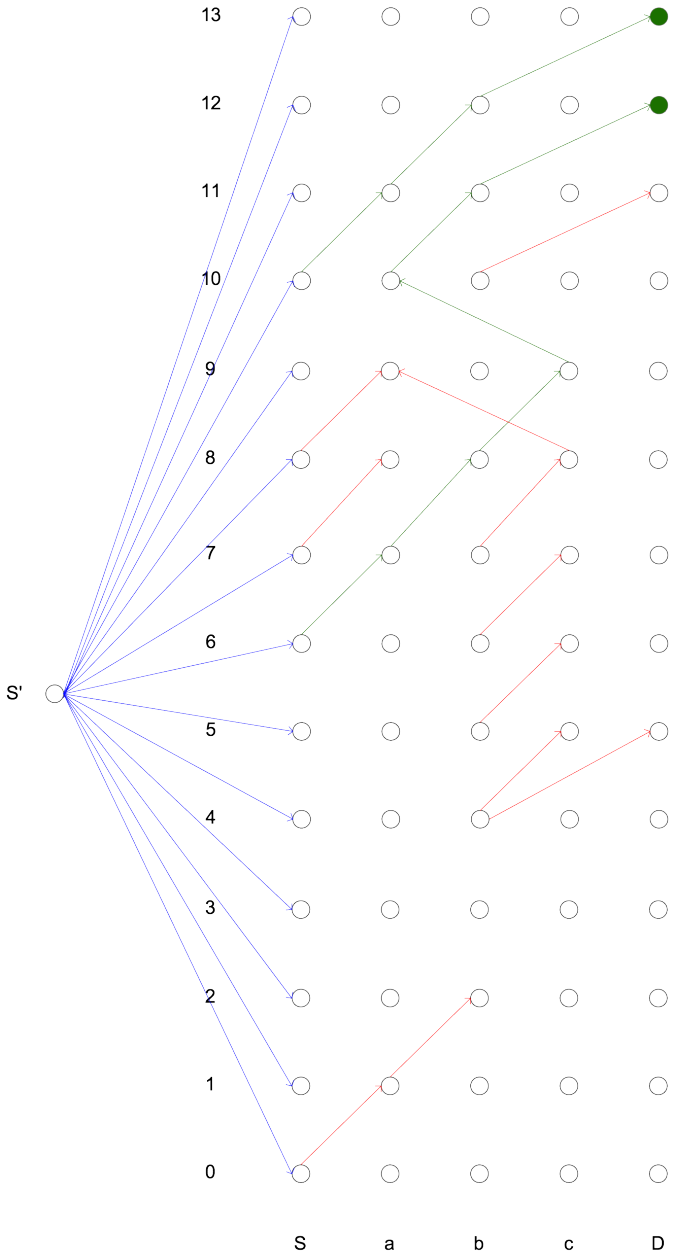


Figure 8.8. Time Expanded Network for the Instance of Figure 8.7.

Theorem 8.1.14. *NS_1^{arb} is NP-Hard.*

Proof. To prove that NS_1^{arb} is NP-hard, we use a reduction from the $\mathrm{At\text{-}Most\text{-}3SAT}$ problem similar to the proof of Theorem 8.1.8. For a given instance $\mathcal{I}$ of $\mathrm{At\text{-}Most\text{-}3SAT}$ with m clauses and n variables, we construct a graph as follows. For each $variable$ $x_i, 1 \leq i \leq n$, we create a gadget of 10 connected nodes, consisting of two parallel paths of 4 nodes connected to an $entry$ node and an $exit$ node at each end. We call these paths *upper* and *lower* paths. For the sake of simplicity, we only mention the availability periods of edges. If an edge is feasible at a certain time, this means that the available capacity of the edge is set equal to the requested bandwidth *rbw*. Each upper path and each lower path contains again two *literal edges*, which are made feasible to be used in correspondence to a literal in a clause. The duration of the data transfer is set long enough to ensure that each literal edge can be used only once in the time horizon of interest. The *literal edges* in the upper path can be used for positive literals in clauses, while the *literal edges* in the lower path can be used for negative literals. The gadget for the variable x_i is connected to the gadget of variable x_{i+1} by identifying the $exit$ node of the x_i gadget with the $entry$ node of the x_{i+1} gadget, for $1 \leq i \leq n-1$. The $entry$ node of the first variable gadget is defined to be S. The $exit$ node of the nth variable gadget is connected to the $entry$ node of the first $clause$ gadget by an edge. For each clause $\mathcal{C}_j$, $1 \leq j \leq m$, we again create a gadget with two nodes, the $entry$ node and $exit$ node of the $clause$ gadget. The only feasible paths from the $entry$ node of the $clause$ gadget for $\mathcal{C}_j$ to the $exit$ node of the same clause gadget go via a *literal edge* of the $upper$ or $lower$ path of the corresponding gadget of the variable in the clause. The path starts from the $entry$ node of the clause, goes through the source node and the target node of a *literal edge*, and ends at the $exit$ node of the clause gadget.

The $exit$ node of the clause gadget for $\mathcal{C}_j$ is connected to the $entry$ node of the clause gadget for $\mathcal{C}_{j+1}$ by an edge, for $1 \leq j \leq m-1$. The $exit$ node of the mth clause gadget is defined to be D. As mentioned earlier, the duration of the data transfer needs to be set in such a way that it will cause a conflict if a *literal edge* is used more than

once in the path from S to D. By conflict we mean that the data enters the edge for the second time while the edge is still being used for the earlier data transmission (i.e., before the last bit of the data has entered the edge for the first time), which means that the capacity of the edge is exceeded. A duration of the data transfer that ensures that a conflict is created if an edge is used twice in our construction is called *conflict duration*, denoted by cd. We can choose $cd = 5 \cdot n + 4 \cdot m$, where 5 is the number of edges per side of a $variable$ gadget and 4 is the number of edges that a path uses when it traverses a $clause$ gadget (which include the incoming edge to the $entry$ node of the clause gadget and three edges for a path from the $entry$ node to the $exit$ node of the same $clause$ gadget via a *literal edge*). Thus cd is the total number of edges in a path from S to D that first passes through all the variable gadgets and then all the clause gadgets. We set the duration of the data transfer to $b = cd$.

To restrict a $clause$ path to use only one *literal edge* of a $variable$ gadget for the corresponding variable, the availabilities of the edges are set as follows. For the x_i gadget, set the interval of availability of the edges to be from $5 \cdot (i-1) + c$ to $5 \cdot (i-1) + c + b$, where $c \in \{0, 2, 4\}$ is the index of the edges, defined as in the proof of Theorem 8.1.8. The availability of an edge related to the clause $\mathcal{C}_j$ is set to be the interval from $5 \cdot n + 4 \cdot (j-1) + c$ to $5 \cdot n + 4 \cdot (j-1) + c + b$, where $c \in \{0, 1, 3\}$ is the index of the edge, again defined as in the proof of Theorem 8.1.8.

We illustrate the above graph construction with the help of an example. Given the Boolean formula $(\neg X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_2 \vee \neg X_3) \wedge (\neg X_1 \vee X_2 \vee \neg X_3)$ with $m = 3$ clauses and $n = 3$ variables, we produce the graph shown in Figure 8.9. The graph consists of a gadget for each *variable* and a gadget for each *clause*. The labels of edges give their availability for data transfer. The value of cd for this graph is 27, but for the simplicity of representation we assume the duration of the data transfer is $b = 100$ (any value of b that is at least 27 would work). All the *literal edges* are made feasible at all times.

The shown path from S to the last node of the nth variable gadget corresponds to a truth assignment that sets X_1 and X_2 to *false* and X_3 to *true*. The $exit$ node of the

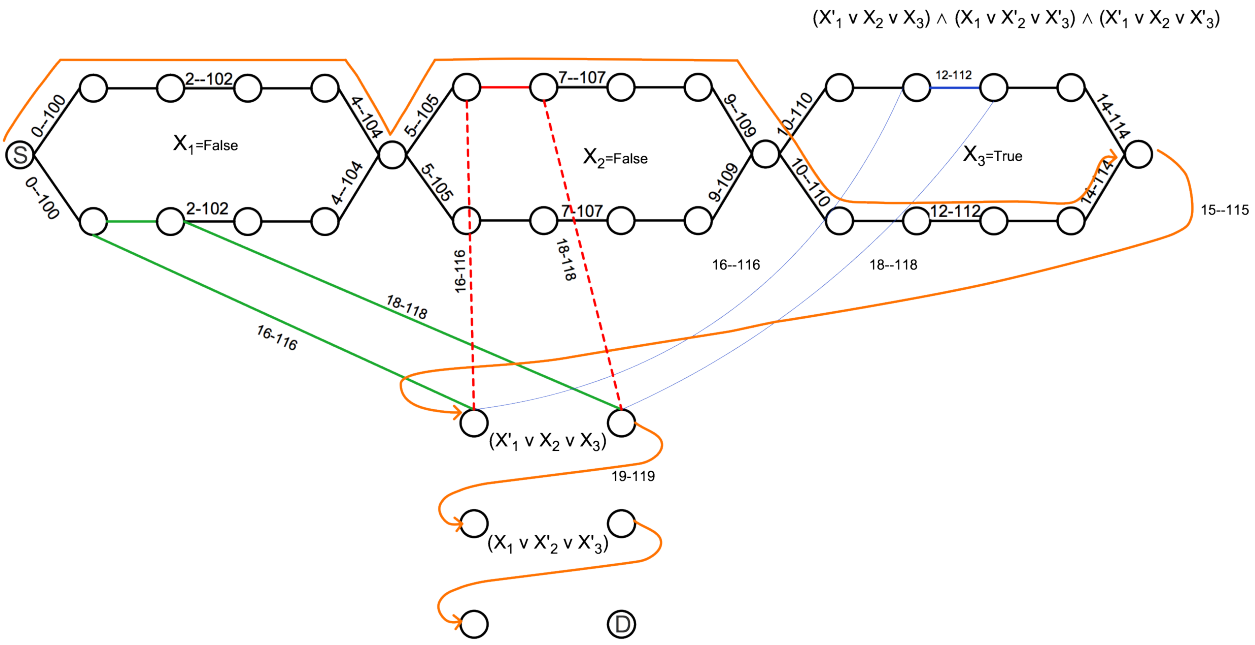


Figure 8.9. Graph construction for an example instance of the At-Most-3SAT problem.

