7-28-2011

Optimization and Management of Large-scale Scientific Workflows in Heterogeneous Network Environments: From Theory to Practice

Yi Gu

To the University Council:

The Dissertation Committee for Yi Gu certifies that this is the final approved version of the following electronic dissertation: “Optimization and Management of Large-scale Scientific Workflows in Heterogeneous Network Environments: From Theory to Practice.”

_____
Qishi Wu, Ph.D.
Major Professor

We have read this dissertation and recommend
its acceptance:

_____
Lih-Yuan Deng, Ph.D.

_____
King-Ip Lin, Ph.D.

_____
Vinhthuy Phan, Ph.D.

Accepted for the Graduate Council:

_____
Karen D. Weddle-West, Ph.D.
Vice Provost for Graduate Programs

OPTIMIZATION AND MANAGEMENT OF LARGE-SCALE SCIENTIFIC WORKFLOWS IN HETEROGENEOUS NETWORK ENVIRONMENTS: FROM THEORY TO PRACTICE

by

Yi Gu

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

August 2011

This dissertation is dedicated to my beloved parents.

Acknowledgements

In the past five years of my Ph.D. study at the University of Memphis, I had the privilege of working with a great many people who have contributed to the final production of this dissertation in different ways. I am sincerely grateful to all those who offered their help, support, and inspiration throughout my research conduct, which have greatly enriched my graduate experience that I will cherish forever.

First of all, I would like to express my deepest gratitude and appreciation to my advisor, Prof. Qishi Wu, for bringing me to the University of Memphis and guiding me through my entire Ph.D. study with constant support and encouragement. His mentorship was paramount in providing a well rounded experience to be consistent with my long-term career goals. I have not only learned the beauty of science from him, but the importance of creativity, scrupulousness, integrity, and hard work as a researcher by his own example. Prof. Wu has been also very supportive and considerate in many non-academic aspects, which have had significant impacts on my life as an international student.

I would like to express my heartfelt gratitude to my committee members, Prof. Lih-Yuan Deng, Prof. King-Ip Lin, and Prof. Vinhthuy Phan, for their constructive comments and insightful suggestions on my dissertation. My sincere thanks also go to all other faculty and staff members in our department and university for their help during my study. I had the pleasure of collaborating with Prof. Yves Robert and Prof. Anne Benoit at École Normale Supérieure de Lyon, France, who offered great help with some parts of the complexity proof. I would also like to acknowledge my other collaborators, Mr. Michael Reuter, Dr. Stephen Miller, and Dr. Nageswara Rao at Oak Ridge National Laboratory, Dr. Xin Liu, Dr. Dantong Yu, Dr. Yangang Liu, and Dr. Wuyin Lin at Brookhaven National Laboratory, for sharing with me their expert and domain knowledge. My special thanks go to Dr. Michael Smith at Oak Ridge National Laboratory for his valuable comments and suggestions that have greatly improved the presentation of this dissertation. I would like to thank all my friends and lab mates, Mr. Xiaoshan Cai for his help with the SDEDS GUI implementation, Mr. Xukang Lu, Mr. Yunyue Lin, and Mr. Patrick Brown for their great contributions to the development of the SWAMP system.

Finally, I am deeply grateful to my parents for their spiritual and physical support. Without their unconditional, unchanging, and unending love and care, this work would never have come into existence.

Table of Contents

List of Figures

Abstract

Yi Gu, Ph.D., The University of Memphis, August 2011, “Optimization and Management of Large-scale Scientific Workflows in Heterogeneous Network Environments: From Theory to Practice”. Major Professor: Prof. Qishi Wu

Next-generation computation-intensive scientific applications feature large-scale computing workflows of various structures, which can be modeled as simple as linear pipelines or as complex as Directed Acyclic Graphs (DAGs). Supporting such computing workflows and optimizing their end-to-end network performance are crucial to the success of scientific collaborations that require fast system response, smooth data flow, and reliable distributed operation.

We construct analytical cost models and formulate a class of workflow mapping problems with different mapping objectives and network constraints. The difficulty of these mapping problems essentially arises from the topological matching nature in the spatial domain, which is further compounded by the resource sharing complicacy in the temporal dimension. We provide detailed computational complexity analysis and design optimal or heuristic algorithms with rigorous correctness proof or performance analysis. We decentralize the proposed mapping algorithms and also investigate these optimization problems in unreliable network environments for fault tolerance.

To examine and evaluate the performance of the workflow mapping algorithms before actual deployment and implementation, we implement a simulation program that simulates the execution dynamics of distributed computing workflows. We also develop a scientific workflow automation and management platform based on an existing workflow engine for experimentations in real environments. The performance superiority of the proposed mapping solutions are illustrated by extensive simulation-based comparisons with existing algorithms and further verified by large-scale experiments on real-life scientific workflow applications through effective system implementation and deployment in real networks.

Chapter 1

Introduction

1.1 Overview

Advances in supercomputing technology are expediting the transition in various basic and applied sciences from traditional laboratory-controlled experimental methodologies to modern computational paradigms involving complex numerical model analyses and large-scale simulations of physical phenomena, chemical reactions, climatic changes, and biological processes, as illustrated in Fig. 1.1. Such computation-based simulations and analyses have become an essential research and discovery tool in next-generation scientific applications and are producing colossal amounts of simulation data, on the order of terabyte at present and petabyte or even exabyte in the predictable future. Other scientific data of similar scales generated in broad science communities include real-world environmental observations such as satellite climate data [1] and multimodal sensor data, and high-throughput experimental measurements such as Spallation Neutron Source [2] and Large Hadron Collider [3]. Both computation and computing needs in these scientific applications have gone far beyond the capability of traditional solutions based on standalone PCs, and have driven the rapid deployment of a wide variety of system resources including high-performance computing facilities, data repositories, experimental facilities, network infrastructures, storage systems, and display devices across the nation and around the globe. These massively distributed resources must be pooled together to produce unprecedented data collections, simulations, visualizations, and analyses.

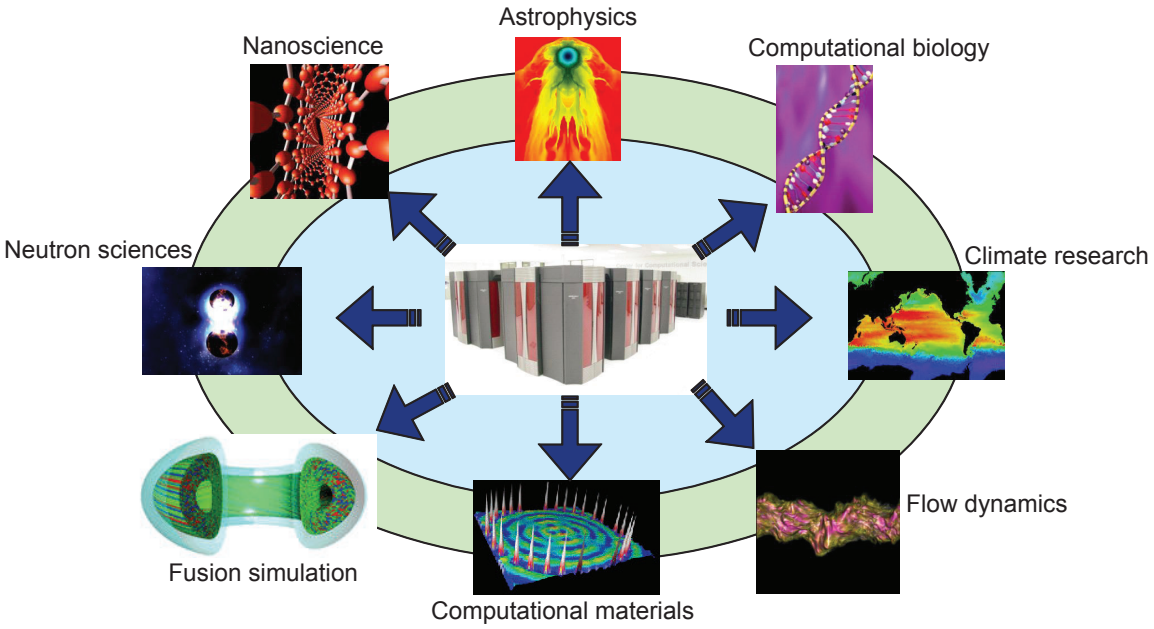


Figure 1.1: Impact of supercomputing on various scientific applications.

On the one hand, the e-science based on computational techniques with diverse physical, chemical, climatic, and biological models typically entails domain-specific computing solutions with multifarious performance requirements. On the other hand, the resources generally serve a large user base over wide-area networks and thus exhibit an inherent dynamic nature in their accessibility, availability, capacity and reliability. Therefore, the success of these extreme-scale scientific applications requires effective and efficient end-to-end solutions for geographically distributed users to transfer, process, visualize, analyze, and synthesize the data (raw data, intermediate results, or final outputs) in heterogeneous network environments for collaborative research. Fortunately, the ready availability of tremendous computational resources enables the development of robust problem-solving environments where large-scale, collaborative, computationally intensive applications can be set up, executed, and managed. No longer confined to just one fast machine, this work has moved to include a collection of large clusters – some with tens to hundreds of thousands of processor cores, grids (e.g., TeraGrid) with a dozen sites and 100K+ processors,

and supercomputers with up to 200K processors (i.e., IBM BlueGene/L and BlueGene/P, Cray XT5, and Sun Constellation). Effectively utilizing these high-performance computing resources presents a great opportunity for significant progress in a number of scientific fields.

1.2 Problem Statements

The computing tasks of these scientific applications commonly feature complex workflows consisting of many computing modules[1] with intricate inter-module execution dependencies. The execution of these modules typically involves the invocation of many diverse distributed computing tools or services for collaborative data analysis and knowledge discovery [4, 60, 111, 126]. Supporting such distributed computing workflows in heterogeneous network environments and optimizing their end-to-end performance are crucial to ensuring the success of mission-critical distributed e-science applications and maximizing the utilization of networked resources.

We consider two types of large-scale distributed computing workflow applications:

- **Unitary processing applications** where a single dataset is processed through the entire workflow. Typical examples include an interactive parameter update on a remote visualization system that triggers a number of individual processing subtasks for data filtering, isosurface extraction, geometry rendering, image compositing, and final display.

- **Sequential processing (or streaming) applications** where a series of datasets continuously flow through the entire workflow. Typical examples include a video-based real-time monitoring system for detecting criminal suspects that performs feature extraction and detection, facial reconstruction, pattern recognition, data mining and identity matching on continuously captured images.

[1] In some context, a computing module is also referred to as a subtask, an activity, a stage, a job, or a transformation.

In unitary processing applications, we consider the performance metric of end-to-end delay or latency; while in streaming applications, we consider the metric of frame rate[2] or throughput. A general optimization goal is to achieve Minimum End-to-end Delay (MED) for fast system response in the former and achieve Maximum Frame Rate (MFR) for smooth data flow in the latter.

A special case of workflows consists of computing modules that are to be executed in a sequential (linear) order in distributed environments with heterogeneous resources under different network constraints. We categorize the mapping problems for such pipeline-shaped workflows in distributed environments into six classes with two mapping objectives, i.e., MED and MFR, and three network constraints on node reuse:

1. MED with No Node Reuse (MED-NNR),
2. MED with Contiguous Node Reuse (MED-CNR),
3. MED with Arbitrary Node Reuse (MED-ANR),
4. MFR with No Node Reuse (MFR-NNR),
5. MFR with Contiguous Node Reuse (MFR-CNR),
6. MFR with Arbitrary Node Reuse (MFR-ANR).

Here, "contiguous node reuse" means that multiple consecutive modules along the pipeline may run on the same node, and "arbitrary node reuse" imposes no restriction on node reuse. Note that in the MED problems of unitary processing pipelined applications, the resources of a node are not shared even if multiple modules are mapped on it (i.e., node reuse) because only one module is running at any time point; while in the MFR problems of streaming pipelined applications, the resources of a reused node are shared by multiple concurrent modules executing on it with a sequence of datasets that need to be processed.

[2]Frame rate is the final data production rate at the last module in the workflow.

The workflows of many other applications have more complex structures that are often modeled as Directed Acyclic Graphs (DAGs), where each vertex represents a computing module and each directed edge represents a communication and execution dependency. A general workflow consisting of a large number of computing modules may have very complex execution dependencies that must be satisfied. In our workflow execution model, each module is considered as an autonomous computing agent: it receives data as input from one or more preceding modules, executes a predefined processing routine on the aggregate input data, and sends the results as output to one or more succeeding modules. We assume that neither can a module start the execution until all the needed input datasets arrive nor send out the results until the execution is successfully completed.

We consider the network environment as an overlay computer network consisting of heterogeneous computer sites or nodes interconnected by disparate network links. Depending on the network infrastructure, the topology of an overlay network may be complete as in the case of the Internet based on layer-3 IP routing, or not as in the case of most dedicated research testbed networks using layer-1 or 2 circuit/lambda switching or MPLS/GMPLS tunneling techniques. Note that even in Internet environments, the network topology may not be always complete because the network connectivity or facility accessibility could be largely affected by firewall settings on either routers or end systems. We would also like to point out that computer nodes are of different processing capabilities and network links are of different transfer properties in terms of bandwidth and minimum link delay in most wide-area networks. We assume that the bottlenecks of overlay links (i.e., path bandwidths) reside on the backbone, not on its edge, and are independent of each other.

The DAG-structured workflow mapping/scheduling problem has been extensively studied in the literature [25, 37, 56, 73, 75, 116, 123, 128, 129, 142], and is known to be NP-complete in a general sense [73] as well as some simplified cases [142] such as the mapping of linear pipelines, or restricted cases [128] such as the assignment of tasks with one or two time units on only two processors. We generalize the workflow mapping problem

with two different objectives: select an appropriate set of heterogeneous computer nodes in the overlay network of an arbitrary topology and assign each computing module in the workflow to one of those selected nodes to achieve (i) MED for fast system response in unitary processing applications with a single dataset and (ii) MFR for smooth data flow in streaming applications with time-series datasets. We further investigate the problem of optimizing the workflow end-to-end performance under fault-tolerant constraints.