$clause$ gadget for $(\neg X_1 \vee X_2 \vee X_3)$ is reachable through only two paths, one via a lower *literal edge* of the X_1 gadget for the literal $\neg X_1$ and the other one via an upper *literal edge* of the X_3 gadget for the literal X_3. For literal X_2, the corresponding path would cause a conflict on the literal edge since all the upper edges of the X_2 gadget are still under use for the first data transmission at time 17. In the same manner, the edges for the second and the third clause can be constructed (not shown in the figure).

Recall that $\mathcal{I}$ is the given instance of At-Most-3SAT, and let $\mathcal{I}'$ denote the constructed instance of NS_1^{arb}. We claim that $\mathcal{I}$ is satisfiable if and only if $\mathcal{I}'$ has a feasible path from S to D with arrival time at most $5n + 4m + b$. The proof in both directions is as follows.

(a) $\mathcal{I}$ is satisfiable $\Rightarrow$ $\mathcal{I}'$ has a feasible path from S to D with arrival time at most $5n + 4m + b$.

Let τ be a truth assignment that makes $\mathcal{I}$ true. It can be used to construct a path from S to D as follows. First, the path goes from S to the $exit$ node of the X_n gadget by following in each *variable* gadget either the upper or the lower path, depending on whether τ sets the corresponding variable to false or true, respectively. Denote this initial part of the path by $\mathcal{P}'$. For each clause C_j, let

l_j be a literal of C_j that is made true by τ. Complete the path $\mathcal{P}'$ by traversing all the $clause$ gadgets, choosing in the C_j gadget the path via the *literal edge* corresponding to l_j. This gives a path from S to D that consists of $5n + 4m$ edges and thus has arrival time $5n + 4m + b$. Furthermore, the path does not use any edge twice (ensuring that there is no conflict), and each edge of the path is feasible at the time period when data is transmitted through it. Hence, $\mathcal{I}'$ has a feasible path from S to D with arrival time $5n + 4m + b$.

(b) $\mathcal{I}'$ has a feasible path from S to D with arrival time at most $5n + 4m + b \Rightarrow \mathcal{I}$ is satisfiable.

Let $\mathcal{P}$ be a path from S to D in $\mathcal{I}'$ with arrival time at most $5n + 4m + b$. As the duration of the data transfer is b and all edges have delay 1, the path must consist of at most $5n + 4m$ edges. As the duration of the data transfer is not smaller than $cd = 5n + 4m$ and no edge has more than *rbw* available capacity at any time, $\mathcal{P}$ does not use any edge twice. Furthermore, by the choice of the availability periods of the edges, the path $\mathcal{P}$ must traverse first all the n *variable* gadgets and then all the m *clause* gadgets and thus consist of exactly $5n + 4m$ edges.

Let $\mathcal{P}'$ be the initial part of $\mathcal{P}$ from S to the $exit$ node of the nth *variable* gadget. Let τ be the truth assignment corresponding to $\mathcal{P}'$. As $\mathcal{P}$ must traverse each *clause* gadget via a *literal edge* corresponding to a literal of the clause that is made true by τ, we have that τ satisfies $\mathcal{I}$.

Hence, the above reduction show that NS_1^{arb} is an NP-hard problem. □

ARRP with $\lambda = 0$

In some scenarios, the delay of the links in the network may be negligible compared to the units of time for which advance reservations are recorded in the utilisation profiles. In such scenarios, the link delay can effectively be assumed to be zero. Finding an earliest arrival path is possible in polynomial time in this case. Using our notation, we consider

	delay = 0		*delay ≥ 1*	
	duration = 1	*arbitrary duration*	*duration = 1*	*arbitrary duration*
Simple Path	polynomial	polynomial	NP-Hard	NP-Hard
Edge-Simple Path	polynomial	polynomial	NP-Hard	NP-Hard
Non-Simple Path	polynomial	polynomial	polynomial	NP-Hard

Table 8.1. Complexities of variants of routing problem in advance reservation environment.

ARRP with $\lambda = 0$. The problem is solvable in polynomial time as follows (see [41] for a similar algorithm): To find a path from S to D when $b = 1$, for each $\theta \in [0, T-1]$ remove all the links from graph G which are not feasible at time θ. Check whether a path from node S to D exists using only the remaining links. Repeat the process until a path is found or the time limit T is reached. For the case when $b \geq 1$, the same approach can be followed, except that links are only kept in the graph if they are feasible at all times from time θ to $\theta + b - 1$. Furthermore, we note that in the case $\lambda = 0$ a simple path with arrival time t exists if and only if an edge-simple or non-simple path with arrival time t exists.

Table 8.1 summarises the complexity results for the variants of *ARRP* considered in this section.

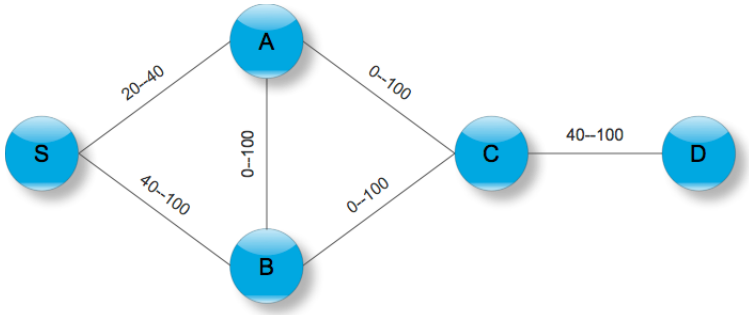
8.2 Shortcomings of Previous Routing Methods

Finding a simple path that gives the earliest arrival time for a data transfer from a source to destination in an OBS network with non-zero link delays that supports AR is NP-hard, as shown in the previous section. For this reason, different polynomial-time heuristics have been proposed by Varvarigos et al. [70], and a Dijkstra variant was proposed by us (see Section 4.1). These heuristics do not guarantee to produce an optimal path. We noticed that there are scenarios, even when the delay of all the links is zero, in which these heuristics can be misled while there is a way of routing the data on a much better path.

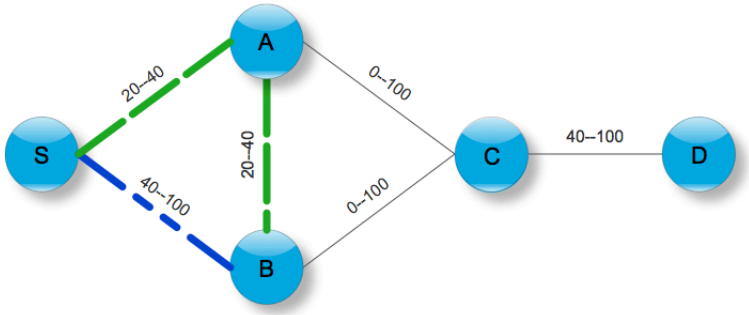
To explain the scenarios the problem S_0^{arb} is defined as follows. Given the problem *ARRP*, we restrict the path to be a simple path from the source to the the destination, $\lambda = 0$

and $b \geq 1$. Consider the problem S_0^{arb} and the graph G with $V = \{S, A, B, C, D\}$ shown in Figure 8.10. The labels of the links show the range of future time slots at which *rbw* bandwidth is available. It is assumed that the duration is $b = 20$ if bandwidth *rbw* is used for the data transfer. S is the source node and D is the destination node. We show that the variant of Dijkstra's algorithm not only fails to find a simple path with earliest arrival time but also fails to find any existing path. Due to the *earliest arrival time* criterion used by our variant of Dijkstra's algorithm, the path to a node which is feasible for transmission at earlier time will be selected and the path with later transmission time will be dropped. By following the *Associative Operation* $\oplus$ for path calculation, it can be seen that the arrival time at the node B is 40 for the path S-A-B. The path S-B is dropped because of its later arrival time (i.e., 60). For the node C the path S-A-B-C is better but it cannot be continued to node D because of the unavailability of link C-D at the time when the path S-A-B-C is feasible. The decision of dropping the path S-B caused the path S-B-C to be missed, which would have led to D. Thus, in AR scenarios the simple variant of Dijkstra's algorithm may miss an optimal path or even non-optimal existing path(s). For this reason, routing algorithms which unlike Dijkstra's approach maintain a set of multiple paths to each node were proposed in [70].

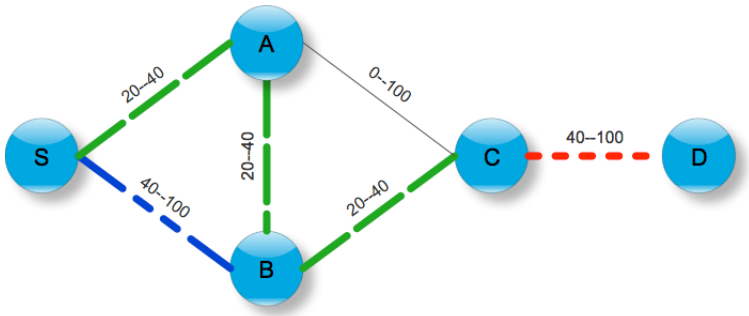
In Section 4.1, we discussed the *AW* and the *CS* heuristics proposed in [70]. Here we illustrated the potential inefficiency exhibited by these heuristics in certain situations using the example graph given in Figure 8.11. Consider the S_{arb}^{arb} problem and a graph G with $V = \{S, A, D\}$. S is the source node and D is the destination node. The delays of links S-D, S-A and A-D are 3, 1 and 1, respectively, and it is assumed that $b = 20$. First, *AW* and *CS* compute a set of non-pseudo-dominated paths. S-A-D and S-D are the two paths from S to D. According to the pseudo-domination checks of both *AW* and *CS*, the path S-A-D dominates the path S-D because its delay is smaller, it is feasible during more time slots and it has more fragments of 20 consecutive feasible slots. Thus, the path S-D is discarded. However, the path S-D has an earlier arrival time. Hence, a better path is missed. To address the problems shown in Figures 8.10 and 8.11, we will propose a *K-th shortest path* variant in the next section.



(a) Link labels show the range of time slots at which the requested bandwidth is available.



(b) Two paths are found from node S to node B, S-A-B and S-B. Dijkstra's algorithm drops the path S-B because of its later arrival time.



(c) Due to the decision made in the previous figure, the path S-A-B-C is adopted, which fails to reach node D because of mist-iming.

Figure 8.10. The variant of Dijkstra's algorithm fails to find an existing path for earliest arrival time of data from node S to node D.