1.3 Main Approaches

The workflow performance in terms of end-to-end delay or frame rate is determined by module execution time on computer nodes and data transfer time over communication links. The difficulty of these workflow mapping problems essentially arises from the topological matching nature in the spatial domain, which is further compounded by the resource sharing complicacy in the temporal dimension. We construct analytical cost models for computing workflows and computer networks, and formulate workflow mapping as a class of optimization problems based on different objectives and network constraints.

In linear computing pipeline mappings, we prove that MED-ANR can be solved in polynomial time; while the other five problems are NP-complete, among which we further prove that all NP-complete problems, except MFR-ANR, cannot be approximated by any constant factor (unless $P = NP$). We propose a set of mapping algorithms, named *Efficient Linear Pipeline Configuration* (ELPC), in which an optimal solution based on dynamic programming is designed for MED-ANR, and a heuristic approach is designed for each of the NP-complete problems [79–81, 84, 140, 142, 147, 149].

In general DAG-structured workflow mappings, for unitary processing applications, we first propose an efficient algorithm to compute the exact end-to-end delay of a mapped workflow with arbitrary node reuse and then develop a new workflow mapping algorithm *Recursive Critical Path* (RCP) based on a recursive optimization procedure to minimize the latency. For streaming applications, we first conduct a rigorous workflow stability analysis

and then develop a new *Layer-oriented Dynamic Programming* (LDP) solution based on topological sorting to identify and minimize the global bottleneck time [78, 83, 144]. We construct the cost models for workflows and networks subject to probabilistic node and link failures, and rigorously prove that the bi-objective optimization problem, i.e., MED/MFR and minimal failure rate, cannot be approximated within one constant factor. Consequently, we convert this problem to a reliability-constrained delay minimization or throughput maximization problem, and adapt the original RCP and LDP mapping algorithms to calculate the latency or throughput in a distributed manner under a given reliability constraint [82, 139].

For performance evaluation and comparison, we implement the proposed mapping solutions as well as four existing mapping algorithms, namely *Greedy* A^* [121], *Streamline* [26], *DLS* [125] and *Greedy*. We also develop a lightweight multi-threaded simulation program, *Simulation of Dynamic Execution of Distributed Systems* (SDEDS), to simulate the dynamic execution process of distributed systems with data execution on computer nodes and data transfer along network links before actual deployment and experimentation [141, 146]. We design and implement a generic *Scientific Workflow Automation and Management Platform* (SWAMP) and conduct experiments on two real-life scientific workflow applications [145, 150], i.e., analysis of measurement data from the Spallation Neutron Source (SNS) at Oak Ridge National Laboratory [2] and Climate Modeling (CM) at Brookhaven National Laboratory [5]. We conduct an extensive set of experiments on simulated and real-life workflows and networks, and both simulation and experimental results illustrate the performance superiority of our proposed algorithms over the existing ones.

1.4 Dissertation Organization

The rest of the dissertation is organized as follows:

- In Chapter 2, we conduct a comprehensive survey of related work on workflow mappings in the literature;

- In Chapter 3, we present analytical cost models of workflow and network components as well as two performance metrics, and formulate mapping optimization problems;
- In Chapter 4, we investigate the complexity of linear computing pipeline mappings, and provide rigorous NP-completeness proofs and design optimal or heuristic algorithms to solve these mapping problems;
- In Chapter 5, we tackle the mapping problems of general DAG-structured workflows and optimize their end-to-end performance under fault-tolerance constraints;
- In Chapter 6, we introduce the framework and design principles of SDEDS;
- In Chapter 7, we present the development of a workflow management system, SWAMP;
- In Chapter 8, we describe environmental settings and implementation details, and conduct extensive comparison-based performance evaluations;
- In Chapter 9, we conclude our work and discuss future research directions and efforts.

1.5 Main Contributions

This dissertation has made the following contributions to the fields of workflow mapping and scientific computing:

1. We consider a more realistic workflow model of dependent subtasks that better describes real-life scientific workflow applications;
2. We consider a more general network model of arbitrary topology with heterogeneous computer nodes and network links;
3. We consider a commonly used multi-port model where one node can receive, process and send data simultaneously;

4. We consider unknown time cost for module execution and data transfer until a mapping scheme is found;

5. We investigate resource sharing dynamics among concurrent module executions and data transfers;

6. We provide exhaustive problem categorization and complexity analysis for mapping problems as well as the non-approximability proofs;

7. We provide a rigorous stability analysis for streaming applications with time-series datasets being processed;

8. We investigate workflow optimization problems under fault-tolerance constraints;

9. We propose a set of efficient and effective optimal and heuristic mapping algorithms for both pipelined and DAG-structured workflows, and further decentralize the mapping process to facilitate the scalability of the mapping problems;

10. We conduct an extensive set of workflow mapping experiments in both simulated and wide-area networks with real-life workflow applications for model validation and performance valuation.

Chapter 2

Background Survey and Related Work

2.1 Computing Workflow Optimization

Many parallel and distributed applications in science and engineering fields face a major performance optimization problem of mapping or scheduling computing workflows with complex execution dependencies in distributed environments. This workflow mapping or scheduling problem has attracted a great deal of attention from researchers in various disciplines [26,29,31,33,38,44,47,48,56,75,92,105,116,117,123,128,129,135] and continues to be the focus of distributed computing due to both its theoretical significance and practical importance. There are two types of optimization problems for distributed computing tasks: one is to assign the component subtasks in a workflow to an appropriate set of computer nodes, which is referred to as workflow mapping, and the other is to decide the order and resource share (i.e., fair, proportional or priority-based, etc.) of independent subtask executions on a specific node or processor, which is referred to as task scheduling.

In the early years when shared network resources were still scarce, many research efforts were focused on workflow mapping on multiprocessors that are considered as identical/homogeneous resources [29, 75, 98]. Over the years, workflow mapping problems in heterogeneous environments have been increasingly identified and investigated by many researchers to facilitate a much larger scope of collaborations among different institutes

and domains across wide-area networks [37, 38, 58, 90, 117]. However, some of these efforts consider independent tasks in the workflows [41] or assume fully-connected computer networks, such as HEFT [128], which may not be sufficient to model the complexity of real applications.

Recently, with the pervasive deployment of networked resources based on grid infrastructures, a significant amount of efforts have been devoted to workflow mapping or task scheduling in grid environments under different mapping and resource constraints. The grid environments provide compositional programming and execution models to enable the resource interactions that support large-scale scientific applications by supplying the mechanisms to access, aggregate, and manage the network-based infrastructure of science [92]. Such grid-based scientific applications include data analysis for the Large Hadron Collider (LHC) supported by the Worldwide LHC Computing Grid (WLCG) [6], climate modeling supported by the Earth System Grid (ESG) [7], and NASA's National Airspace Simulation supported by the Virtual National Airspace Simulation (VNAS) grid [92, 108]. Other grid initiatives and projects such as Open Science Grid (OSG) [8], TeraGrid, ASKALON [135], Critical Path [116], GridFlow [44], Nimrod/G [42], Globus Toolkit [69, 70], Pegasus [57] and Condor/DAGMan [55, 71] (please see [9] for a more complete list) provide toolkits and middleware to deploy grid computing systems and several visualization applications were built using these services. Note that the execution model in the workflow mapping problem we consider differs from Condor/DAGMan: each module in our model only has one copy running on the mapping node, while Condor/DAGMan may dispatch multiple copies of each module to different nodes at the same time.

Several mapping-related studies are closely related to the pipeline mapping problems we consider. Particularly, in [37], Benoit *et al.* presented the mapping of computing pipelines onto different types of fully connected networks with identical processors and

links (fully homogeneous platforms), with identical links but different processors (communication homogeneous platforms), or with different processors and links (fully heterogeneous platforms). Their problems map each module or stage (including receiving, processing, and sending) to one node in a serial manner, and assume receiving, processing and sending should not be executed simultaneously which is defined as a one-port model. In [65], three local search heuristic algorithms are employed to solve the problem of finding a sequence of pipelined multiprocessor tasks on a processor and a proper mapping of tasks to the processors that are already being sequenced. Chatterjee *et al.* proposed a hierarchical end-to-end analysis technique that decomposes a very complex heterogeneous multi-resource scheduling problem into a set of single-resource scheduling problems with well-defined interactions [46]. Choi *et al.* proposed an approach to optimally configure sessions in programmable networks by converting the session configuration problem to the problem of finding a shortest path in a special layer-constructed graph [49]. The NP-completeness of this problem was further studied considering the capacity constraints of each link and node and a heuristic solution using extended Dijkstra's algorithm was proposed in [50].

Many other research efforts have been devoted to general DAG-structured workflow mappings. Foggia *et al.* introduced an adequate representation of the search space and process, and pruned unprofitable search paths in the search space to reduce the mapping complexity [66]. Ullmann [130] and Schmidt *et al.* [120] proposed a backtracking algorithm to reduce the size of the search space using different structures. VF2, a more recent algorithm based on a depth-first search strategy, was developed by Foggia *et al.* [52] to reduce the complexity. Note that most theoretical graph mapping problems consider exact mapping such that each module in the task graph is mapped to one node and each node in the network graph cannot be assigned more than one module.

In the past few years, many researchers have developed static scheduling algorithms[1] on multiprocessors that are considered as identical resources. Kwok *et al.* proposed Dynamic Critical-Path (DynCP) scheduling algorithm [98] to map task graphs with arbitrary computation and communication costs to a multiprocessor system with an unlimited number of identical processors in a fully-connected network. Based on DynCP, two CP and priority based algorithms were proposed by Ma *et al.* to schedule workflows with parameter-sweep tasks on global grids [105], where the application is optimized by pipelining the subtasks and dispatching each of them to well-selected resources. Another two algorithms based on DynCP were proposed by Rahman *et al.* for efficient mapping of tasks by calculating the CP in the workflow at each step and assigning a priority to each subtask on the CP to achieve an earlier completion [116], and McCreary *et al.* for determining the DynCP after each subtask is scheduled, allowing the partial schedule on a processor to change during the scheduling process [107]. A workflow mapping scheme for streaming data, Streamline, is designed in [26] to improve the performance of graph mapping for streaming applications with various demands in distributed environments, which places a coarse-grain dataflow graph on available grid resources. Most of these workflow mapping or task scheduling problems in grid environments assume complete networks with heterogeneous resources.

Similar mapping problems were also studied in the context of services and business process management workflows [100, 102, 119, 143], sensor networks [121] and bioinformatics workflows such as Taverna project [113]. Wu *et al.* tackled the problem of mapping workflow-structured web service compositions to heterogeneous environments and optimizing their performance in [143]. Saha *et al.* aimed to combine and compress a set of workflows such that computation can be minimized in [119]. Sekhar *et al.* proposed an optimal algorithm for mapping subtasks to a large number of sensor nodes based on an A^* algorithm in [121]. A greedy version of the A^* algorithm was proposed to reduce the complexity of the original optimal solution accounting for the limited energy of each sensor

[1]The scheduling decision in a static scheduling algorithm is made before the program or task executes and remains unchanged during execution.

node. A distributed data stream processing middleware, System *S*, was developed in [74], and a scheduler for resource allocation, SODA [136]. An optimization scheme for fusing compile-time operators into run-time software units, COLA [96], were proposed to support the *S* system. Note that the physical nodes they used to execute processing elements are in a tightly connected cluster environment (i.e., fully-connected), while we consider an arbitrary topology in a more loosely coupled network environment.

2.2 Classification of Workflow Mapping Algorithms

The workflow mapping algorithms can be categorized along different dimensions. For example, based on the time when the mapping is performed, these algorithms can be divided into two subclasses, i.e., static algorithms [44, 99], where the entire mapping scheme of a given workflow is produced before the actual workflow starts execution, and dynamic algorithms [71, 113], where the mapping of each module is determined on the fly at run-time. Also, based on the mapping methodology, these algorithms can be roughly classified into five categories [106, 128]: (i) Graph-based methods [52, 110]. Among the traditional graph mapping problems in theoretical aspects of computing, subgraph isomorphism is known to be NP-complete [73], while the complexity of graph isomorphism still remains open. Many special cases of the graph isomorphism problem under different topology constraints on the mapped (tasks or workflows) or mapping (network) graphs can be solved in polynomial time, including isomorphism between planar graphs [88] and bounded valence graphs [104]. (ii) List scheduling techniques, in which the most commonly used is critical path method [98, 105, 116]. (iii) Clustering algorithms [38, 75], which assume an unlimited number of processors and thus are not very feasible for practical use. (iv) Duplication-based algorithms [27, 40, 117], most of which are of a prohibitively high time complexity. (v) Guided random search methods such as genetic algorithm [29, 133] and

simulated annealing [124], where additional efforts are often required to determine an appropriate termination condition and usually there is no performance guarantee.

2.3 Existing Workflow Simulation Tools and Workflow Management Systems

The management of large-scale scientific workflows in heterogeneous computing environments is a crucial task that determines the overall application performance. There exist a large number of research efforts on the design and development of efficient and user-friendly workflow simulation tools or real-life workflow management systems. These tools or systems are often used to test or execute workflows in simulated or real network environments. Due to space limit, we are not able to provide an exhaustive enumeration of existing tools or systems. The following listing is not meant for competition as each workflow system, for either simulation or real execution, has its own design goals for evaluation, modeling, or management, and hence leads to different user experiences in terms of performance, convenience, flexibility and reliability.