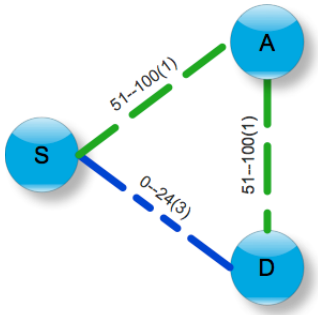


Figure 8.11. Two paths from node S to node D are found. Link labels show the time slots when the requested bandwidth is available and the delay, in parentheses. According to *CS* and *AW* the path S-A-D pseudo-dominates the path S-D.

8.3 K-th Shortest Path Variant for AR Routing Problem

The K-th Shortest Path (*KSP*) problem was initially introduced by Hoffman and Pavely in 1959 [42]. Afterwards, several algorithms were proposed for this problem. In the *KSP* technique, a set $\{p_1, p_2, \cdots, p_K\}$ of paths from source to target node are determined for an objective function c in such a way that

$$c(p_k) \leq c(p), \text{ for any } p \in P - P(k-1) \text{ for all } k \in \{2, 3, \cdots, K\},$$

where P is the set of all simple paths from source to target node and

$P(k) = \{p_1, p_2, \cdots, p_k\}$ is the set of k shortest paths, for $k \in \{1, 2, \cdots, K\}$.

So along with finding the shortest path, a second, third, ..., Kth shortest path are also found. A typical application where *KSP* is used is to find multiple shortest paths in a network, so that in case a problem arises on the shortest path, the next shortest path can be used.

Computing the K shortest paths with respect to edge delays and then choosing the path with minimum arrival time among the K paths seems a promising approach to the routing problems we are considering. If the *KSP* technique is applied in the scenario shown in Figure 8.10, it may find the path S-B-C-D as the second shortest path. It will also work in scenarios like the one shown in Figure 8.11.

One of the issues with the *KSP* technique in the AR scenario is the choice of the value of K. In larger networks it would be difficult to decide what value of K is necessary or sufficient to yield the optimal path. The *KSP* approach may miss a better path in larger networks similarly as Dijkstra's variant can miss it already among few feasible paths (Figure 8.10). The other issue is the fine tuning of the *KSP* algorithm for the AR scenario. Even though different algorithms with different running times are available for the standard *KSP*, in the AR environment where a vector data structure computation is involved, *KSP* algorithms may get slow. Our work is not intended to address these issues of the *KSP* algorithm, but to show that in some cases it can indeed find a better path than

the *CS* and *AW* heuristics. In our experiments we, used a modified *KSP* code written by Guillaume Boulmier [22], which is a variant of Bellman-Ford algorithm [17, 33, 54]. The original algorithm for *KSP* implemented by Guillaume Boulmier and our modification are described in the following.

Notation

A directed graph G has a set of vertices V and a set of edges E. The weight of each edge in E is called its *delay*. The weight w of a path is the sum of the delays of the edges in the path. The objective is to find the K shortest paths on the basis of w from a source vertex S to a destination vertex D. K paths to each vertex $v \in V$ are stored in $KPathList_v$ in ascending order of w. $curr\_iV$ is a set of currently improved vertices, to which a better path has been found in the current iteration of the outermost loop in the procedure *FindKShortestPaths*, and $prev\_iV$ is a set of previously improved vertices to which a better path was found in the previous iteration of the outermost loop.

Pseudo-code

Description

To find a shortest path in a graph, the standard Bellman-Ford algorithm uses two nested loops: The number of iterations of the outer loop is equal to the number of vertices, and in each iteration of the outer loop, the inner loop processes all edges. In each iteration of the inner loop, the distance and predecessor node of the vertex reached by the edge are updated if a shorter path is found. The complexity of the Bellman-Ford algorithm is $\mathcal{O}(|V| \cdot |E|)$. In the pseudo-code above, this approach has been adapted to the *KSP* as follows. Unlike the Bellman-Ford algorithm, the pseudo-code above does not go through all edges at each iteration of the outermost loop. Instead, it keeps a set of vertices for which a better path was found in the previous iteration of the outer loop. It only goes through the edges of these previously improved vertices and may insert the

Algorithm 8.1 Guillaume Boulmier Implementation of KSP Algorithm

```
function FindKShortestPaths(G, S, D)
    Initialization(S, prev_iV)
    for passNumber = 0 to V.size() do
        for each piv in prev_iV do
            updateOutgoingEdges(piv)
        end for
        prev_iV ← curr_iV
    end for
end function
function Initialization(S, prev_iV)
    prev_iV ← prev_iV ∪ S
    add empty path to KPathList_S
end function
function updateOutgoingEdges(piv)
    for each v in V neighbour of piv do
        if newPathAdded(cv, piv, e) then
            curr_iV ← curr_iV ∪ cv
        end if
    end for
end function
function newPathAdded(cv, piv, e)
    pathAdded ← false
    for each p in KPathList_piv do
        np ← p ∪ e (np is the encountered path to cv)
        w_np ← weight of np
        for each q in KPathList_cv do
            w_q ← weight of q
            if w_np ≤ w_q then
                insert np in KPathList_cv
                pathAdded ← true
                if size of KPathList_cv > K then remove last path from
    KPathList_cv
                end if
            end if
            if (w_np > w_q) and (q is the last path in KPathList_cv) and (size of
    KPathList_cv < K) then
                insert np in KPathList_cv
                pathAdded ← true
            end if
        end for
    end for
    return(pathAdded)
end function
```

encountered paths into their neighbour's $KPathList$. For each vertex, a set of the up to K best paths found from the source to this vertex is maintained. The complexity of this algorithm is $\mathcal{O}(K \cdot |V| \cdot |E|)$

In our approach, we maintain the $KPathList$ on the basis of *EAT* instead of w. The *EAT* of a path is computed in the same manner as discussed in Section 3.4. The *EAT* of each path is computed only once, when it is first encountered, and stored in the $KPathList$ together with the path and its *CAV*. The time complexity of our approach is $\mathcal{O}(|V| \cdot |E| \cdot K \cdot T)$, where T is the size of the utilisation profile of a link (see Section 8.2).

One point is important to be mentioned regarding the difference between the path selection technique used by this *KSP* variant and by the heuristics due to Varvarigos et al. [70]: The *KSP* variant starts looking for a path on the basis of the objective criterion of earliest arrival time from the beginning, while *CS* and *AW* first build a set of non-pseudo-dominated paths and only at the end select the best path among them on the basis of the objective criterion. The *CS* and *AW* approach is flexible and can be used for different objective functions at the same time, but may not perform better in every case.

8.4 Evaluation

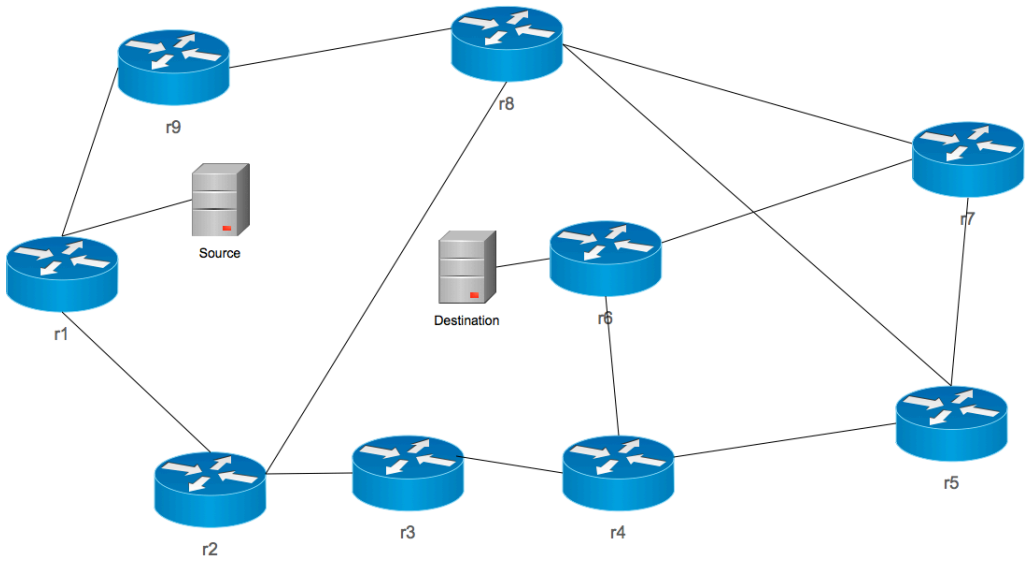
We evaluate six heuristics for the routing problem with advance reservation using the GridSim simulator [67]. Among them, three are different versions of *KSP*, namely $K5$, $K10$ and $K20$, for which the value of K has been chosen as 5, 10 and 20, respectively. The other three algorithms are the non-polynomial optimal method (*OM*), *CS* and *AW* proposed by Varvarigos et al. [70]. The main evaluation criterion is the arrival time of the computed path in an AR network for a data transmission from a single source node to a single destination node using the requested bandwidth. The tendency of a heuristic to fail to find an existing path in the network is also studied. The time complexities and CPU running time of the heuristics are compared as well. Experiments are conducted for four network topologies. Two network topologies (Figures 8.12b and 8.12c) were

used by Varvarigos et al. for their experiments. One network topology is a subset of the European DataGrid Testbed (Figure 8.12a) and one is a randomly generated network (Figure 8.12d). The link delay is kept the same for all network topologies. There is only one $source$ node and one $destination$ node, and they are shown in each network figure.

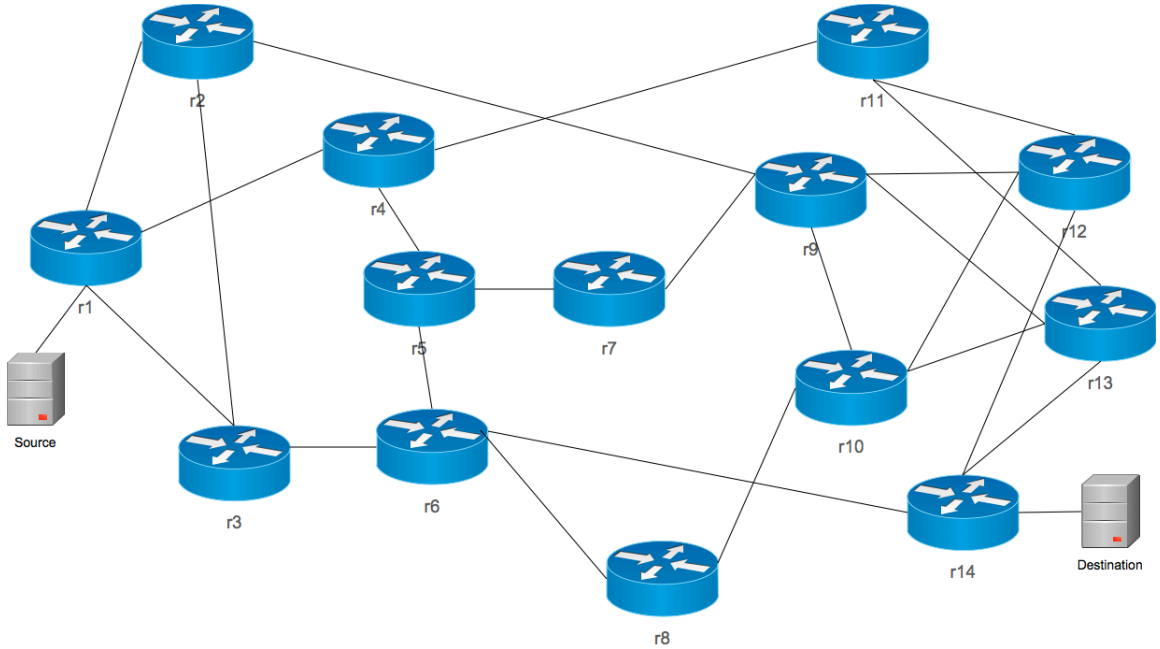
The simulation parameters and the procedure for setting them are as follows. We assume that each link has a *maximum bandwidth* of 1000 Megabits per second (Mbps). The parameter *minimum load* is specified as percentage of the *maximum bandwidth*, e.g., a minimum load of 80% would correspond to 800 Mbps in this case. To get the actual $load$ value, a random value between 0 and the difference of maximum bandwidth and minimum load is generated, which would be 20 in this case, and then subtracted from the *maximum bandwidth*. The reason for adopting this technique is to make the situation harder for path finding. While populating utilisation profile with the randomly generated $load$ values, each $load$ value is assigned to a *number of contiguous slots* (ncs). The choice of ncs has a significant effect on the availability of a path for the requested bandwidth, as explained below. The *size of the utilisation profile* $(uSize)$, within which we have to find a path, is also set. The other two parameters are the *data file size (dfs)* in Megabytes, which will be transferred from the *source* to the *destination* node, and the *requested bandwidth (rbw)*, against which the utilisation profile is checked.

The combination of the above parameters can greatly affect the outcome of an experiment. If the *rbw* is kept closer to the available bandwidth, say 6 Mbps for an 18 MB file, then the transfer requires 24 contiguous slots (assuming a slot duration of one second) on multiple links from the source to the destination. If the value of parameter ncs is small, say 1, then the probability of the occurrence of 24 contiguous slots in a utilisation profile having available bandwidth equal to or greater than 6 Mbps is low. The probability will be high if ncs is set to 25, however. So the four parameters $minimumload$, ncs, *rbw* and *dfs*, can be adjusted to make the path finding difficult or easier.

We experimented with 5 different combinations of these above parameters for each network. Due to different topologies, the number of links in a path from $source$ to

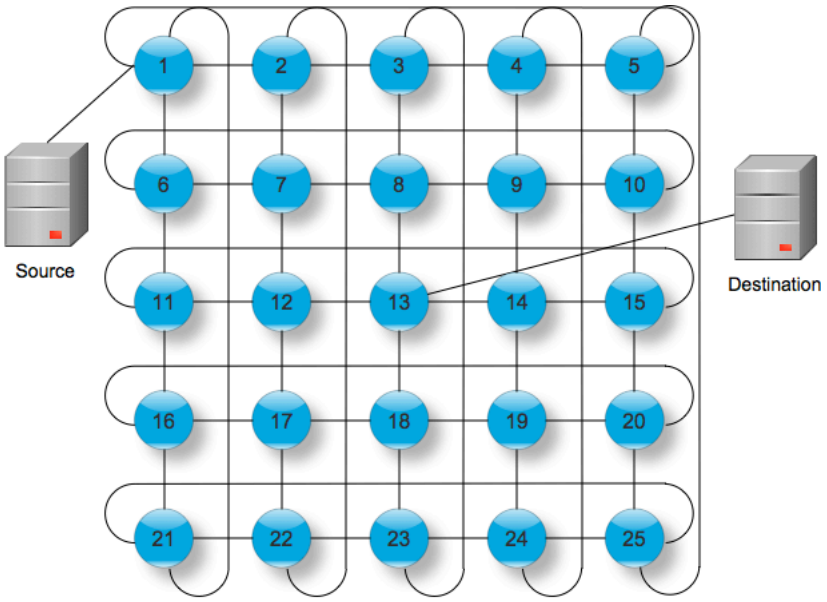


(a) The network topology of the subset of the European Data Grid test bed [16].

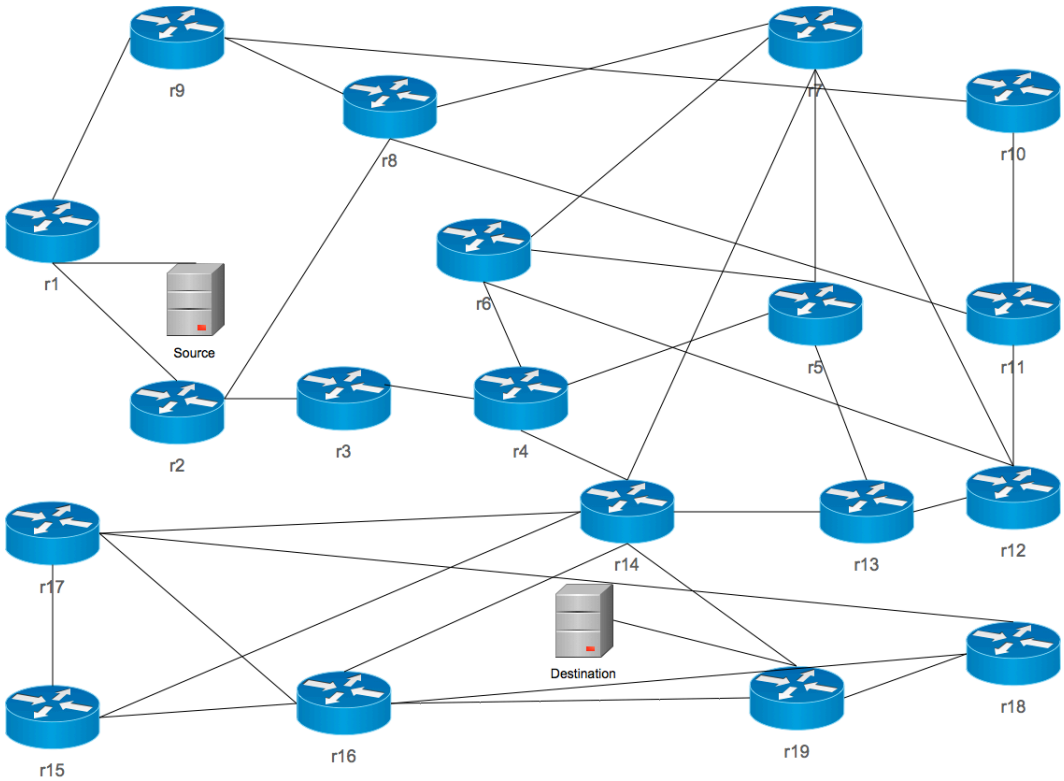


(b) The NSF network topology used in [70].

Figure 8.12. Four network topologies used for experiments. Source and destination nodes are also shown. (Figure continues on next page.)



(c) 5X5 mesh network topology used in [70].



(d) Random network topology.

Figure 8.12. (cont'd) Four network topologies used for experiments. Source and destination nodes are also shown.

	$min.\ load$	rbw	dfs	ncs	$duration$
$D1$	80-90%	30 Mbps	100 MB	5-10	27
$D2$	80%	30 Mbps	200 MB	5-10	54
$D3$	80%	30 Mbps	300 MB	5-15	80
$D4$	80%	30 Mbps	400 MB	10-15	107
$D5$	85%	40 Mbps	400 MB	10-15	80

Table 8.2. Parameter Settings for Experiments.

$destination$ varies between the topologies, and we slightly modified the parameters in the 5 combinations. We named these 5 combinations $D1$,$D2$,$D3$,$D4$ and $D5$ (see Table 8.2). For each parameter combination, on every network the simulation is run at least 100 times and the average of the *EAT* of only those simulation runs are shown in the results in which all the algorithms found a path.

Performance for earliest arrival time

First we discuss the results for the EDG network topology (Table 8.3a). For parameter settings $D1$, $D2$ and $D3$ the value of ncs is set to 5. For $D4$ and $D5$ the value of ncs is set to 10 to relax the situation for the algorithms. The results are variable but overall the *KSP* variants performed very well. For $D1$, $D2$ and $D5$ all the *KSP* variants gave optimal results, the same as *OM*, while gaining improvements up to approximately 13% over *CS* and *AW*. For the EDG network, even $K5$ seems efficient. The reason *CS* and *AW* could not perform well or optimally is the way parameter ncs is set. *KSP* variants remain successful because there are few possible paths from S to D, so the chance of missing a better path or even the optimal path is low. *CS* and *AW* seem to be misled due to the way the parameters are set.

In case of the NSF network topology, again the *KSP* variants performed better than *CS* and *AW*, especially for $D2$ where 30% improvement is obtained (Table 8.3b). Once again, the *KSP* variants either give optimal paths or are close to the optimal paths. Another point that is common with the EDG network topology result is that the performance of $K5$ is the same as the performance of $K20$. For $D5$, all the algorithms gave the same results. A possible explanation is that the harder setting of parameters forced all algorithms to choose from among very few paths. The paths with fewer edges have a

better chance of being feasible and are selected by all of the algorithms.

The 5X5 network topology with a large number of possible paths from the source to the destination increased the difference between the optimal path and the paths found by the heuristics (Table 8.3c). Still, the *KSP* variants performed better than *CS* and *AW* with up to 33% improvement, except for $D1$ and $D2$, where *CS* is marginally better. It is noticeable that with the increase in the possible number of paths, $K20$ performed better than $K5$ and $K10$, which shows that the network size may affect the suitable choice of K for finding a good solution.

In the Random network topology not only the number of possible paths are higher but many paths from the source to the destination node are longer as well (Table 8.3d). This make it difficult for heuristics, even for $K20$, which was closer to optimal for the other discussed networks, to stay close to the optimal method, unless the parameter settings are relaxed. The probability is higher that a longer path can change a good decision into a bad one in the end, as illustrated in Figure 8.10. Still, the *KSP* variants performed very well with $K20$ obtaining a maximum improvement of 60% over *AW* and 34% over *CS* for $D2$. Further improvements may be possible with higher values of K, but as discussed earlier it is not easy to tell for which network with what parameter settings which value of K would be the right choice. Experience may be helpful in selecting the right value of K.