2.3.1 Workflow Simulation Tools

There exist a number of simulation tools that are designed to study the behavior of distributed systems and investigate the performance of distributed algorithms. Such tools include ChicSim [118] and OptorSim [34] for studying data replication on grids, as well as PlanetSim [72] and PeerSim [91] for simulating P2P applications. We provide below a brief description of two widely adopted simulation tools, SimGrid and GridSim.

(1) SimGrid

SimGrid is a C language-based toolkit developed by H. Casanova, A. Legrand and M. Quinson *et al.* for simulating distributed applications on distributed platforms [45]. It is designed for the study of distributed systems and algorithms for large-scale platforms, using a low-level network simulation approach that approximates the behavior of TCP networks and decreases simulation costs by orders of magnitude when compared to packet-level simulation [132]. SimGrid has three key features [45]: (i) a scalable and extensible simulation engine that implements several validated simulation models; (ii) high-level user interfaces for geographically distributed researchers or users to quickly prototype simulations either in C or in Java; and (iii) Application Programming Interfaces (APIs) for advanced developers to design and implement distributed applications.

SimGrid also has some limitations: it is restricted to a single scheduling entity and time-shared systems, which makes it difficult to simulate multiple user requirements, applications, and schedulers, each with its own policies. Moreover, many network resources in Grid environments are space-shared machines and need to be supported during actual workflow execution.

(2) GridSim

GridSim is a Java-based discrete-event Grid simulation toolkit developed by R. Buyya *et al.* It was initially intended for grid economy, and later became used for general distributed resource management and scheduling in Grid environments [43]. It supports primitives for application composition, information services for resource discovery, and interfaces for assigning application tasks to resources and managing their execution, which could be used to model and simulate Grid schedulers for performance evaluation of scheduling algorithms or heuristics on various classes of parallel and distributed computing systems such as clusters, Grids, P2P networks, etc. GridSim toolkit extends the ideas in existing systems and overcomes the aforementioned limitations in SimGrid.

The developers mentioned several future efforts in their work [43]: (i) Develop a better network model to support the application model with tasks collaborating and exchanging partial results among themselves in a P2P fashion. (ii) Strengthen the network model by supporting various types of networks with different static and dynamic configurations and cost-based quality of services. (iii) Enhance the resource models by interfacing with off-the-shelf storage I/O simulators.

2.3.2 Workflow Management Systems

We provide a brief overview of some widely used workflow management systems implemented and deployed in real network environments.

(1) Condor/DAGMan/Stork

The Condor project [10], a specialized workload management system for compute-intensive jobs, began in 1988 under the lead of Prof. M. Livny. Condor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Condor places serial or parallel jobs into a queue after users submit them, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the users upon completion. Condor has expanded its focus from executing jobs on local clusters of computers to running jobs in distributed grid environments (i.e., Globus, NorduGrid, Oracle, etc.), which is named Condor-G.

In the late 1990s, the Directed Acyclic Graph Manager (DAGMan) was built on Condor to allow users to submit large workflows involving many jobs with execution dependency to Condor, and automate the submission and management of complex workflows with a focus on reliability and fault tolerance [53].

Note that Condor/DAGMan has a centralized execution model where the output of each module is sent back to the submitter for forwarding to its succeeding modules. This centralized model may introduce prohibitively large data traffics in the network, especially for

data-intensive workflows with many computing modules. To accommodate distributed resource allocation and storage, Stork, a scheduler for data placement activities in the Grid was developed to facilitate the queuing, scheduling and optimization of data placement jobs, and to provide a level of abstraction between user applications and underlying data transfer and storage resources [97].

With a combination of Condor, DAGMan and Stork, users can create, submit, execute and monitor large-scale complex workflows in a distributed manner in a grid environment.

(2) Pegasus

Pegasus, which stands for Planning for Execution in Grids, developed by E. Deelman *et al.*, takes an abstract description of a workflow and finds appropriate data and Grid resources to execute the workflow [54, 57]. Users can use Chimera to construct abstract workflows [68] or write the code by themselves. Chimera takes the partial workflow descriptions as inputs which describe the logical input files, the logical transformations and their parameters, as well as the logical output files produced by these transformations. The current system is semi-dynamic since the workflows are fully mapped to their concrete form when they are submitted to Pegasus. The fully dynamic mode of mapping workflows can be implemented by combining the technology of Pegasus and Condor's workflow executer, DAGMan. Pegasus enables scientists to focus on their domain research and design workflows at the application level without the need of considering the actual execution environments.

(3) Kepler

Kepler is a java-based application developed and maintained by the cross-project collaboration, who is designed to help domain scientists, analysts, and computer programmers to easily create, execute, and share models and analyses across a broad range of scientific and engineering disciplines [11, 103].

Kepler provides a Graphical User Interface (GUI) and a run-time engine that can execute workflows either in GUI or from a command line. Kepler Manager is designed as an actor that resides in the Kepler framework. A constant string actor in Kepler is used to represent an abstract module in the workflow. The name of an executable module corresponding to an abstract module is set in the "value" property in each constant string actor. Kepler saves the abstract workflow in an XML file, and the Kepler Manager takes this abstract workflow in XML format as input and generates the DAG and submit files, which are then transferred to the Web Server Manager.

Using Kepler's GUI and processing components, scientists with little background in computer science can create executable scientific workflows for accessing scientific data and running analysis tools on the retrieved data.

(4) ASKALON

ASKALON, developed by the group of Prof. T. Fahringer at the University of Innsbruck in Austria, is a grid application development and computing environment, whose ultimate goal is to hide the low-level grid environment from application users [127]. In ASKALON, users can compose grid workflow applications using a graphical workflow composition and modeling service based on UML, or programmatically describe workflows using XML-based Abstract Grid Workflow Language (AGWL) at a higher level of abstraction. The AGWL representation of a workflow is submitted to the ASKALON Web Services Resource Framework (WSRF) specification for scheduling and execution in grid environments.

(5) Taverna/*my*Grid

*my*Grid is a project funded by the United Kingdom's e-Science Programme since 2001 to build middleware to support workflow-based *in silico* experiments in biology [113, 127]. *my*Grid's workflow execution and development environment, Taverna, links and executes all

kinds of heterogeneous open services. Taverna is an application that eases the use and integration of molecular biology tools and databases available on the web, especially web services. It allows bioinformaticians to construct workflows or pipelines of services through GUI to perform a range of different analyses, such as sequence analysis and genome annotation [89]. *my*Grid components are Taverna plug-ins and services.

(6) Sedna

Sedna is a Business Process Execution Language (BPEL)-based environment for visual scientific workflow modeling [127], which provides language abstractions in addition to those used in BPEL in order to simplify the modeling of scientific workflows and hide the complexity of underlying technologies and middleware.

(7) SWAMP

We designed and developed a Condor/DAGMan-based generic Scientific Workflow Automation and Management Platform (SWAMP), which contains a set of easy-to-use computing and networking toolkits for application users to conveniently assemble, execute, monitor, and control workflows in distributed network environments [145, 150].

SWAMP provides a web-based user interface for users to remotely configure, dispatch, monitor and control workflows. It also features a special workflow mapper as an optimization engine, which automatically maps abstract workflows to real networks to achieve optimal end-to-end performance based on real-time network status measurements. We conducted an extensive set of performance evaluation and comparison in SWAMP using both simulated workflow and network data as well as real-life ones. The design and implementation details are provide in Chapter 7.

Chapter 3

Mathematical Models and Problem Formulation

In this chapter, we construct cost models for computing modules and dependency edges in a general workflow, and computer nodes and network links in a computer network to facilitate a mathematical formulation of scientific workflow mapping problems.

3.1 Cost Models of Computing Workflows and Computer Networks

We model the workflow as a Directed Acyclic Graph (DAG) $G_w = (V_w, E_w)$, $|V_w| = m$, where vertices represent computing modules starting from module w_0 and ending at module w_{m-1}. The dependency between a pair of adjacent modules w_i and w_j is represented by a directed edge $e_{i,j} \in E_w$ between them. Module w_j receives a data input from each of its preceding modules and performs a predefined computing routine whose complexity is modeled as a function $\lambda_{w_j}(\cdot)$ on the aggregate input data size z_{w_j}. The complexity of a module is an abstract quantity that not only depends on the computational complexity of the algorithm defined in the module but also the implementation details such as the specific data structures used in the program. To make the complexity function general, we do not specify the data aggregation method in our model, which, in a real application, could be

simply replaced by a specific aggregation function determined by the particular algorithm implemented in that computing module. For example, we perform a simple summation operation for data aggregation in our later simulated workflows. Once the routine is executed successfully in its entirety, module w_j sends a different data output to each of its succeeding modules. We assume that a module cannot start its execution until all input data required by this module arrive. For a workflow with multiple source or destination modules, we could convert it to this model by inserting a virtual start or end module of complexity zero connected to all source or destination modules with zero-sized output or input data transfers.

We model the computer network as an arbitrary weighted graph $G_c = (V_c, E_c)$, consisting of $|V_c| = n$ nodes interconnected by $|E_c|$ overlay links. We use a normalized variable p_i to represent the overall processing power of node v_i without specifying its detailed system resources, which combines a variety of host factors such as processor frequency, bus speed, memory size, I/O performance, presence of co-processors etc. A statistical approach to performance modeling and prediction is detailed in [137]. The link $l_{i,j}$ between nodes v_i and v_j has Bandwidth (BW) $b_{i,j}$ and Minimum Link Delay (MLD) $d_{i,j}$. We specify a pair of source and destination nodes (v_s, v_d) to run the start module w_0 and the end module w_{m-1}, respectively, and further assume that module w_0 serves as a data source without any computation to supply all initial data needed by the application and module w_{m-1} performs a terminal task (e.g., display) without any further data transfer. This is based on the consideration that the system knows where the raw data is stored and where an end user is located before a workflow executes in an existing network.

The computational complexity of a module together with the incoming data size determines the number of floating operations (flops) to complete the computing routine defined in that module. The actual module execution time also depends on the capacity of system resources, i.e., overall processing power, deployed on the selected network node as well as their availability during runtime.

We further introduce the following notations for module/edge assignment and time cost of module execution and data transfer.

- $x_{wv}, \forall w \in V_w, v \in V_c$: if module w is mapped to node v, $x_{wv} = 1$; otherwise, $x_{wv} = 0$.
- $y_{el}, \forall e \in E_w, l \in E_c$: if edge e is mapped to link l, $y_{el} = 1$; otherwise, $y_{el} = 0$.
- t^s_w, t^f_w, $\forall w \in V_w$: execution start and finish times of module w.
- $t^s_{e_{i,j}}$, $t^f_{e_{i,j}}$, $\forall e_{i,j} \in E_w$: transfer start and finish times of dependency edge $e_{i,j}$ from module w_i to w_j.

We consider the following conditions on workflow mapping and execution dynamics.

1. Single-node and single-link mapping:

   $\sum_{v \in V_c} x_{wv} = 1, \ \forall \ w \in V_w$, and $\sum_{l \in E_c} y_{el} = 1, \ \forall \ e \in E_w$.

2. Module execution precedence: $t^s_{w_j} \geq t^f_{e_{i,j}}, \ \forall \ w_j \in V_w, \ e_{i,j} \in E_w$.

3. Data transfer precedence: $t^s_{e_{i,j}} \geq t^f_{w_i}, \ \forall \ w_i \in V_w, \ e_{i,j} \in E_w$.

Condition 1 requires each module/edge to be mapped to only one node/link. Condition 2 ensures that a computing module cannot start execution until all its required input data arrive, and Condition 3 ensures that a dependency edge cannot start data transfer until its preceding module finishes execution.

We use $T_{\text{exec}}(w_i, v_j)$ to represent the execution time of module w_i on node v_j, which is computed as the module's total computation requirements divided by the node processing power:

$$T_{\text{exec}}(w_i, v_j) = \sum_{t=t^s_{w_i}}^{t^f_{w_i}} \frac{\alpha_i(t) \cdot \delta_{w_i}(t)}{p_j}, \ \forall \ w_i \in V_w, \ v_j \in V_c, \tag{3.1.1}$$

where $\alpha_i(t) = \sum_{w_i \in V_w : (t^f_{w_i} - t)(t - t^s_{w_i}) \geq 0} x_{w_i v_j}$. The amount of partial module execution completed during time interval $[t, \, t + \Delta t]$ when $\alpha_i(t)$ remains unchanged is denoted by $\delta_{w_i}(t) = \frac{p_j}{\alpha_i(t)} \Delta t$, and $\lambda_{w_i}(z_{w_i}) = \sum_{t=t^s_{w_i}}^{t^f_{w_i}} \delta_{w_i}(t)$ is the total computational requirement of computing module w_i.

Similarly, we use $T_{\text{tran}}(e_{i,j}, l_{h,k})$ to represent the data transfer time of dependency edge $e_{i,j}$ over network link $l_{h,k}$, which is computed as the transferred data size divided by the bandwidth plus MLD:

$$T_{\text{tran}}(e_{i,j}, l_{h,k}) = \sum_{t=t^s_{e_{i,j}}}^{t^f_{e_{i,j}}} \frac{\beta_{i,j}(t) \cdot \delta_{e_{i,j}}(t)}{b_{h,k}} + d_{h,k}, \ \forall\, e_{i,j} \in E_w,\ l_{h,k} \in E_c, \tag{3.1.2}$$

where $\beta_{i,j}(t) = \sum_{e_{i,j} \in E_w : (t^f_{e_{i,j}} - t)(t - t^s_{e_{i,j}}) \geq 0} y_{e_{i,j} l_{h,k}}$. The amount of partial data transfer completed during time interval $[t,\ t + \Delta t]$ when $\beta_{i,j}(t)$ remains unchanged is denoted by $\delta_{e_{i,j}}(t) = \frac{b_{h,k}}{\beta_{i,j}(t)} \Delta t$, and $z_{e_{i,j}} = \sum_{t=t^s_{e_{i,j}}}^{t^f_{e_{i,j}}} \delta_{e_{i,j}}(t)$ is the total data transfer size of dependency edge $e_{i,j}$.

Note that $\alpha_i(t)$ in Eq. 3.1.1 denotes the constant number of concurrent modules for module w_i on node v_j during time interval $[t,\ t + \Delta t]$, and $\beta_{i,j}(t)$ denotes the number of concurrent data transfers for data $z_{i,j}$ transferred over network link $l_{h,k}$ during time interval $[t,\ t + \Delta t]$. For MED of unitary processing applications, in the case of no node reuse (i.e., one-to-one mapping), $\alpha(t) \equiv 1$ and $\beta(t) \equiv 1$, and in the case of arbitrary node reuse, we have $\alpha(t) \geq 1$ and $\beta(t) \geq 1$ when CPU cycles or link bandwidths are shared by concurrent independent modules or data transfers; while for MFR of streaming applications, even in the case of only dependent modules mapped on the same node, we still have $\alpha(t) \geq 1$ and $\beta(t) \geq 1$ because computing and networking resources could be shared by multiple dependent modules and edges that are processing and transferring different instances of input datasets. Note that in the special case of DAG-structured workflows, i.e., linear pipelines, $\alpha(t) \geq 1$ and $\beta(t) \geq 1$ only happens in the contiguous/arbitrary node reuse in streaming applications. Due to the dynamics in concurrent workload on nodes and concurrent traffic over links, both $\alpha(t)$ and $\beta(t)$ are time-varying in nature and their distributions generally do not exist in a continuous form, which renders standard mathematical solvers inapplicable. We develop an efficient algorithm in Chapter 5.2 for exact delay calculation by determining the values of $\alpha(t)$ and $\beta(t)$ at discrete times and incorporate it into the proposed workflow mapping algorithms. We tabulate the parameters defined in the cost models in Table 3.1.

Table 3.1: Parameters in the cost models and problem formulation.

Parameters	Definitions
$G_w = (V_w, E_w)$	computing workflow
m	the number of modules in the workflow
w_i	the i-th computing module
$e_{i,j}$	dependency edge from module w_i to w_j
$z_{i,j}$	transferred data size of dependency edge $e_{i,j}$
z_{w_i}	aggregated input data size of module w_i
$\lambda_{w_i}(\cdot)$	computational complexity of module w_i
$G_c = (V_c, E_c)$	computer network
n	the number of nodes in the network
v_i	the i-th network or computer node
v_s	source node
v_d	destination node
p_i	processing power of node v_i
$l_{i,j}$	network link between nodes v_i and v_j
$b_{i,j}$	bandwidth of link $l_{i,j}$
$d_{i,j}$	minimum link delay of link $l_{i,j}$
x_{wv}	binary indicator if module w is mapped to node v
y_{el}	binary indicator if edge e is mapped to link l
t_w^s, t_w^f	execution start and finish times of module w
$t_{e_{i,j}}^s$, $t_{e_{i,j}}^f$	transfer start and finish times of edge $e_{i,j}$
$T_{\text{exec}}(w,v)$	execution time of module w on node v
$\alpha(t)$	number of concurrent modules on node v during Δt
$\delta_w(t)$	amount of partial module execution completed during Δt
$T_{\text{tran}}(e,l)$	data transfer time of edge e over link l
$\beta(t)$	number of concurrent data transfers over link l during Δt
$\delta_e(t)$	amount of partial data transfer completed during Δt

3.2 Performance Metrics

3.2.1 End-to-end Delay

End-to-end Delay (ED), or latency, is the total completion time of the entire workflow from the time when the original input dataset is fed into the first module to the time when the final result is generated at the last module. This is an important performance metric in time-critical applications, especially for unitary processing applications.

Once a mapping scheme is determined, ED is calculated as the total time cost incurred on the CP, i.e., the longest path in the DAG. We denote the set of contiguous modules on the CP that are allocated to the same node as a "group", and refer to those modules located on the CP as "critical" modules and others as "branch" or "non-critical" modules. A general mapping scheme divides the CP into q ($1 \leq q \leq m$) contiguous groups g_i, $i = 0, 1, \cdots, q-1$ of critical modules and maps them to a network path P of not necessarily distinct q nodes, $v_{P[0]}, v_{P[1]}, \cdots, v_{P[q-1]}$ from source $v_s = v_{P[0]}$ to destination $v_d = v_{P[q-1]}$. The ED of a mapped workflow is calculated as:

$$T_{\text{ED}}\left(\text{CP mapped to a path } P \text{ of } q \text{ nodes}\right) = T_{\text{exec}} + T_{\text{tran}} = \sum_{i=0}^{q-1} T_{g_i} + \sum_{i=0}^{q-2} T_{e(g_i,g_{i+1})}$$
$$= \sum_{i=0}^{q-1} \left(\sum_{j \in g_i, j \geq 1} \left(\sum_{t=t^s_{w_j}}^{t^f_{w_j}} \frac{\alpha_j(t) \cdot \delta_{w_j}(t)}{p_{P[i]}} \right) \right) + \sum_{i=0}^{q-2} \left(\sum_{t=t^s_{e(g_i,g_{i+1})}}^{t^f_{e(g_i,g_{i+1})}} \frac{\beta_{e(g_i,g_{i+1})}(t) \cdot \delta_{e(g_i,g_{i+1})}(t)}{b_{P[i],P[i+1]}} + d_{P[i],P[i+1]} \right), \tag{3.2.1}$$

where $e(g_i, g_{i+1})$ denotes the dependency edge from group g_i to g_{i+1} mapped to link $l_{P[i],P[i+1]}$ between nodes $v_{P[i]}$ and $v_{P[i+1]}$.

3.2.2 Frame Rate

Frame Rate (FR) or throughput, i.e., the reciprocal of the global Bottleneck Time (BT) of the workflow, is the data production rate at the last module which is the most critical performance metric for streaming applications that process multiple (e.g., time-series) datasets. The bottleneck time T_{BT} is the longest time unit among all module execution and data transfer times, which could be either on a computing module or a dependency edge, defined as:

$$T_{\text{BT}}(G_w \text{ mapped to } G_c) = \max_{\substack{w_i \in V_w, e_{j,k} \in E_w \\ v_{i'} \in V_c, l_{j',k'} \in E_c}} \begin{pmatrix} T_{\text{exec}}(w_i, v_{i'}), \\ T_{\text{tran}}(e_{j,k}, l_{j',k'}) \end{pmatrix}$$
$$= \max_{\substack{w_i \in V_w, e_{j,k} \in E_w \\ v_{i'} \in V_c, l_{j',k'} \in E_c}} \begin{pmatrix} \sum_{t=t^s_{w_i}}^{t^f_{w_i}} \frac{\alpha_i(t) \cdot \delta_{w_i}(t)}{p_{i'}}, \\ \sum_{t=t^s_{e_{j,k}}}^{t^f_{e_{j,k}}} \frac{\beta_{j,k}(t) \cdot \delta_{e_{j,k}}(t)}{b_{j',k'}} + d_{j',k'} \end{pmatrix}. \tag{3.2.2}$$

We assume that the inter-module communication cost on the same node is negligible. Since the execution start time of a module depends on the availability of all its input datasets, the modules assigned to the same node may not run simultaneously. The same is also true for concurrent data transfers over the same network link.

Chapter 4

Linear Pipeline Optimization

In this chapter, we formulate and categorize the pipeline mapping problems into six classes with two optimization objectives, i.e., Minimum End-to-end Delay (MED) and Maximum Frame Rate (MFR), and three mapping constraints, i.e., No Node Reuse (NNR), Contiguous Node Reuse (CNR) and Arbitrary Node Reuse (ANR). We propose a set of mapping algorithms, *Efficient Linear Pipeline Configuration* (ELPC), in which a polynomial-time optimal solution based on Dynamic Programming (DP) is designed for MED-ANR and heuristics are designed for the rest five problems, which are proved to be NP-complete.

4.1 Problem Categorization and Complexity Analysis

4.1.1 Problem Categorization

For both MED and MFR, we consider three different types of mapping constraints as shown in Fig. 4.1: (i) NNR, where a node on the selected network path P executes exactly one module; (ii) CNR, where multiple contiguous modules along the pipeline are allowed to run on the same node; and (iii) ANR, where multiple modules, either contiguous or non-contiguous, are allowed to run on the same node. We category the pipeline mapping problems into six constrained versions and further investigate their complexities, i.e., MED/MFR-NNR, MED/MFR-CNR, and MED/MFR-ANR. These mapping problems and their complexity analysis are tabulated in Fig. 4.2.

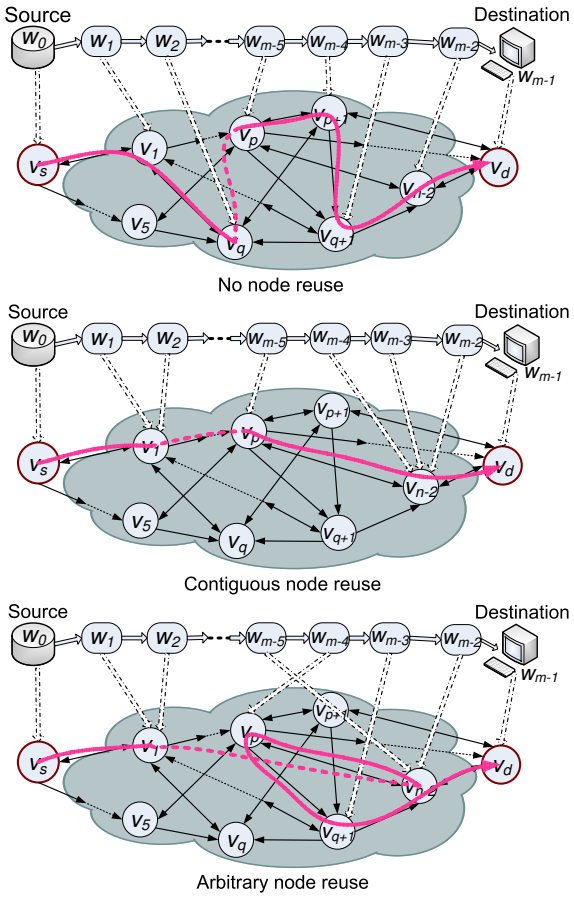


Figure 4.1: Three network constraints on pipeline mappings.

We show that MED/MFR-NNR/CNR are NP-complete by reducing to them from the problem of finding two vertex-disjoint paths in a directed graph, which also allows us to prove that these problems cannot be approximated by any constant factor, unless $P = NP$ [81]. For ANR, the difficulty of finding disjoint paths does not occur any more since the mapping scheme can reuse some computer nodes that are mapped by previous modules. Indeed, MED-ANR is close to the shortest-path problem and we are able to develop a polynomial-time optimal solution to this problem using a dynamic programming-based procedure. Resource sharing adds an additional level of complicacy to MFR-ANR, whose NP-completeness is proved by reducing to it from the Widest path with Linear Capacity Constraints (WLCC) problem [151].

Objective Function / Constraints	Minimum End-to-end Delay	Maximum Frame Rate
No Node Reuse	NP-complete	NP-complete
Contiguous Node Reuse	NP-complete	NP-complete
Arbitrary Node Reuse	Polynomial (Dyn. Prog.)	NP-complete

Figure 4.2: Complexity analysis of six mapping problems.

4.1.2 NP-completeness Proof for MED/MFR-NNR/CNR

We first define pipeline mapping with MED/MFR-NNR/CNR as decision problems:

Definition 1. *Given a linear computing pipeline with m modules, a directed weighted computer network G and a bound T, does there exist a mapping scheme that maps the pipeline to the network under the constraint of NNR or CNR, such that the ED or BT does not exceed T?*

Note that here we consider BT, the reciprocal of FR, to make the problem definitions uniform among these four problems.

Theorem 1. *MED/MFR-NNR/CNR are NP-complete.*

Proof. We use a reduction from DISJOINT-CONNECTING-PATH (DCP) [73], which is NP-complete even when restricting to two paths in the case of directed graphs (2DCP) [67]. The problems clearly belong to NP: given a division of the pipeline into q groups and a path P of q nodes, we can compute the ED or BT in polynomial time using Eqs. 3.2.1 or 3.2.2, and check if the bound T is satisfied. We prove their NP-hardness by showing that 2DCP $\leq_p$ MED/MFR-NNR/CNR.