The results are also shown as bar charts in Figures 8.13, 8.14, 8.15 and 8.16. Overall, the *KSP* variants have outperformed CS and AW. Among the two heuristics by Varvarigos et al. [70], *CS* performed better than *AW* in a number of cases. In the harder scenarios, among the *KSP* variants $K20$ performed better. In relaxed situations, all variants of *KSP* gave almost the same performance.

Path finding failures

It was mentioned earlier that in the AR environment, the heuristics under consideration may give an optimal or non-optimal path or miss existing paths entirely (i.e., fail to find

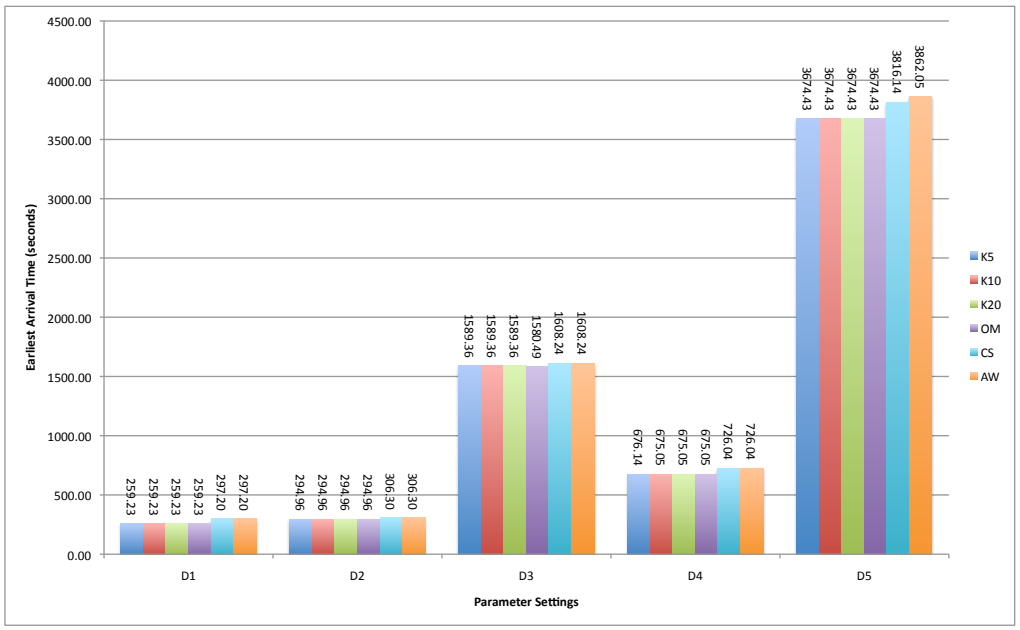


Figure 8.13. *EAT* found by all routing heuristics for each distribution on EDG network topology.

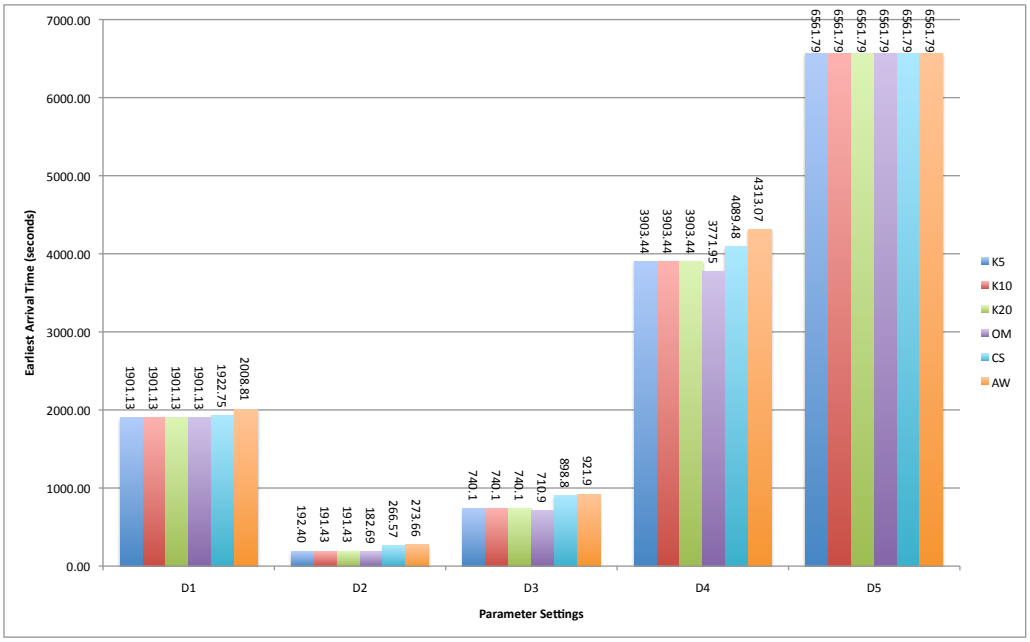


Figure 8.14. *EAT* found by all routing heuristics for each distribution on NSF network topology.

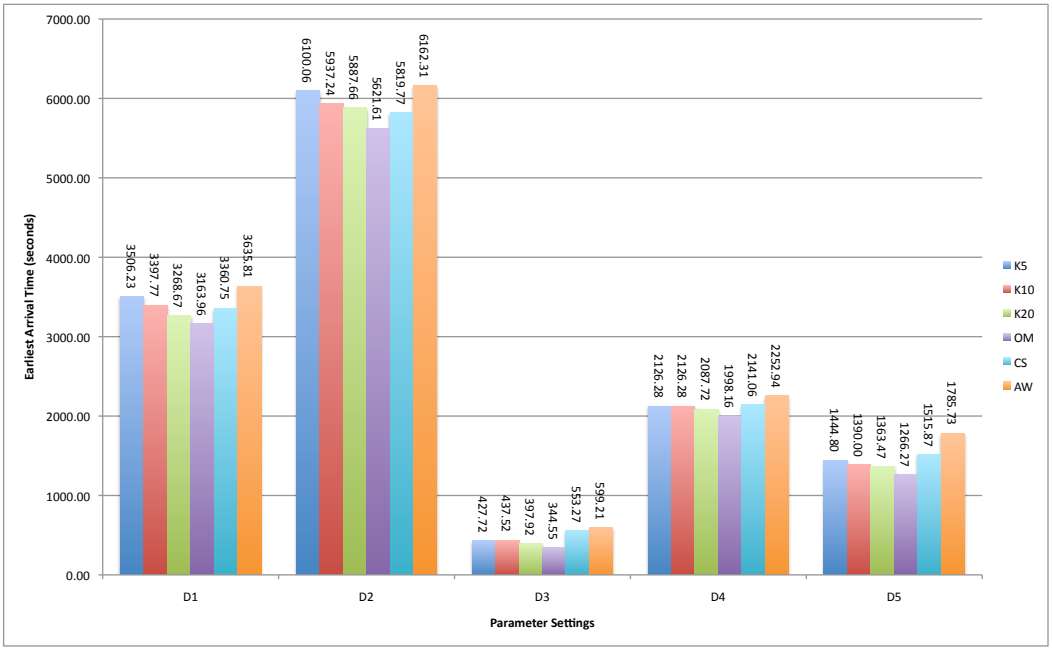


Figure 8.15. *EAT* found by all routing heuristics for each distribution on 5X5 network topology.

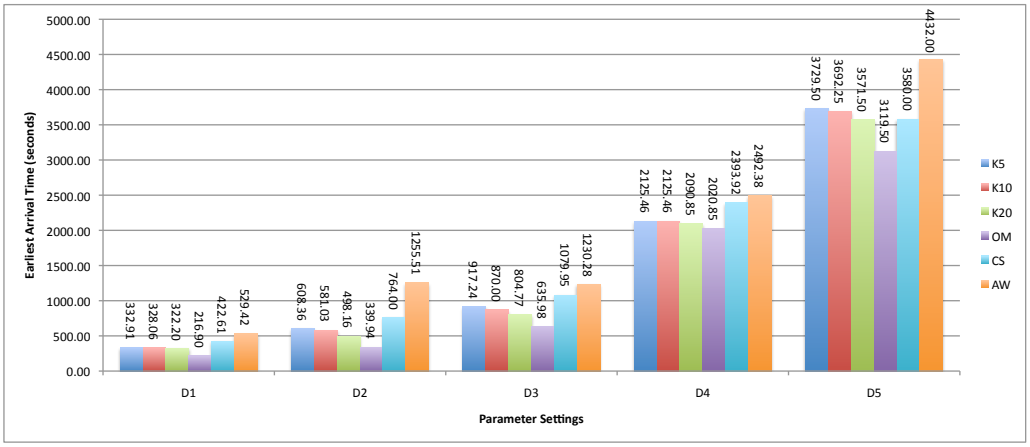


Figure 8.16. *EAT* found by all routing heuristics for each distribution on Random network topology.

	D1			D2			D3			D4			D5		
	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW
K5	0	12.8	12.8	0	3.7	3.7	-0.6	1.2	1.2	-0.2	6.9	6.9	0	3.7	4.9
K10	0	12.8	12.8	0	3.7	3.7	-0.6	1.2	1.2	-0.2	6.9	6.9	0	3.7	4.9
K20	0	12.8	12.8	0	3.7	3.7	-0.6	1.2	1.2	-0.2	6.9	6.9	0	3.7	4.9

(a) Results for EDG network topology (Figure 8.12a).

	D1			D2			D3			D4			D5		
	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW
K5	0	1.1	5.4	-5.3	27.8	29.7	-4.1	17.7	19.7	-3.5	4.6	9.5	0	0	0
K10	0	1.1	5.4	-4.8	28.2	30.1	-4.1	17.7	19.7	-3.5	4.6	9.5	0	0	0
K20	0	1.1	5.4	-4.8	28.2	30.1	-4.1	17.7	19.7	-3.5	4.6	9.5	0	0	0

(b) Results for NSF network topology (Figure 8.12b).

	D1			D2			D3			D4			D5		
	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW
K5	-10.8	-4.3	3.6	-8.5	-4.8	1	-24.1	22.7	28.6	-6.4	0.7	5.6	-14.1	4.7	19.1
K10	-7.4	-1.1	6.6	-5.6	-2	3.7	-27	20.9	27	-6.4	0.7	5.6	-9.8	8.3	22.1
K20	-3.3	2.7	10.1	-4.7	-1.2	4.5	-15.5	28.1	33.6	-4.5	2.5	7.3	-7.7	10	23.7

(c) Results for 5X5 network topology (Figure 8.12c).