Consider an arbitrary instance $\mathscr{I}_1$ of 2DCP, i.e., a network graph $G=(V,E)$, $n=|V|\geq 4$ and two disjoint vertex pairs (x_1,y_1), $(x_2,y_2)\in V^2$. We ask if G contains two mutually vertex-disjoint paths, one going from x_1 to y_1, and the other going from x_2 to y_2. We can create the following instance $\mathscr{I}_2$ of our mapping problems (MED/MFR-NNR/CNR). The pipeline consists of $m=2n+1$ modules and the computational complexity of each module

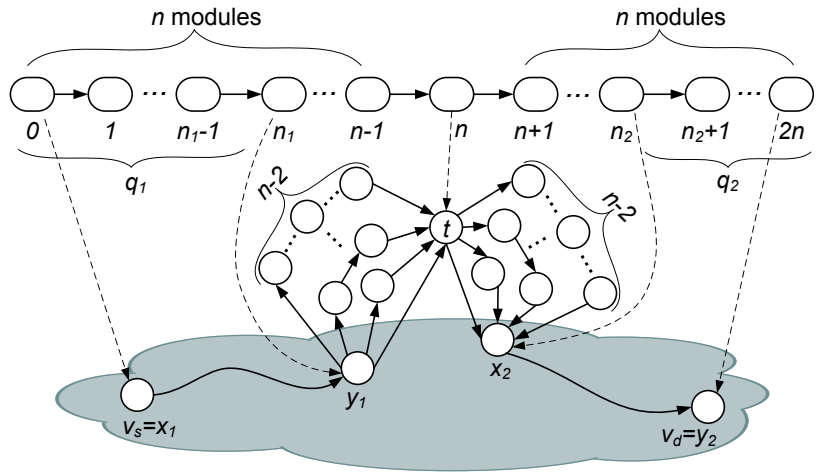


Figure 4.3: Reduction from 2DCP problem.

w_i is $\lambda_{w_i}(z) = 1$ for $0 \leq i \leq 2n, i \neq n$ and $\lambda_{w_n}(z) = n^2$. In other words, only module w_n has a much higher complexity than all other modules. The network consists of $|V| + 1 + (n-1)(n-2)$ nodes and $|E| + n(n-1)$ links, and is constructed as follows: starting from graph G, we add a new node t, which is connected to G with an incoming link from y_1 to t, and an outgoing link from t to x_2. We also add $0 + 1 + 2 + \cdots + (n-2)$ additional nodes between y_1 and t and their corresponding links to connect y_1 to t through a set of $n-1$ paths of length l (in terms of the number of nodes excluding two end nodes y_1 and t), $0 \leq l \leq n-2$, as shown in Fig. 4.3.

Similarly, additional nodes and links are added between t and x_2 so that there exist a set of $n-1$ paths of length l, $0 \leq l \leq n-2$, between nodes t and x_2. The processing capability of each node $v_j \in V'$ in the new graph $G' = (V', E')$ is set to $p_j = 1$ except for node t, whose computing power is set to be n^2, which is much higher than all other nodes. All link bandwidths are set to be sufficiently high so that transfer time between nodes is ignored compared to computing time. The source and destination nodes v_s and v_d are set to be x_1 and y_2, respectively. We ask if we can achieve an ED that does not exceed $T_{\text{MED}} = 2n + 1$ or BT that does not exceed $T_{\text{MFR}} = n$. Obviously, this instance transformation can be done in polynomial time.

We show that given a solution to $\mathscr{I}_1$, we can find a solution to $\mathscr{I}_2$ of four mapping problems. Let q_i be the length of the path from x_i to y_i (including two end nodes), $i = 1,2$. We have $2 \leq q_i \leq n$ for both paths, and paths are vertex-disjoint. The mapping solutions are derived in two cases under different mapping constraints, i.e., NNR and CNR, as follows:

(1) Steps to derive mapping solutions for NNR

1. We map the first q_1 modules starting from module w_0 one-to-one onto q_1 nodes of the path from x_1 to y_1 with NNR. Thus, w_0 is mapped on the source node $x_1 = v_s$ and w_{q_1-1} is mapped on node y_1.

2. We map the next $n - q_1$ modules from w_{q_1} to w_{n-1} along the pipeline one-to-one onto a path between y_1 and t (excluding two end nodes y_1 and t) of length $n-q_1$ with NNR. There must exist such a mapping path since $0 \leq n - q_1 \leq n-2$. Obviously, each of the first n modules from w_0 to w_{n-1} incurs a delay of 1.

3. We map module w_n with a complexity of n^2 to the fast node t, which also incurs a delay of $n^2/n^2 = 1$.

4. Similar to the first n modules, we map the last n modules from w_{n+1} to w_{2n} one-to-one onto the path of length $n - q_2$ from t to x_2 (excluding two end nodes) and then on the path of length q_2 from x_2 to y_2 (including two end nodes) with NNR. Each of the last n modules also incurs a delay of 1.

(2) Steps to derive mapping solutions for CNR

1. Same as Step 1 in the case of NNR, we map the first q_1 modules starting from module w_0 one-to-one onto q_1 nodes of the path from x_1 to y_1 with NNR, where each module incurs a delay of 1 (excluding module w_{q_1-1}).

2. We map all of the next $n - q_1$ modules from w_{q_1} to w_{n-1} along the pipeline to node y_1, together with module w_{q_1-1}, where each module incurs a delay of $\frac{1}{(1/(n-q_1+1))} = n - q_1 + 1$ due to resource sharing.

3. Same as Step 3 in the case of NNR, we also map module w_n with a complexity of n^2 to the fastest node t, which incurs a delay of $n^2/n^2 = 1$.

4. Similar to the first n modules, we map all of $n - q_2$ modules in the last n modules starting from w_{n+1} to node x_2, and then map the last q_2 modules one-to-one onto the path of length q_2 from x_2 to y_2 (including two end nodes) with NNR. Each module on node x_2 incurs a delay of $\frac{1}{(1/(n-q_2+1))} = n - q_2 + 1$ due to resource sharing and each of the modules mapped on the path from x_2 to y_2 (excluding those mapped on x_2) incurs a delay of 1.

Obviously, the above derived mapping solutions meet the requirements of NNR/CNR since we use the solution to $\mathscr{I}_1$: two paths from x_1 to y_1 and from x_2 to y_2 are disjoint, so either no node is reused or only contiguous modules are mapped to the same node (either y_1 or x_2). For MED problems, since each module incurs a delay of 1 with no resource sharing and there are $2n+1$ modules in total, the ED of this mapping solution is $2n+1 \leq T_{\text{MED}}$ in both NNR and CNR; while for MFR problems: (i) in the case of NNR, since all modules have an identical delay of 1, the BT of the entire pipeline is $1 \leq n = T_{\text{MFR}}$; (ii) in the case of CNR, all modules incur an identical delay of 1 except those mapped on nodes y_1 and x_2, where each module on node y_1 has a delay of $n - q_1 + 1$ and each module on node x_2 has a delay of $n - q_2 + 1$ because of resource sharing. The BT of the entire pipeline is either $n - q_1 + 1 \leq n - 2 + 1 < n = T_{\text{MFR}}$ if $q_1 \leq q_2$, or $n - q_2 + 1 \leq n - 2 + 1 < n = T_{\text{MFR}}$ if $q_1 > q_2$. Therefore, we find a valid solution to $\mathscr{I}_2$ of all four mapping problems.

Reciprocally, if $\mathscr{I}_2$ has a solution, we show that $\mathscr{I}_1$ also has a solution. We prove that the mapping of $\mathscr{I}_2$ has to be of a similar form as the mapping described above (or similar but with CNR in some instances of the problem), and thus that there exist disjoint paths $x_1 \to y_1$ and $x_2 \to y_2$. This property comes from the fact that node t must be used in the mapping. Indeed, if node t is not used to process module w_n, this module will incur a delay of n^2, thus the ED will be at least $n^2 + 4$ (each of 4 nodes, i.e., x_1, y_1, x_2, and y_2, incurs a delay of 1), which is larger than the total delay bound $T_{\text{MED}} = 2n + 1$, and also becomes

the bottleneck with cost larger than the bound $T_{\text{MFR}} = n$. Since the ED of $\mathscr{I}_2$ is less than $2n+1$, or the BT is less than or equal to n, module w_n must be mapped on node t in the solution to $\mathscr{I}_2$. Note that the only way to reach node t involves using one of the $n-1$ paths going from y_1 to t. Thus, there is at least one module mapped on y_1. Let w_{n_1} be the last of these modules: $n_1 < n$, and w_{n_1+1} is not mapped on y_1. Similarly, all paths departing from t go through x_2, thus there is at least one module mapped on x_2. Let w_{n_2} be the first of these modules: $n_2 > n$, and w_{n_2-1} is not mapped on x_2. Moreover, source and destination nodes x_1 and y_2 are also used, since w_0 is mapped on x_1 and w_{2n} is mapped on y_2. Therefore, the entire mapping scheme is made up of the following three segments: (i) modules w_0 to w_{n_1} are mapped on a path between x_1 and y_1 (including two end nodes); (ii) modules w_{n_1+1} to w_{n_2-1} are mapped on a path between y_1 and x_2 (excluding two end nodes) going through t; and (iii) modules w_{n_2} to w_{2n} are mapped on a path between x_2 and y_2 (including two end nodes). Since only contiguous modules along the pipeline can be deployed on the same node, or NNR is performed at all in this mapping, nodes in both paths $x_1 \to y_1$ and $x_2 \to y_2$ should be distinct, and they are connected only by edges in G according to the construction of $\mathscr{I}_2$. Thus, we find two disjoint paths, which constitute a solution to $\mathscr{I}_1$. □

Theorem 2. *Given any constant $\varepsilon > 0$, there exists no ε-approximation to the MED/MFR-NNR/CNR problems, unless $P = NP$.*

Proof. Given ε, assume that there exists a ε-approximation to one of the four problems. Let $\mathscr{I}_1$ be an instance of 2DCP (see proof of Theorem 1). We build the same instance $\mathscr{I}_2$ as in the previous proof, except for the speed of the fast node t and the computational complexity of module w_n, both of which are set to be εn^2 instead of n^2.

We use the ε-approximation algorithm to solve this instance $\mathscr{I}_2$ of our problem, which returns a mapping scheme of ED or BT T_{alg} such that $T_{\text{alg}} \leq \varepsilon T_{\text{opt}}$, where T_{opt} is the optimal ED or BT. Then we prove that we can solve 2DCP in polynomial time. We need to differentiate the cases of ED and FR (the reciprocal of BT) as follows:

(1) MED problems

1. If $T_{\text{alg}} > \varepsilon(2n+1)$, then $T_{\text{opt}} > 2n+1$ and there does not exist two disjoint paths; otherwise, we could achieve a mapping of ED equal to $2n+1$. In this case, 2DCP has no solution.

2. If $T_{\text{alg}} \leq \varepsilon(2n+1)$, we must map w_n on t; otherwise, it would incur a delay of $\varepsilon n^2 > \varepsilon(2n+1) \geq T_{\text{alg}}$, which conflicts with the condition. Hence, the mapping is similar to the one described in the proof of Theorem 1, and we conclude that 2DCP has a solution.

(2) MFR problems

1. If $T_{\text{alg}} > \varepsilon n$, then $T_{\text{opt}} > n$ and there does not exist two disjoint paths; otherwise, we could achieve a mapping of frame rate equal to n. In this case, 2DCP has no solution.

2. If $T_{\text{alg}} \leq \varepsilon n$, we must map w_n on t; otherwise, it would incur a delay of $\varepsilon n^2 > \varepsilon n \geq T_{\text{alg}}$, which conflicts with the condition. Hence, the mapping is similar to the one described in the proof of Theorem 1. We conclude that 2DCP has a solution.

Therefore, in both cases, if 2DCP has a solution in polynomial time, then $P = NP$, which establishes the contradiction and proves the non-approximability result. □

Because of the arbitrary node reuse, the non-approximability result is not directly applicable to MFR-ANR. We conjecture that MFR-ANR can be approximated, for instance by modifying classical bin-packing approximation schemes [30].

4.1.3 NP-completeness Proof for MFR-ANR

The NP-completeness proof for MFR-ANR is based on the Widest path with Linear Capacity Constraints (WLCC) problem, which is shown to be NP-complete in [151]. An LCC-graph is a three-tuple $(G = (V,E),C,b)$, where the capacity of each link $e \in E$ is a

variable x_e, and (C,b) represent a set of m linear capacity constraints $Cx \leq b$, C is a 0-1 coefficient matrix of size $m \times |E|$, x is a $|E| \times 1$ vector of link capacity variables, and $b \in R^m$ is a capacity vector. Each link $e \in E$ has a capacity $c(e) \geq 0$. Given an LCC-graph (G,C,b), the width $\omega(P)$ of a path $P = (e_1, e_2, \cdots, e_k) \subset G$ is defined as the bottleneck capacity x_{BN} subject to $x_{e_j} = 0$, $\forall e_j \notin P$, $Cx \leq b$, and $x_{e_1} = x_{e_2} = \cdots = x_{e_k} = x_{\text{BN}}$. We define WLCC as a decision problem: Given an arbitrary instance $(G = (V,E), C, b)$ of WLCC, two nodes $v_s, v_d \in V$ and a positive integer $K \leq Max\{b_i\}$, does there exist a path P from v_s to v_d whose width, i.e., bottleneck capacity x_{BN}, is no less than K?

We first define pipeline mapping with MFR-ANR as a decision problem:

Definition 2. *Given a linear computing pipeline with m modules, a computer network G and a bound B, does there exist a mapping scheme that maps the pipeline to the network under the constraint of ANR, such that the FR (the reciprocal of the BT) is no less than B?*

Theorem 3. *MFR-ANR is NP-complete.*

Proof. For a given solution to an instance of MFR-ANR, we can go through the entire path to calculate the MFR and check if it satisfies the bound in polynomial time, which means that MFR-ANR $\in$ NP. We prove its NP-hardness by showing that WLCC $\leq_p$ MFR-ANR.

Given an arbitrary instance $\mathscr{I}_1$ in WLCC, in which there are several LCCs among the links, as shown in Fig. 4.4, we can transform it into an instance $\mathscr{I}_2$ of MFR-ANR, i.e., $\mathscr{I}_1 \in WLCC \Rightarrow \mathscr{I}_2 = F(\mathscr{I}_1) \in$ MFR-ANR, where $F(\cdot)$ is a polynomial-time transformation function. We first construct a pipeline that consists of $|V|$ identical modules w, whose computational complexity is denoted as a function $\lambda_w(\cdot)$ of the same input data size z. We then make a copy of the entire topology of G and denote it as graph $G' = (V', E')$, where $V' = V$ and $E' = E$. Any vertex $v \in V$ in G not on a constrained link and any link $e \in E$ in G not in constraints (C,b) remain unchanged in V' and E', respectively, including the capacity of each link. We consider three types of LCCs:

(1) Case 1 of adjacent LCC: If there are η constrained links within one LCC that are all adjacent, their corresponding vertices in V' are bundled together and replaced with a virtual

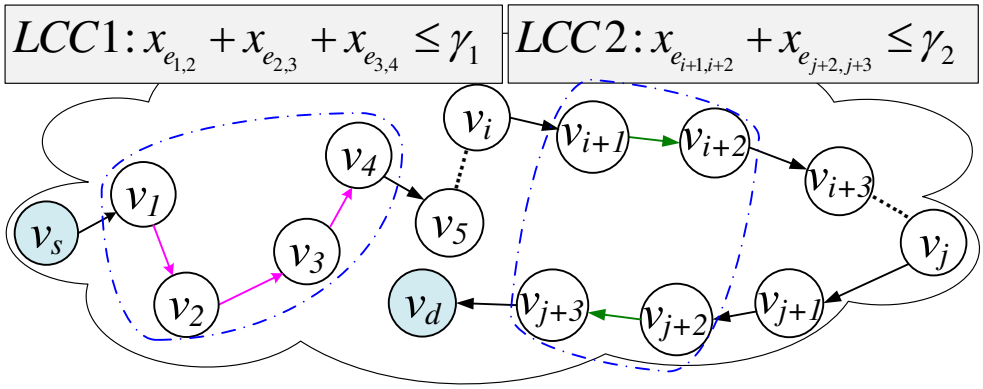


Figure 4.4: An example of the LCC-graph.

node v_{vir}, which contains the same number of self-loops as that of contiguously adjacent constrained links defined in the LCC. The computing power p of the virtual node is set to $\lambda_w(z) \cdot \gamma$, where γ is the LCC capacity. For example, in Fig. 4.4, the links $e_{1,2}$, $e_{2,3}$ and $e_{3,4}$ have an LCC $x_{e_{1,2}} + x_{e_{2,3}} + x_{e_{3,4}} \leq \gamma_1$ among them, so we have $\eta = 3$ and $\gamma = \gamma_1$;

(2) Case 2 of nonadjacent LCC: If there are ζ constrained links within one LCC that are all nonadjacent, their corresponding vertices in V' are bundled together and replaced with a virtual node v_{vir}. The computing power p of the virtual node is again set to $\lambda_w(z) \cdot \gamma$, where γ is the LCC capacity. For example, in Fig. 4.4, the links $e_{i+1,i+2}$ and $e_{j+2,j+3}$ have an LCC $x_{e_{i+1,i+2}} + x_{e_{j+2,j+3}} \leq \gamma_2$ between them, so we have $\zeta = 2$ and $\gamma = \gamma_2$;

(3) Case 3 of mixed LCC: This is a combination of the first two cases. The corresponding vertices in V' of all adjacent and nonadjacent constrained links within one LCC are bundled together and replaced with a virtual node v_{vir}, whose properties (self-loops and computing power) are treated the same way as in the first two cases.

Furthermore, the computing power of all other nodes is set to infinity. The newly constructed graph G' with one adjacent LCC and one nonadjacent LCC is shown in Fig. 4.5. Finally, we select a bound $B = K$. Obviously, this construction process can be done in polynomial time. The question for MFR-ANR is: does there exist a mapping path P' from v'_s to v'_d in G' that provides frame rate no less than B?

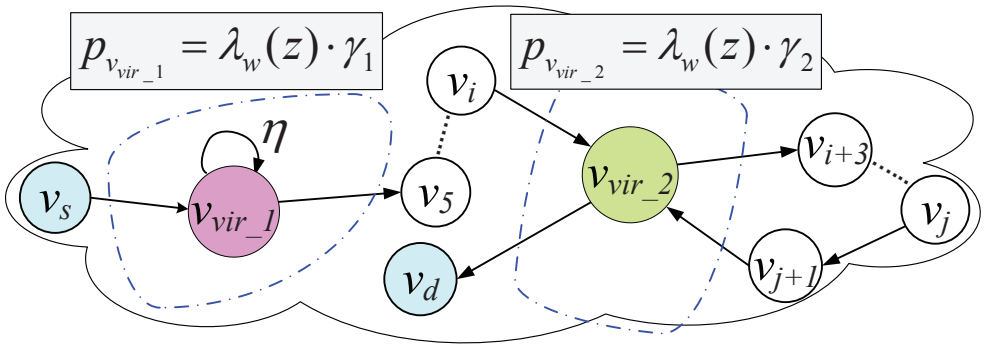


Figure 4.5: LCC-graph replaced with virtual nodes.

We show that given a solution to WLCC problem, we can find a solution to the MFR-ANR problem. Suppose that there exists a path P from v_s to v_d in G of width no less than K. We first identify a path P' in G' corresponding to path P in G, and then sequentially map the modules in the pipeline onto the nodes along the path P' from v'_s to v'_d, with the first module mapped onto source node v'_s and the last module mapped onto destination node v'_d. Between v'_s and v'_d, we sequentially map each module onto a regular node along the path, and when a virtual node is encountered, we have three different mapping schemes:

(1) Case 1 of adjacent LCC: If this virtual node is converted from η adjacent constrained links, among which η' ($\eta' \leq \eta$) links are on path P, we map η' contiguous modules to it;

(2) Case 2 of nonadjacent LCC: If this virtual node is converted from ζ nonadjacent constrained links, among which ζ' ($\zeta' \leq \zeta$) links are on path P, we map one module to it every time when the path passes through it for total ζ' times;

(3) Case 3 of mixed LCC: If this virtual node is converted from both adjacent and nonadjacent constrained links, we map modules to it according to a combination of the mapping strategies for virtual nodes converted from either a single adjacent LCC or a single nonadjacent LCC as specified in the first two cases.

Any remaining contiguous modules are mapped onto the non-virtual destination node v'_d. If the destination node v'_d itself is a virtual node in G', we can create a special node with infinite computing power and connect the destination node v'_d to it with infinite bandwidth

on the link. This special node is then considered as the new destination node to run the last module and all remaining unmapped modules. We consider the following four cases:

Case 1: where the maximum capacity of path P is not on any LCC link: the frame rate in G' is equal to the corresponding maximum capacity in G. Therefore, path P' from v'_s to v'_d in G' provides frame rate that is no less than $K = B$;

Case 2: where the maximum capacity of path P is on one of the adjacent LCC links: we calculate the frame rate on the virtual node in G' as $1/(\lambda_w(z)/\frac{p}{\eta'})$. After plugging in $p = \lambda_w(z) \cdot \gamma$, the frame rate becomes: $1/(\lambda_w(z)/\frac{\lambda_w(z)\cdot\gamma}{\eta'}) = \frac{\gamma}{\eta'}$. In WLCC problem, the bottleneck bandwidth x_{BN} of the widest path P has the following inequality: $K \leq x_{\mathrm{BN}} \leq \frac{\gamma}{\eta'}$ since all constrained links share the capacity γ. Hence, the corresponding mapping path P' from v'_s to v'_d in G' provides frame rate at $\frac{\gamma}{\eta'} \geq K = B$;

Case 3: where the maximum capacity of path P is on one of the nonadjacent LCC links: we calculate the frame rate on the virtual node in G' as $1/(\lambda_w(z)/\frac{p}{\zeta'})$. After plugging in $p = \lambda_w(z) \cdot \gamma$, the frame rate becomes: $1/(\lambda_w(z)/\frac{\lambda_w(z)\cdot\gamma}{\zeta'}) = \frac{\gamma}{\zeta'}$. In WLCC problem, the bottleneck bandwidth x_{BN} of the widest path P has the following inequality: $K \leq x_{\mathrm{BN}} \leq \frac{\gamma}{\zeta'}$ since all constrained links share the capacity γ. Hence, the corresponding mapping path P' from v'_s to v'_d in G' provides frame rate at $\frac{\gamma}{\zeta'} \geq K = B$;

Case 4: where the maximum capacity of path P is on one of the mixed LCC links: we calculate the frame rate on the virtual node in G' as $1/(\lambda_w(z)/\frac{p}{\eta'+\zeta'})$. After plugging in $p = \lambda_w(z) \cdot \gamma$, the frame rate becomes: $1/(\lambda_w(z)/\frac{\lambda_w(z)\cdot\gamma}{\eta'+\zeta'}) = \frac{\gamma}{\eta'+\zeta'}$. In WLCC problem, the bottleneck bandwidth x_{BN} of the widest path P has the following inequality: $K \leq x_{\mathrm{BN}} \leq \frac{\gamma}{\eta'+\zeta'}$ since all constrained links share the capacity γ. Hence, the corresponding mapping path P' from v'_s to v'_d in G' provides frame rate at $\frac{\gamma}{\eta'+\zeta'} \geq K = B$.

Therefore, we conclude that the mapping path P' is the solution to the instance $\mathscr{I}_2$ of MFR-ANR problem.

Now we show that if there is a solution to MFR-ANR problem, we can also find a solution to WLCC problem in polynomial time. Given a path P' from v'_s to v'_d in G' with frame

rate $\geq B$, we first identify a corresponding path P in G. We also consider the following four cases:

Case 1: where the frame rate is incurred on a network link in G', the corresponding link in G has the bottleneck bandwidth of the widest path in WLCC;

Case 2: where a virtual node converted from an adjacent LCC incurs the frame rate as: $1/(\lambda_w(z)/\frac{p}{\eta'}) = 1/(\lambda_w(z)/\frac{\lambda_w(z)\cdot\gamma}{\eta'}) = \frac{\gamma}{\eta'} \geq B$, the corresponding path P in G has the bottleneck bandwidth $x_{\mathrm{BN}} = \frac{\gamma}{\eta'} \geq B = K$ of the widest path in WLCC;

Cases 3 and 4: where the solution derivation steps are very similar to those in case 2, except for replacing η' with ζ' and $\eta' + \zeta'$, respectively.

Therefore, we conclude that path P in G from v_s to v_d is the solution to the instance $\mathscr{I}_1$ of WLCC problem. □

4.2 Optimal and Heuristic Algorithm Design

4.2.1 Optimal Solution to MED-ANR

For unitary processing applications, our goal is to minimize the ED incurred on nodes and links from source to destination to achieve the fastest response. Since a single dataset is processed and there is only one module being executed at any particular time point, nodes can be reused but are not shared concurrently among different modules mapped on the same node. We present a DP-based polynomial-time optimal solution to solve MED-ANR and provide its correctness proof.

Let $T_{\mathrm{ED}}^{j-1}(v_i)$ denote the MED with the first j modules mapped to a path from the source node v_s to node v_i in the network. We have the following recursion leading to the final solution $T_{\mathrm{ED}}^{m-1}(v_d)$:

$$T_{\mathrm{ED}}^{j-1}(v_i)_{j=2 \text{ to } m, v_i \in V_c} = \min \left(\begin{array}{l} T_{\mathrm{ED}}^{j-2}(v_i) + \frac{\lambda_{w_{j-1}}(z_{j-2,j-1})}{p_i} \\ \min\limits_{v_u \in pre(v_i)} \left(T_{\mathrm{ED}}^{j-2}(v_u) + \frac{\lambda_{w_{j-1}}(z_{j-2,j-1})}{p_i} + \frac{z_{j-2,j-1}}{b_{u,i}} + d_{u,i} \right) \end{array} \right), \quad (4.2.1)$$

where $pre(v_i)$ denotes the set of preceding (incident) neighbor nodes of node v_i, with the base condition computed as:

$$T_{\mathrm{ED}}^{1}(v_i)_{\substack{v_i \in V_c, \text{ and } v_i \neq v_s}} = \begin{cases} \frac{\lambda_{w_1}(z_{0,1})}{p_i} + \frac{z_{0,1}}{b_{s,i}} + d_{s,i}, & \text{if } l_{s,i} \in E_c \\ +\infty, & \text{otherwise} \end{cases} \tag{4.2.2}$$

on the second column of a 2D table as shown in Fig. 4.6, where modules are listed in order along the horizontal axis and nodes are arranged along the vertical axis, constructed by a typical DP-based method.

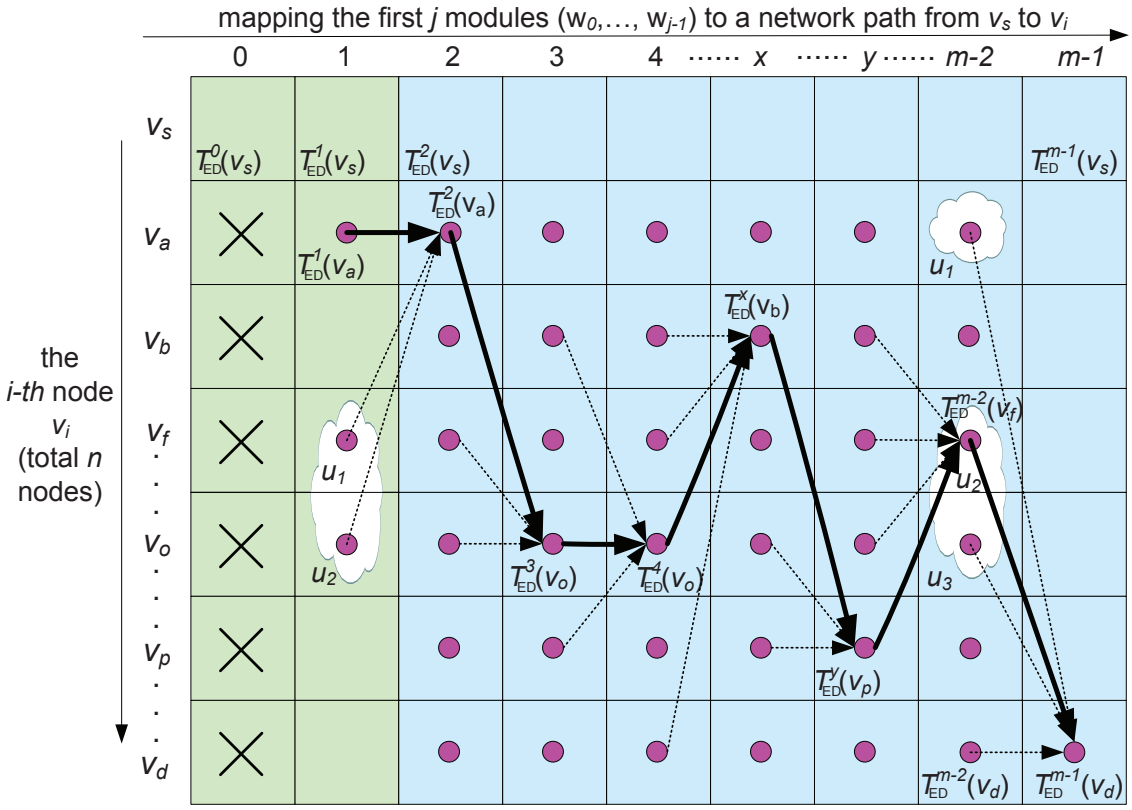


Figure 4.6: Construction of 2D table based on DP.