	D1			D2			D3			D4			D5		
	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW	OM	CS	AW
K5	-53.6	21.2	37.2	-79	20.4	51.6	-44.2	15.1	25.4	-5.2	11.2	14.7	-19.6	-4.2	15.9
K10	-51.5	22.4	38	-70.9	24	53.7	-36.8	19.4	29.3	-5.2	11.2	14.7	-18.4	-3.1	16.7
K20	-48.6	23.8	39.1	-46.5	34.8	60.3	-26.5	25.5	34.5	-3.5	12.7	16.1	-14.5	0.2	19.4

(d) Results for Random network topology (Figure 8.12d).

Table 8.3. Improvement percentage of *KSP* variants for *EAT* over other algorithms for four network topologies.

any path that allows transmitting the data from the source to the destination within the given time horizon even though such a path exists). To study this further, we collected the data shown in Table 8.4. The only algorithm which cannot miss an existing path is *OM*, so it was used to check if a path exists at all in each given instance of the routing problem. The total number of instances where *OM* finds a feasible path was used to calculate for each of the other algorithms the percentage of instances where that algorithm misses an existing path. For the EDG topology, none of the algorithms ever missed an existing path, presumably because of the relatively small network size. In the case of the NSF topology, *AW* missed an existing path several times for $D1$ and $D5$, and once for $D4$. *CS* missed an existing path once. None of the *KSP* versions missed a path for the NFS topology. Although the topology is neither large nor dense (in terms of the ratio of edges to nodes), the reason why *AW* misses a path could be its path pruning technique, which was discussed earlier in Section 8.2. The 5X5 topology is a denser and larger network, which made heuristics miss a path more often than for

	K5	K10	K20	CS	AW
EDG Network					
$D1$	0	0	0	0	0
$D2$	0	0	0	0	0
$D3$	0	0	0	0	0
$D4$	0	0	0	0	0
$D5$	0	0	0	0	0
NSF Network					
$D1$	0	0	0	1.1	2.4
$D2$	0	0	0	0	0
$D3$	0	0	0	0	0
$D4$	0	0	0	0	1
$D5$	0	0	0	0	2.5
5X5 Network					
$D1$	9.7	7.5	4.3	3.2	4.3
$D2$	6.5	6.5	5.2	2.6	3.9
$D3$	0	0	0	0	0
$D4$	10.3	10.3	5.1	5.1	5.1
$D5$	13.6	8	5.7	3.4	3.4
Random Network					
$D1$	0	0	0	0	0
$D2$	0	0	0	0	0
$D3$	1.8	0.9	0.9	2.7	2.7
$D4$	18.4	15.8	11.8	25	34.2
$D5$	15.6	11.1	7.8	14.4	21.1

Table 8.4. Percentage of instances where the algorithms fail to find an existing path for the four network topologies.

the smaller topologies. Another factor which should be considered is the setting of the parameters, as tighter settings make it harder to find a path. For $D3$ the parameter settings are relaxed as compared to $D1$ and $D2$, and they are gradually made tighter for $D4$ and $D5$. It is noticeable that $K5$ is the heuristic that has the lowest chance to find a feasible path. This drawback is related to the remarks made in Section 8.2 stating that in larger networks with tighter parameter settings, a smaller value of K might cause the algorithm to miss an existing path. By increasing the value of K, the chances for missing an existing path can be reduced, as the results of $K20$ show. The Random topology is also dense and large, and the parameter settings are gradually made tighter from $D1$ to $D5$. *CS* and *AW* missed existing paths more often than the *KSP* variants. It seems that the topology of the Random network plays a major role in this result. We conclude that in the AR environment, larger size and density of a network and tighter parameter settings can make it harder for heuristics to find an existing path.

Time complexity and CPU time

Another aspect for the evaluation is the time complexity of the heuristics and their CPU usage time during the simulation experiments. According to our analysis, the time complexity of the polynomial heuristics by Varvarigos et al. is $\mathcal{O}(|V| \cdot T \cdot \log(|V| \cdot T) + |E| \cdot T^3)$ which is greater than the time complexity of the *KSP*-based heuristic, which is $\mathcal{O}(|V| \cdot |E| \cdot K \cdot T)$ (see Section 8.3). To measure the CPU usage time, we adopted the following procedure. At the start of a simulation run, the utilisation profiles of all links in the simulated network are populated. Each algorithm is run one by one on the same instance in the simulation. The utilisation profile remained the same for all algorithms. It was observed that an algorithm which is run earlier runs more slowly and one which is run later runs faster, which could be an effect of the availability of data (like the utilisation profiles) in the system cache of the simulation platform. To get fairer measurements, each algorithm was run twice and the CPU time was measured for the second run. Still the results should only be seen as a rough indication of the running time. A point that needs to be mentioned regarding the running time of the *KSP* variant is that due to its implementation technique the running time may vary for the same network with the same utilisation profile size. This variation is related to the path availability, as the algorithm runs faster if fewer paths are feasible and need to be added to the path lists of the nodes. The running time of the algorithms for all network topologies are given in seconds in Table 8.5.

For the smaller networks, EDG and NSF, the running time differences are marginal. For the Random network, which has more vertices and edges, the *KSP* heuristic K20, which is the slowest of the *KSP* variants, is 10 to 20 times faster than *AW* for different parameter settings, and the *CS* heuristic is 4 to 7 times slower than *AW* for the same settings.

For the 5X5 network, K20 is an order of magnitude faster than *AW*, and *CS* performed worse than *AW* especially for $D1$ and $D3$. The reason is the network topology of the 5X5 network and the domination condition used by *AW* and *CS* (see (4.7)). This can

	K5	K10	K20	OM	CS	AW
EDG Network						
$D1$	0.2	0.5	0.7	0.6	0.3	0.3
$D2$	0.2	0.5	0.8	0.6	0.2	0.3
$D3$	0.2	0.5	0.5	0.6	0.3	0.3
$D4$	0.1	0.1	0.1	0.6	0.3	0.3
$D5$	0.1	0.2	0.2	0.6	0.3	0.3
NSF Network						
$D1$	0.5	1.0	1.6	3.6	0.5	0.5
$D2$	0.2	0.2	0.2	3.6	1.0	0.5
$D3$	0.3	0.5	0.5	3.6	0.8	0.5
$D4$	0.1	0.1	0.1	3.6	1.2	0.5
$D5$	0.1	0.1	0.1	3.6	1.2	0.5
5X5 Network						
$D1$	0.06	0.06	0.2	747.6	133.5	8.3
$D2$	0.06	0.06	0.2	601.5	4.5	5.0
$D3$	0.05	0.05	0.2	810.6	574.7	16.8
Random Network						
$D1$	0.07	0.1	0.2	52.7	2.1	2.0
$D2$	0.04	0.07	0.1	61.0	13.7	2.0
$D3$	0.06	0.13	0.2	14.7	5.2	1.6
$D4$	0.05	0.1	0.1	13.8	6.4	1.9
$D5$	0.04	0.07	0.1	14.1	7.8	1.8

Table 8.5. Indication of running time (in seconds) of algorithms in the simulation.

be explained as follows. As the delay of each link in the 5X5 network is the same, when the set of non-dominated paths from node 1 to node 25 (see Figure 8.15) is made on the basis of the condition that a path can dominate another only if the delay of the path is strictly less than the delay of the other path, then a large number of paths will remain in the set because of the existence of many alternative paths having the same minimum delay for this network topology. Additionally, the second condition of the domination check will not make a difference if all the paths are made completely feasible or unfeasible because the weight of *CAV* or the number of runs of consecutive ones in a *CAV* will be the same. If the condition related to delay is modified to require inly that the delay of one path is at most the delay of the other path (i.e., domination is possible between paths of the same delay), then this will prune many paths and make the *CS* and *AW* algorithms faster for the above-mentioned situation. In situations where paths are mostly feasible, the new modified pruning condition will help, but if the availabilities of the paths are variable then the danger of moving further away from the optimal path may also increase.

In summary, it can be concluded that the *KSP* variants do not only have a better theoretical time complexity than *CS* and *AW*, but are also observed to run more quickly with respect to actual CPU time measured in the simulation experiments.

8.5 Conclusion

In this chapter we have analysed routing problem in the AR environment. We proved that finding a path from a source node to a destination node with earliest arrival time of the data is an NP-hard problem in the setting with non-zero edge delays. For this problem, a variant of the *KSP* heuristic was proposed, which outperformed the *CS* and *AW* heuristics proposed by Varvarigos et al. in terms of earliest arrival time. The theoretical time complexity of the *KSP* variant was also shown to be better than the complexity of the *CS* and *AW* heuristics. It was also indicated that on some networks the actual running time of the *KSP* variant is an order of magnitude faster than both of these competing heuristics.

Chapter 9

Conclusion and Future Work

We summarise the thesis in Section 9.1, and the plans for the future are discussed in Section 9.2.

9.1 Summary

9.1.1 PDCPG

The main objective of this thesis is to explore ways to improve the scheduling of workflows in Grid environments with advance reservation. Taking advantage of the future information regarding the availability of resources, we proposed PDCPG, a new partner based resource mapping technique, and combined it with the dynamic critical path based job selection technique. PDCPG showed great potential over other resource mapping techniques. The key feature of the partner based resource mapping is to schedule partner jobs on a single resource so that the chances of child job(s) getting scheduled on the same resource become higher, thus minimising the communication cost between the jobs and achieving a shorter makespan of the workflow. PDCPG combines the advantages of having the low running time of list scheduling heuristics and the chance of minimising the communication cost between interdependent job like cluster based scheduling heuristics. In PDCPG, jobs are prioritised as in list scheduling heuristics, and at the time

of scheduling a cluster of jobs is conditionally scheduled on the same resource. PDCPG variants not only have the same time complexity as DCPG or HEFT depending on which heuristic's job selection technique it is combined with, but it also tends to practically reduce the time for resource selection in the AR environment. We believe that it is not always a good decision to schedule a cluster of jobs on the same resource especially when the situation of capacity availability is tight. Thus, we have proposed a balanced condition by which PDCPG schedules clusters of jobs on the same resource.