Every cell $T_{\mathrm{ED}}^{j-1}(v_i)$ in the 2D DP table represents an optimal mapping solution that maps the first j modules in the pipeline to a path between the source node v_s and node v_i in the network. In MED-ANR, each cell is calculated from the intermediate mapping results up to its preceding nodes or itself, which are stored in its left column $T_{\mathrm{ED}}^{j-2}(\cdot)$. The

pseudocode of this algorithm is presented in Alg. 1, whose complexity is determined by the number of steps needed to fill out the 2D DP table as shown in Fig. 4.6. Since we have total m modules to map in the pipeline, for each module we consider n computer nodes in the network, and for each node we consider all of its preceding nodes, the complexity of filling out the 2D DP table is $O(m \times |E_c|)$, where $|E_c|$ is the number of links in the network, which is upper bounded by $n \times n$.

Algorithm 1 Optimal solution to MED-ANR

Input: A linear computing pipeline with m modules, a heterogenous network $G_c = (V_c, E_c)$, where $|V_c| = n$, and a pair of nodes (v_0,v_{n-1}) in V_c, representing source v_s and destination v_d, respectively.

Output: MED of the pipeline mapped onto a selected network path P from v_s to v_d.

Initialize the 1st column of the 2D table: $T_{[0][0]} = 0.0, T_{[i][0]} = NULL$, $i = 1, 2, \ldots, n-1$;
for all nodes v_i from $i = 0$ to $n-1$ with only two modules w_0 and w_1 **do**
    if $l_{s,i} \in E$ **then**
        Calculate MED for cell $T_{[i][1]}$ in the 2nd column as the base condition;
    else
        $T_{[i][1]} = \infty$;
for all modules w_j from $j = 2$ to $m-2$ **do**
    for all nodes v_i from $i = 0$ to $n-1$ **do**
        if module w_{j-1} is mapped to node v_i **then**
            Map module w_j to node v_i, calculate total delay D_1;
        else
            for all possible mapped preceding nodes $pre(v_i)$ directly connected to v_i **do**
                Map module w_j to node v_i, calculate total delay $D(pre(v_i))$;
            Choose the minimum delay among all preceding nodes of v_i: $D_2 = \min(D(pre(v_i)))$;
        $T_{[i][j]} = \min(D_1, D_2)$;
if module w_{m-2} is mapped to node v_{n-1} **then**
    Map module w_{m-1} to node v_{n-1}, calculate total delay D'_1;
else
    for all possible mapped preceding nodes $pre(v_{n-1})$ directly connected to v_{n-1} **do**
        Map module w_{m-1} to node v_{n-1}, calculate total delay $D(pre(v_{n-1}))$;
    Choose the minimum delay among all preceding nodes of v_{n-1}: $D'_2 = \min(D(pre(v_{n-1})))$;
$T_{[n-1][m-1]} = \min(D'_1, D'_2)$;
return $T_{[n-1][m-1]}$ as the final MED.

Theorem 4. *The DP-based solution defined in Eq. 4.2.1 to MED-ANR is optimal.*

Proof. At each recursive step, there are only two sub-cases, the minimum of which is chosen as the MED to fill in a new cell $T_{\text{ED}}^{j-1}(v_i)$: (i) In sub-case 1, we run the new module on the same node running the last module in the previous mapping subproblem $T_{\text{ED}}^{j-2}(v_i)$. In other words, the last two or more modules are mapped to the same node v_i. Therefore, we only need to add the computing time of the last module on node v_i to the previous total delay, which is represented by a horizontal incident link from its left neighbor cell in the 2D table. (ii) In sub-case 2, the new module is mapped to node v_i and the last node v_u in a previous mapping subproblem $T_{\text{ED}}^{j-2}(v_u)$ is one of the neighbor nodes of node v_i, which is represented by a slanted incident link from a neighbor cell on the left column to node v_i. In Fig. 4.6, a set of neighbor nodes of node v_i are enclosed in a cloudy region in the previous column. We calculate the ED for all possible mappings using slanted incident links of node v_i and choose the minimal one, which is further compared with the one calculated in sub-case 1 using the horizontal incident link from the left neighbor cell.

The minimum of these two sub-cases is selected as the MED for the partial pipeline mapping to a path between nodes v_s and v_i. Since each cell provides an optimal partial solution to a subproblem and mapping a new module does not affect the optimality of any previously computed partial solutions. Thus, DP-based procedure provides an optimal solution to MED-ANR. □

4.2.2 Heuristic Algorithms for NP-Complete Problems

We develop heuristic solutions by adapting the optimal DP-based method for MED-ANR to the NP-complete MED mapping problems with some necessary modifications. The heuristics for MED-NNR/CNR are similar to that defined in Eq. 4.2.1 except that (i) in MED-CNR, we skip the cell of a neighbor node (slanted incident link) in the left column whose solution (path) involves the current node to ensure a loop-free path, and (ii) in MED-NNR, we further skip the immediate left neighbor cell (horizontal incident link) to ensure that the

current node is never reused. Such path backtracking and cell exclusion reflect the heuristic nature of these algorithms since those skipped cells may lead to the actual optimal solution.

For streaming applications, we use $1/T_{\mathrm{BT}}^{j-1}(v_i)$ to denote the MFR with the first j modules mapped to a network path from source node v_s to node v_i in the computer network, and the following recursion leads to the final solution $T_{\mathrm{BT}}^{m-1}(v_d)$:

$$T_{\mathrm{BT}}^{j-1}(v_i) \underset{j=2 \text{ to } m, v_i \in V}{} = \min \left(\begin{array}{l} \max \left(\begin{array}{l} T_{\mathrm{BT}}^{j-2}(v_i), \\ \frac{\alpha_i \lambda_{w_{j-1}}(z_{j-2,j-1})}{p_i} \end{array} \right), \\ \min\limits_{v_u \in pre(v_i)} \left(\max \left(\begin{array}{l} T_{\mathrm{BT}}^{j-2}(v_u), \\ \frac{\alpha_i \lambda_{w_{j-1}}(z_{j-2,j-1})}{p_i}, \\ \frac{\beta_{u,i} z_{j-2,j-1}}{b_{u,i}} + d_{u,i} \end{array} \right) \right) \end{array} \right) \tag{4.2.3}$$

with the base condition computed as:

$$T_{\mathrm{BT}}^{1}(v_i) \underset{v_i \in V, \text{ and } v_i \neq v_s}{} = \begin{cases} \max(\frac{f_{w_1}(z_{0,1})}{p_i}, \frac{z_{0,1}}{b_{s,i}} + d_{s,i}) , & \text{if } l_{s,i} \in E_c \\ +\infty , & \text{otherwise} \end{cases} \tag{4.2.4}$$

on the second column of the 2D table and we have $T_{\mathrm{BT}}^{0}(v_s) = 0$. Note that α_i denotes the number of modules assigned to node v_i and $\beta_{u,i}$ denotes the number of datasets transferred over the link between nodes v_u and v_i. In MFR-NNR, we have $\alpha = 1$ and $\beta = 1$.

The steps for filling out the 2D table for MFR problems are similar to those for their corresponding MED problems but differ in the following aspects: at each step, we ensure that the computing power of reused nodes be equally shared and calculate the bottleneck of the path instead of the total delay. These solutions are heuristic in nature because (i) in MFR-CNR/ANR, the share of resources affects the optimality of previously computed partial solutions, and (ii) in MFR-NNR, when a node has been used by all its neighbor nodes at previous mapping steps, this heuristic algorithm may not be able to find an optimal solution if this node is the only one leading to the destination node or obtain a suboptimal solution if there are multiple nodes leading to the destination. Based on our extensive experiments, we would like to point out that the occurrences of unsuccessful mapping

solutions due to backtracking are quite rare and all these heuristics achieve satisfactory mapping performance on a large number of simulated pipelines and networks with different problem scales.

Chapter 5

DAG-structured General Workflow Optimization

In this chapter, we investigate the mapping optimization problems with more complicated workflow structure, i.e., intricate inter-module dependencies, conduct a rigorous analysis of stability for streaming applications, and present heuristic algorithms to optimize workflow end-to-end performance. We further propose the distributed version of the mapping algorithms and adapt them to the faulty network environments under fault-tolerant constraint.

5.1 Problem Categorization and Complexity Analysis

In general DAG-structured workflow mappings, modules may have multiple incoming or outgoing dependencies, which impose multiple constraints on data arriving or sending times. Hence, the general workflow scheduling is more challenging compared to linear pipelines, which is known to be NP-complete in general scenarios [73], even in some simplified cases [142] such as the mapping of linear pipelines, or restricted cases [128] such as the assignment of tasks with one or two time units on only two processors.

As shown in Fig. 5.1, we systematically categorize the workflow mapping problems into a number of classes based on different mapping objectives, mapping constraints, resource share constraints, network topology constraints, resource constraints and fault-tolerance

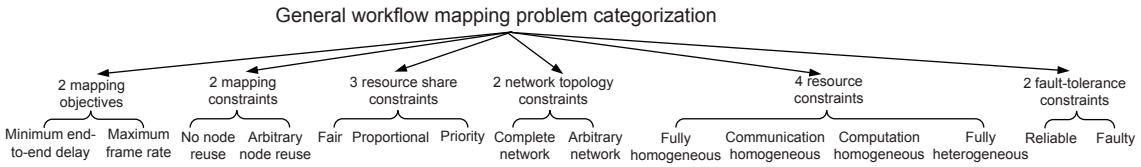


Figure 5.1: Problem categorization of general workflow mappings.

constraints. MED and MFR are two most important end-to-end performance metrics considered in many practical distributed systems. Workflow applications for MED take a one-time dataset as input and aim to minimize the ED for fast system response, e.g., an interactive view operation in scientific visualization; while for MFR, a sequence of datasets are continuously fed into the computing workflow to produce a smooth flow of final results, e.g., an online simulation producing time-series datasets to simulate a constantly evolving phenomenon. Depending on the nature of the computation, the topology of the network, and the availability of the required software/hardware resources, computer nodes may or may not be used for executing multiple modules. The resource constraints are posed by the wide variety of network environments with either homogeneous or heterogeneous communication links and computer nodes. Fault tolerance and recovery mechanism have become more and more important in today's large-scale collaborative computing infrastructures.

5.2 Exact End-to-end Calculation (extED)

We investigate the problem of mapping general DAG-structured workflows to networks to optimize the end-to-end workflow performance. Obviously, the design and evaluation of a workflow mapping algorithm rely on an accurate calculation of module execution time on each node and data transfer time over each link for any given mapping scheme. Note that if multiple modules are mapped onto the same node (i.e., node reuse), the node's computing

resource is shared in a fair manner by concurrent modules on that node[1]. Similarly, the bandwidth of a network link is equally shared by concurrent data transfers over the same link[2]. The time cost calculation is a complicated task because determining the exact number of concurrent module executions on a node or concurrent data transfers over a link at different time intervals is not trivial. In unitary processing applications, multiple independent modules mapped on the same node may not necessarily share computing resources if their execution times do not overlap due to different arrival times of their input data, while in streaming applications, multiple dependent modules mapped on the same node may run concurrently to process different instances of input data. Here, "independent" means that there does not exist a dependency path between two modules in the workflow. The difficulty of this mapping problem essentially arises from the topological matching nature in the spatial domain, which is further compounded by the resource sharing complicacy in the temporal dimension if multiple modules are deployed on the same node.

5.2.1 An Example of Resource Share

We shall use a simple numerical example to illustrate the complexity of resource sharing in a workflow with five modules and six dependency edges mapped onto a network consisting of four nodes and five links. As shown in Fig. 5.2, modules w_0 and w_4 are mapped to the source node v_s and destination node v_d, respectively, and three independent intermediate modules w_1, w_2 and w_3 are mapped to node v_1. In the workflow, the number above each module is the module's computational requirement (CR) and the number on each edge is the data size (DS) transferred along that edge. Similarly, in the network, the number below each node is the node's processing power (PP) and the number on each link is the link's bandwidth (BW). Here, we consider module execution time defined as $\frac{CR}{PP/CM}$ and

[1]The fair share of CPU cycles is supported by many modern Operating Systems that employ a round-robin type of CPU scheduling algorithm.

[2]The fair share of link bandwidths is supported by the wide use of TCP-friendly transport protocols in wide-area networks.