The performance of PDCPG is compared with DCPG, which is known to outperform HEFT in the non-AR environment. Different levels of improvement are observed in different situations. Considerable improvements are achieved by PDCPG in situations when the load on the systems is high and the communication cost of a workflow is higher than its computation cost. Only marginal improvements or even a worsening of the makespan is observed for situations when the communication cost of a workflow is similar to its computation cost. Furthermore, we have noticed some factors which can affect the improvement achieved by PDCPG, which are as follows. The first factor is the pattern/trend of partner jobs and its conformance with the pattern/trend of available CPUs per faster resources. The second factor is the suitability of the size of the fragments in the capacity availability vectors of the resources, which may cause the other heuristics to find an earlier available space on other resource while PDCPG may fit the jobs into the fragments of available capacity on the same resource as a partner job. In our experiments, the size of the capacity availability fragments is dependent on the value of the parameter that determines the number of consecutive slots that receive the same random load value when filling in the utilisation profile. Smaller values of this parameter create smaller fragments.

We also study PDCPG in a dynamic environment where schedules may have to be altered because of resource failures. It is observed that in most of the cases, PDCPG achieved huge improvements over DCPG for the final schedule makespan. One of the reasons for such a huge improvement is the resource model we used for the experiments, where the fastest resource fails, making more than one resources the fastest resources among

the remaining ones. Thus, DCPG may spread the scheduled jobs over several fastest available resources while PDCPG may fit the interdependent jobs on fewer resources, improving the makespan.

9.1.2 HDCPG

Inconsistency in the performance of different heuristics for workflow scheduling in AR environments is observed in our experiments. Based on this observation, we proposed a heuristic called HDCPG, which is a hybrid of five list scheduling heuristics. The five different schedules can be computed in parallel based on the basis of future resource availability information. Thus, the parallel running time of HDCPG is the same as the sequential running time of the slowest amongst the five heuristics. We also introduced a new critical path calculation method which is used in two of the five heuristics for the job selection. In the new critical path technique, we convert the cost of an edge in a workflow to zero based on a criterion based on the idea that higher communication costs may force heuristics to schedule two interdependent jobs on the same resource, which may change the critical path later.

HDCPG showed huge improvements over the other heuristics, HEFT, DCPG and PDCPG, especially for the low granularity scenario. For high granularity, HEFT is only marginally worse than HDCPG, which shows that HEFT is competitive against a combination of multiple heuristics, thus making HEFT a better choice in such cases if only one heuristic has to be chosen to minimise the execution overhead.

9.1.3 Routing in AR Environments

In AR environments where network links and computing resources can both be reserved, better schedules can be obtained for workflows if better paths can be found for transmitting data files from one resource to another. With this motivation, we analysed variants of the routing problem in OBS networks with AR support. The variants are combinations of different restrictions on the type of allowed path, the transfer duration and the delay

of the network links. Except for one variant and the special case where the link delays are zero, for all the other variants it was proved to be NP-hard to find an optimal path with earliest arrival time for a data transmission from a source node to a destination node. This motivates the study of heuristics for the routing problem.

We proposed a polynomial time K-shortest path *(KSP)* variant for finding a simple path for the earliest arrival of data in an OBS network which supports AR. The *KSP* variant is compared with the heuristics proposed by Varvarigos et al. [70] in different network topologies with different parameter settings. Overall the *KSP* variant performed well for most of the cases while having a comparatively low running time. One problem with the *KSP* variant is determining which value of K is sufficient for getting a good path, depending on the network topology and load. We believe that the *KSP* variant has the potential to be a good candidate for adoption in many real world situations.

9.1.4 PDCPG in Non-AR Environments

Originally PDCPG was designed for the AR environment, but many existing Grids do not support AR. Thus we also studied PDCPG in non-AR environments. A small modification was made in PDCPG to adapt it to the non-AR environment. The performance of PDCPG was very encouraging in terms of workflow makespan in the case of low granularity. Therefore, it can be concluded that PDCPG has potential for both AR and non-AR environments.

9.1.5 Critical Analysis of PDCPG

Even though it has been shown that PDCPG can gain makespan improvement over DCPG and HEFT for certain workflows when the granularity is low, it is dependent on the number of partner jobs and the number of available CPUs per resource to accommodate them. In case of workflows with many partner jobs, very few or no partner jobs PDCPG may not give considerable improvement. PDCPG may also lose its other advantage of finding a place quickly for the partner jobs on the resources in an AR environment when

the availability of the resources is tight enough to cause the algorithm to look through all the resources.

9.2 Future Work

Grid workflow scheduling is a complicated problem, and it is hard to identify the best scheduling method. One may find a better method for a particular workflow in a certain situation, but the method may not work well for another workflow in the same situation, or for the same workflow in another situation. In the future, we plan to devise a general workflow scheduling algorithm which takes into account the workflow structure, the granularity of the workflow and the characteristics of the available resource pool. We plan to investigate and develop a new critical path method for job selection that will also consider the relationship between the costs of the edges and nodes and the available resource pool.

We also plan to design an improved partner based resource mapping technique in which a resource will be selected for the first of several partner jobs if it has the capacity to accommodate further partner jobs and the completion time of the first partner job on the resource will be within a margin of affordable delay. The margin of affordable delay will be decided based on the completion time of the job on the fastest available resource.

Partner based resource mapping may create a higher load on few resources while leaving other resources underutilised, which may not be appreciated by the resource provider. It is planned to modify the mapping technique to achieve a more balanced allocation that satisfies both the user and the provider of Grid services. Other objectives that can also be considered are energy efficiency and budget constraints.

Furthermore, the workflow scheduling problem in Cloud computing environments is also of interest to us. In Cloud environments we may have limited time and limited resources to schedule a workflow, so getting an efficient solution in such situations will be challenging task.

Appendix A

Modifications Made to GridSim

The modifications and additions made to the GridSim to meet our simulation requirements are discussed here. Most of the modifications made to GridSim are related to including the AR features according to our requirements. There are also a few changes that are are due to the implementation of the OBS network simulation. In the coming sections the new and modified GridSim entities are discussed.

User Entity

Among the different types of users supported by GridSim, we merged two types of users for our simulations. One of the two types of user is the *DataGridUser*, which can perform actions related to data files such as file attribute request, file location request, or requesting the replication of a file to a destination. The other type of user is the *AdvanceReservationUser*, which can perform AR related activities such as creating, committing, modifying and cancelling the reservation for a job. By merging the *DataGridUser* and the *AdvanceReservationUser* we created a new *AdvanceReservationDataGridUser* which contains all the characteristics from both types of users. Additionally, the *AdvanceReservationDataGridUser* can request the creation of an advance reservation for a data file transfer. This reservation initiates a file transfer which goes through an explicit route to a resource where it is required for the execution of a job. The working of the

newly added explicit routing, which is a feature of OBS networks, is explained in the next section.

Router Entity

After a reservation is made for a job on a resource, additional reservation(s) for the transfer of file(s) are also created. These file(s) are required as inputs by the job. For this file transfer an explicit route, which is decided for the file for the scheduled job, is used. As the result of a job execution reservation, a unique reservation ID is generated, which is used to assign a unique booking ID to each required input data file. This unique booking ID is sent along with an explicit route for the data file in a control packet to all the included routers in the explicit route. An entry is made for this booking ID and its associated next node to be followed in the routing table of the router. When a file transfer is started, the whole file is sent in a single data packet with a booking ID. Upon receiving the data packet, the router forwards it to the next node after looking up its booking ID. Even though this data transmission technique is not an exact simulation of data transmission in an actual OBS network, the arrival time of the data is exactly the same as it would be in OBS. The data packet does not go through router buffering or packet scheduling processes that could cause extra delay.

Network Link Entity

GridSim does not support AR for network links. To add this feature, a utilisation profile was added to the link entity (see Section 4.1). When an instance of the link entity is created during a simulation run, the utilisation profile is populated. To calculate a path for the input file transfer, the utilisation profiles from the links are requested by the user broker. After a path is selected for the reservation, the utilisation profiles are updated accordingly. In our simulation setup, the links are bidirectional and one utilisation profile is maintained for both directions instead of maintaining a utilisation profile for each

direction. Maintaining a single utilisation profile for both directions has no effect on the results of the scheduling and routing experiments reported in this thesis. In the case of the scheduling experiments, sufficient bandwidth is made available to avoid any network bottleneck. The routing related experiments are conducted for only one connection per simulation, thus a link is used only in one direction. In the future, an extension of the implementation to support separate utilisation profiles for both directions of a link would be meaningful.

Computing Resource Entity

A new resource type *ARDataGridResource* was created by merging two of the available types of resources, *DataGridResource* and *ARGridResource*. Before describing *ARDataGridResource*, some of the features that are specific to the *DataGridResource* entity can be given as follows:

- It can handle a *datagridlet*, a gridlet which requires an input file for execution.
- If a required file does not exist on the resource, then the *DataGridResource* requests the replication of the required file(s).
- Other data related operations are creating, deleting, replicating and delivering a file.

For performing these above-mentioned operations, each *DataGridResource* needs its own *Replica Manager* entity. The *ARGridResource* can create, modify, cancel and commit reservations for job execution. To perform these functions, each *ARGridResource* has its own *ARSimpleSpaceShared* scheduler entity. The *ARDataGridResource* inherits all the operations from *DataGridResource* and *ARGridResource*. Additionally it has a utilisation profile which maintains the availability of CPUs in future time slots. The utilisation profile is populated at the time of creation of an instance of *ARDataGridResource*. In the procedure for new reservations, first the start time is decided using the utilisation profile, then the start time and the length of the job is sent to the resource scheduler

which creates a reservation as it was done by the original *ARGridResource* entity. We also added the features of creating a reservation for a file transfer and initiating a file transfer as a result of a reservation. The *ARDataGridResource* entity is used in most of the AR related experiments.

Another added computing resource entity is the *ARDataGridResourceWithFailure* which extends the *ARDataGridResource* with a failure capability. The failure-related behaviour and attributes are taken from the *GridResourceWithFailure* entity. The *ARDataGridResourceWithFailure* is used in simulations in which failures may occur after the reservations for the job executions have been made.

Along with these changes, several small modifications were also made in the simulator according to the requirements of specific scenarios.

Bibliography

[1] Biogrid Project. http://www.biogrid.jp/. Last accessed: April, 2012.

[2] EMAN. http://blake.bcm.tmc.edu/eman/. Last accessed: April, 2012.

[3] Globus Project. http://www.globus.org. Last accessed: April, 2012.

[4] GriPhyN. http://www.griphyn.org/. Last accessed: April, 2012.

[5] Human Genome Project. http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml. Last accessed: April, 2012.

[6] LEAD. http://portal.leadproject.org/. Last accessed: April, 2012.

[7] Montage. http://montage.ipac.caltech.edu/. Last accessed: April, 2012.

[8] Partice Physics Data Grid Project. http://ppdg.net/. Last accessed: April, 2012.

[9] The DataGrid Project. http://eu-datagrid.web.cern.ch/eu-datagrid. Last accessed: April, 2012.

[10] Unicore Project. http://www.unicore.org. Last accessed: April, 2012.

[11] WIEN2K. http://www.wien2k.org/. Last accessed: April, 2012.

[12] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima. Performance evaluation model for scheduling in a global computing system. *The International Journal of High Performance Computing Applications*, 14:268–279, 2000.

[13] S. G. Akl and F. Dong. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, School of Computing — Queen's

University, Kingston, Ontario, Canada, January 2006.

[14] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45:56–61, November 2002.

[15] S. Baskiyar and C. Dickinson. Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors using task duplication. *Journal of Parallel and Distributed Computing*, 65(8):911–921, August 2005.

[16] W. H. Bell, D. G. Cameron, L. Capozza, A. P. Millar, K. Stockinger, and F. Zini. Simulation of dynamic grid replication strategies in OptorSim. In *Proceedings of the Third International Workshop on Grid Computing*, GRID '02, pages 46–57, 2002.

[17] R. E. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[18] S. Benedict and V.Vasudevan. Scheduling of scientific workflows using simulated annealing algorithm for computational grids. *International Journal of Soft Computing*, 2(5):606–611, 2007.

[19] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, Jignesh M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan. New grid scheduling and rescheduling methods in the grads project. *International Journal of Parallel Program.*, 33:209–229, June 2005.

[20] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05) - Volume 2*, pages 759–767, 2005.

[21] K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149(2):268 – 281, 2003.

[22] G. Boulmier. http://www.jgrapht.org/javadoc/org/jgrapht/alg/KShortestPaths.html, 2007. Last accessed: April, 2012.

[23] T. D. Braun, H. Jay Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal Parallel Distributed Computing*, 61(6):810–837, June 2001.

[24] L. Burchard, H. Heiss, B. Linnert, J. Schneider, and C. A. F. De Rose. VRM: a failure-aware grid resource management system. *International Journal of High Performance Computing and Networking*, 5(4):215–226, 2008.

[25] H. Casanova. SimGrid: A toolkit for the simulation of application scheduling. In *1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'01)*, pages 430–437, Brisbane, Australia., 2001.

[26] K. Christodoulopoulos, N. Doulamis, and E. Varvarigos. Joint communication and computation task scheduling in grids. In *CCGrid '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 17–24, Washington, DC, USA, 2008. IEEE Computer Society.

[27] B. Cirou and E. Jeannot. Triplet: A clustering scheduling algorithm for heterogeneous systems. In *International Conference on Parallel Processing Workshops*, pages 231 – 236, 2001.

[28] L. B. Costa, L. F., E. Arajo, G. Mendes, R. Coelho, W. Cirne, and D. Fireman. MyGrid: A complete solution for running bag-of-tasks applications. In *Proceedings of the SBRC 2004 Salao de Ferramentas (22nd Brazilian Symposium on Computer Networks III Special Tools Session)*, 2004.

[29] M. I. Daoud and N. Kharma. A high performance algorithm for static task scheduling in heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 68(4):399–409, April 2008.

[30] H. El-Rewini and T. G. Lewis. Scheduling parallel program tasks onto arbitrary

target machines. *Journal of Parallel and Distributed Computing*, 9(2):138–153, 1990.

[31] R. McNab F. Howell. SimJava: A discrete event simulation package for Java with applications in computer systems modelling. In *1st International Conference on Web-based Modelling and Simulation*, 1998.

[32] L. R. Ford and D. R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.

[33] L.R. Ford and D.R. Fulkerson. Flows in Networks. *Princeton Univ. Press*, 1962.

[34] I. Foster. What is the Grid? - a three point checklist. *GRID today*, 1, 2002.

[35] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and Grid computing 360-degree compared. In *Grid Computing Environments Workshop, GCE 2008*, pages 1–10, 2008.

[36] R. F. Freund and H. J. Siegel. Guest editor's introduction: Heterogeneous processing. *Computer*, 26:13–17, 1993.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[38] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., 1989.

[39] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966.

[40] L. Guodong, C. Daoxu, W. Daming, and Z. Defu. Task clustering and scheduling to multiprocessors with duplication. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, 2003.

[41] R. A. Gurin and A. Orda. Networks with advance reservations: The routing perspective. In *INFOCOM, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies.*, pages 118–127, 2000.

[42] W. Hoffman and R. Pavley. A method for the solution of the nth best path problem.

Journal of the ACM, 6:506–514, October 1959.

[43] D. Hollingsworth. Workflow management coalition - the workflow reference model. Technical report, Workflow Management Coalition, 1995.

[44] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 24(2):280–289, 1977.

[45] B. Jones. Building the European eInfrastructure Ecosystem for Data Intensive Science. Keynote talk in High Performance Computing and Simulation Conference, 2010.

[46] G. Juve, E. Deelman, K. Vahi, and G. Mehta. Experiences with resource provisioning for scientific workflows using Corral. *Scientific Programming*, 18:77–92, April 2010.

[47] R. Kolisch and A. Sprecher. PSPLIB - A project scheduling problem library. *European Journal of Operational Research*, 96(1):205–216, 1997.

[48] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

[49] Y. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31:406–471, December 1999.

[50] Y. K Kwok and I. 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.

[51] M. De Leenheer, D. Van, E. Van Breusegem, P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt, P. Demeester, D. Simeonidou, R. Nejabati, A. Tzanakaki, and I. Tomkos. An OBS based grid architecture. *IEEE Global Telecommunication Conference (GLOBECOM), Workshop on High-Performance Global Grid Networks*, December 2004.

[52] C. L. McCreary, M. A. Cleveland, and A. A. Khan. The problem with critical path scheduling algorithms. Technical report, Department of Computer Science and

Engineering, Auburn University, 1996.

[53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[54] E.F. Moore. The shortest path through a maze. *Proceedings of the International Symposium on the Theory of Switching*, pages 285–292, 1957.

[55] B. Nazir and T. Khan. Fault tolerant job scheduling in computational Grid. In *International Conference on Emerging Technologies*, pages 708–713. Ieee, 2006.

[56] R. Nejabati. Grid optical burst switched networks (GOBS). *Open Grid Forum Draft GFD-I.128*, April 2008.

[57] A. O'Brien, S. Newhouse, and J. Darlington. Mapping of scientific workflow within the E-Protein project to distributed resources. In *UK e-science all-hands meeting, AHM 2004, Nottingham, UK*, pages 404–409, August 2004.

[58] M. Rahman, S. Venugopal, and R. Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*, pages 1–8, 2007.

[59] M. Resende and C. Ribeiro. *Greedy Randomized Adaptive Search Procedures*, volume 57 of *International Series in Operations Research and Management Science*. Springer New York, 2003.

[60] R. Sakellariou and H. Zhao. A hybrid heuristic for DAG scheduling on heterogeneous systems. *18th International Parallel and Distributed Processing Symposium 2004 Proceedings*, pages 111–123, 2004.

[61] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programing*, 12:253–262, December 2004.

[62] G. Singh, C. Kesselman, and E. Deelman. Performance impact of resource provisioning on workflows. Technical report, Department of Computer Science, University

of Southern California, 2006.

[63] H. Singh and A. Youssef. Mapping and scheduling heterogeneous task graphs using genetic algorithms. In *Proceedings Heterogeneous Computing Workshop (HCW'96)*, pages 86–97, 1996.

[64] W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. *Proceedings 14th International Parallel and Distributed Processing Symposium*, pages 127–132, 2000.

[65] H. Song, X. Liu, D. Jaken, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The MicroGrid: A scientific tool for modeling computational Grids. In *IEEE Supercomputing*, pages 4–10, Dallas, TX., 2000.

[66] A. Sulistio. *Advance Reservation and Revenue-based Resource Management for Grid Systems*. PhD thesis, Department of Computer Science and Software Engineering, The University of Melbourne, Australia, 2008.

[67] A. Sulistio, C. S. Yeo, and R. Buyya. Visual modeler for Grid modeling and simulation (GridSim) toolkit. In *International Conference on Computational Science*, pages 1123–1132, 2003.

[68] H. Topcuoglu, S. Hariri, and M. Y Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, 2002.

[69] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[70] E. Varvarigos, V. Sourlas, and K. Christodoulopoulos. Routing and scheduling connections in networks that support advance reservations. *Computer Networks*, 52(15):2988–3006, 2008.

[71] S. Venugopal and R. Buyya. A set coverage-based mapping heuristic for scheduling distributed data-intensive applications on global grids. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing(GRID 06)*, 2006.

[72] L. Wang, H. J. Siegel, V. R. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, 1997.

[73] M. Wieczorek. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Record*, 34(3):56–62, 2005.

[74] M. Wu and D. D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1:330–343, 1990.

[75] M. Wu and X. Sun. Self-adaptive task allocation and scheduling of meta-tasks in non-dedicated heterogeneous computing. *International Journal High Performance Computing Network*, 2:186–197, February 2004.

[76] A. YarKhan and J. Dongarra. Experiments with scheduling using simulated annealing in a grid environment. In *Proceedings of the Third International Workshop on Grid Computing*, GRID '02, pages 232–242, London, UK, UK, 2002. Springer-Verlag.

[77] M. Yoo and C. Qiao. Optical burst switching (OBS) - a new paradigm for an optical internet. *International Journal of High-speed Networks*, 8(1):69–84, 1999.

[78] L. Young, S. Mcgough, S. Newhouse, and J. Darlington. Scheduling architecture and algorithms within the ICENI Grid middleware. In *UK e-Science All Hands Meeting*, pages 5–12, 2003.

[79] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, 2005.

[80] J. Yu and R. Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programing.*, 14(3,4):217–230, December 2006.

[81] J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. *Studies in Computational Intelligence*, 146:173–214, 2008.

[82] Z. Yu and W. Shi. An adaptive rescheduling strategy for grid workflow applications. In *IEEE International Parallel and Distributed Processing Symposium 2007*, IPDPS 07, pages 1–8, 2007.

[83] J. Zheng and H. T. Mouftah. Routing and wavelength assignment for advance reservation in wavelength-routed WDM optical networks. In *ICC 2002: IEEE International Conference on Communications 2002*, volume 5, pages 2722–2726, 2002.

[84] W. Zheng. *Explorations in Grid Workflow Scheduling*. PhD thesis, School of Computer Science, University of Manchester, 2010.