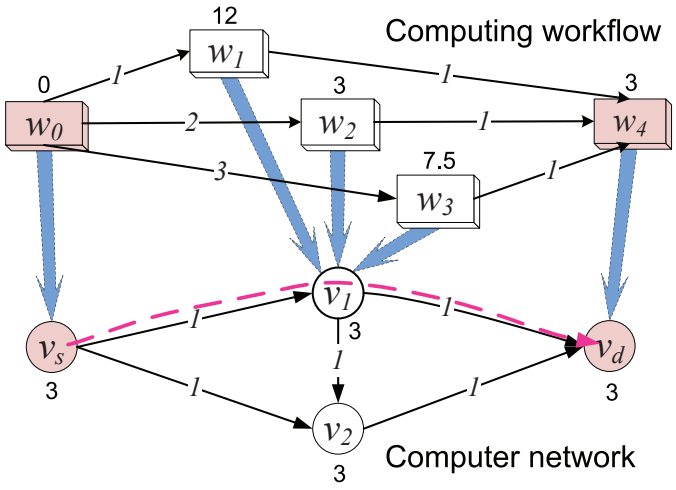


Figure 5.2: An example of workflow mapping.

data transfer time defined as $\frac{DS}{BW/CD}$, where CM and CD denote the number of concurrent module executions and data transfers, respectively.

The resource sharing dynamics in the above example is illustrated in Fig. 5.3, where the numerical expression above each line is the partial time cost of either a module execution or a data transfer. When the workflow execution begins, module w_0 sends out data of 1, 2, and 3 units at the same time to w_1, w_2 and w_3, respectively. Since the data size of edge $e_{0,1}$ is of 1 unit and link $l_{s,1}$ is shared equally by 3 concurrent data transfers at the beginning, it takes 3 (i.e., $\frac{1}{1/3}$) time units to transfer the data to w_1. At time point 3, w_1 finishes receiving the data and starts its execution on node v_1 with an exclusive use of the resource (i.e., $\frac{6}{3/1}$) until time point 5 when module w_2 also starts to execute on node v_1 in a fair share manner (i.e., $\frac{1.5}{3/2}$). The data of edges $e_{0,2}$ and $e_{0,3}$ are still moving together over link $l_{s,1}$ between time points 3 and 5 (i.e., $\frac{1}{1/2}$), and after that only edge $e_{0,3}$ is using link $l_{s,1}$ to transfer the rest of the data (i.e., $\frac{1}{1/1}$). After time point 6, all data transfers are completed and modules w_1, w_2 and w_3 are running concurrently on node v_1 with equally shared resources (i.e., $\frac{1.5}{3/3}$). Module w_2 finishes execution first and starts its data transfer to module w_4 at time point 7.5, and meanwhile, node v_1 is shared by two modules between time points 7.5 and

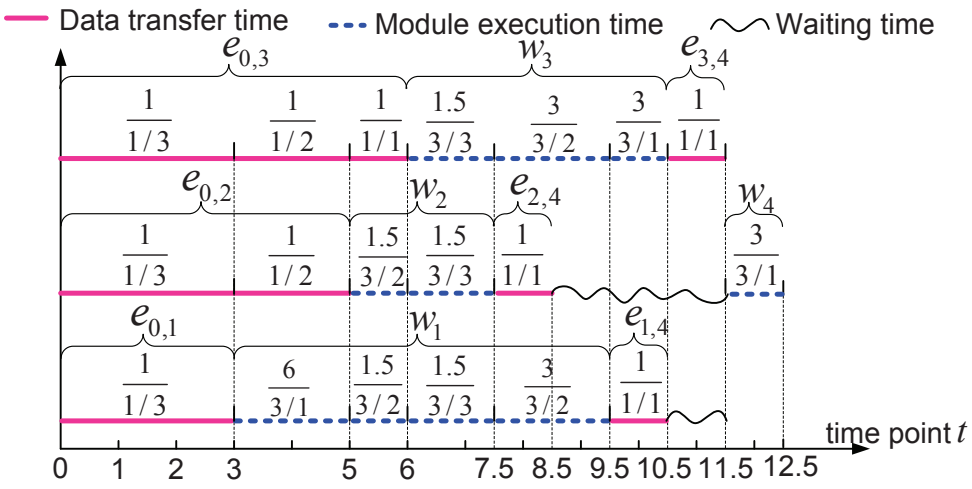


Figure 5.3: Time serial analysis of resource sharing dynamics.

9.5 (i.e., $\frac{3}{3/2}$), after which module w_1 also finishes execution. Module w_4 finishes receiving data from modules w_2, w_1, and w_3 at time point 8.5, 10.5, and 11.5, respectively. Module w_4 starts execution on node v_d when it receives all required inputs from three preceding modules and its execution time is 1 unit (i.e., $\frac{3}{3/1}$). Therefore, the total end-to-end delay of the mapped workflow is 12.5 units, which is calculated along the longest path.

5.2.2 Algorithm Design

We observe that the number of shared modules in above example keeps changing for each of the independent modules/edges mapped on the same node/link, and therefore will pose a demand for a systematic way to calculate the ED.

In unitary processing applications, when multiple modules are deployed on the same node, only those without dependency could run concurrently and share CPU resources in a fair manner, so is the case for bandwidth share. Moreover, those modules with partially overlapped execution times do not share CPU resources during the entire period since they might have different start and end times, which, in turn, depend on the execution times of their preceding modules as well as the associated data transfer times. While for streaming

applications, we assume they always share because node or link resources could be shared by multiple modules or edges that are processing or transferring different instances of input datasets on the same node or over the same link, and hence we only consider FR or throughput. For an accurate estimation of ED, the exact number of concurrently running modules and transferred datasets must be determined at each time interval.

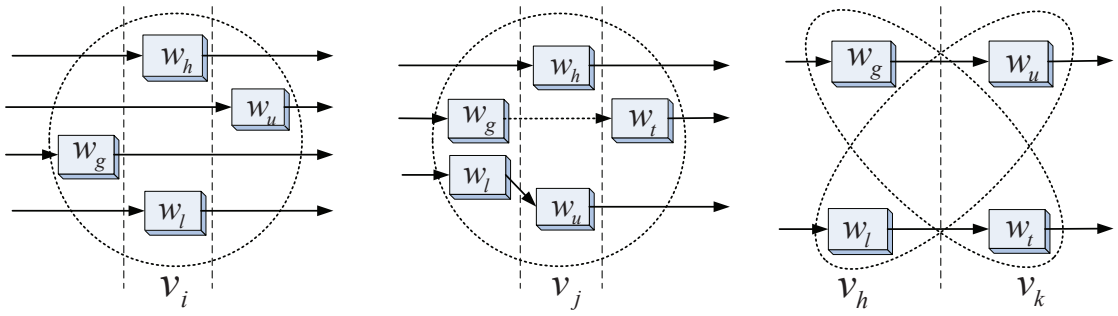
We identify an independent set of modules mapped to the same node to calculate the exact number of concurrent modules. Note again that two modules are considered "independent" if there does not exist any path between them in the workflow, which could be quickly verified by a Breadth-First Search algorithm. We consider three basic types of independent sets: fully-independent, partially-independent with intra-node dependency, and partially-independent with inter-node dependency, which are defined below. Other sets could be the combinations of any two or three of these basic types.

Definition 3. *Fully-Independent Set (FIS) is a set of modules mapped to the same node where there is no path between any pair of modules.*

Definition 4. *Partially-Independent Set with Intra-node Dependency (PIS-IntraD) is a set of modules mapped to the same node where at least one pair of modules have a path between them and at least one pair of other modules do not have any path between them.*

Definition 5. *Partially-Independent Sets with Inter-node Dependency (PIS-InterD) are two sets of independent modules mapped to two different nodes with cross execution dependency such that the upstream modules on one dependency path and the downstream modules on another dependency path are mapped to the same node.*

These three basic types of independent sets are visualized in Fig. 5.4, where modules within a circle or an ellipse denote the independent or partially independent modules mapped on that node. In Fig. 5.4(a), each module mapped onto node v_i is independent of any other modules on the same node. In Fig. 5.4(b), modules w_g and w_t are dependent but they are both independent of module w_h, so is the case for modules w_l and w_u. Fig. 5.4(c)



(a) (b) (c)

Figure 5.4: Three types of independent sets of modules mapped to the same node: (a) FIS, (b) PIS-IntraD, (c) PIS-InterD.

illustrates PIS-InterD between nodes v_h and v_k in a workflow where modules w_u and w_t depend on modules w_g and w_l, respectively. Modules w_u and w_l are mapped to the same node v_h and modules w_g and w_t are mapped to another node v_k. In Fig. 5.4(c), even though the modules on each node are independent of each other, they may or may not run concurrently. We ignore data transfer times for simplicity and consider five possible cases:

- If $t^f_{w_u} \leq t^s_{w_l}$, modules w_g and w_u finish execution before module w_l starts, and hence no resource sharing;
- If $t^f_{w_g} \leq t^f_{w_l}$, module w_u shares resource with w_l, and no resource sharing between modules w_g and w_t;
- If $t^f_{w_l} \leq t^f_{w_g}$, module w_g shares resource with w_t, and no resource sharing between modules w_u and w_l;
- If $t^f_{w_t} \leq t^s_{w_g}$, modules w_l and w_t finish execution before module w_g starts, and hence no resource sharing;
- If $t^f_{w_g} = t^s_{w_t}$ or $t^f_{w_l} = t^s_{w_u}$, no resource sharing.

No matter which type of independent sets (FIS, PIS-IntraD, or PIS-InterD) the mapping might result in, we can calculate the *exact ED* (extED) of a mapped workflow by adding up

all the time cost components (module execution and data transfer) along the CP as shown in Eq. 3.2.1.

Before the extED calculation, we preprocess the workflow and network by replacing each dependency edge in the workflow G_w with a virtual module whose computational requirement is equal to the corresponding transferred data size and replacing each mapped network link with a virtual node whose processing power is equal to the corresponding BW. By doing so, we only need to consider module execution times on nodes in the resultant virtual workflow ${G_w}'$ and network ${G_c}'$. Here, we ignore the MLD for simplicity.

For a given mapping scheme, we employ a well-known polynomial-time longest path algorithm to find the CP since the CP is essentially the longest path in a DAG, which largely depends on the accurate calculation of each time components in the mapped workflow. The key strategy of the extED calculation reflects the dynamics in resource sharing at runtime. For each module, we first calculate its independent set (IDS), in which all modules are independent of the current module, and then recompute the number of its concurrent modules on the same node every time when one of the following conditions is met: (a) one concurrent module finishes execution; and (b) one module in the IDS of the current module starts execution. The details of the extED calculation are provided in Alg. 2.

Considering execution dependency, we always start from the set $ready({V_m}')$ of all currently "*ready*" modules in the workflow for every "while" loop in line 11. A module is in the "*ready*" state when all its preceding modules finish execution. We then compute a set $est(ready({V_w}'))$ (a subset of set $ready({V_w}')$) of earliest start modules from the "*ready*" set as shown in line 12. We consider the following two possible scenarios for module $w_i \in est(ready({V_w}'))$ in the extED algorithm:

1. *Lines 14-15,* $|ids(w_i)| == 0$: Since w_i does not have any independent modules, its execution time is directly computed.

2. *Lines 16-30,* $|ids(w_i)| > 0$: If any module in $ids(w_i)$ is "*ready*", we find a set set_1 of earliest start modules from all "*ready*" modules in $\{ids(w_i) \cup w_i\}$ and estimate

Algorithm 2 extED(G_w', G_c', f)
Input: A converted workflow graph $G_w' = (V_w', E_w')$, a converted network graph $G_c' = (V_c', E_c')$, and a mapping scheme $f : w_i \rightarrow v_h$, $w_i \in V_w'$ and $v_h \in V_c'$.
Output: The extED of the mapped workflow.

$t^s(set)$: the set of start times of all modules in the *set*;
$t^f(set)$: the set of finish times of all modules in the *set*;
$ids(w)$: the independent set of module w on the same node (excluding w);
$|set|$: the number of modules in the *set*;
$est(set) = \{w|\ w \in set$ and $t^s_w = \min(t^s(set))\}$;
$ready(set) = \{w|\ w \in set$ and w is "*ready*"$\}$;
for all module $w_i \in V_w'$ **do**
    Find $ids(w_i)$;
    Set w_i as "$unfinished$";
Set w_0 as "*ready*";
while exist "$unfinished$" modules $\in V_w'$ **do**
    Find $set_0 = est(ready(V_w'))$;
    for all module $w_i \in set_0$ **do**
        if $|ids(w_i)| == 0$ **then**
            Calculate $T_{\text{exec}}(w_i)$ and set w_i as "$finished$";
        else
            Find $set_1 = \{w|\ w$ is "*ready*" & $w \in est(ids(w_i) \cup w_i)\}$;
            Estimate $t^f(set_1)$;
            $set_2 = \{w \mid w \in ids(w_i)$ & $\notin set_1, t^s(w) < \min(t^f(set_1))$, and w is "*ready*" and "$unfinished$"$\}$;
            if $|set_2| > 0$ **then**
                for all module $w_j \in set_1$ **do**
                    Calculate partial amount of datasets finished by module w_j from $t^s(set_1)$ to $\min(t^s(set_2))$;
                    Update the new $t^s_{w_j} = \min(t^s(set_2))$;
            else
                for all module $w_j \in set_1$ **do**
                    if $t^f_{w_j} == \min(t^f(set_1))$ **then**
                        Calculate $T_{\text{exec}}(w_j)$ and set w_j as "$finished$";
                    else
                        Calculate partial amount of datasets finished by module w_j from $t^s(set_1)$ to $\min(t^f(set_1))$;
                        Update the new $t^s_{w_j} = \min(t^f(set_1))$;
    Mark all ready modules as "*ready*";
Compute the CP based on the time components $T_{\text{exec}}(w_i)$ for all $w_i \in V_w'$;
return $T_{\text{ED}}(\text{CP})$ of the mapped workflow.

the finish time of each module in set_1 under the assumption that all those modules are running concurrently during the entire execution. We would like to point out that the above merely provides an estimate because the assumption may not always hold. In fact, these modules will finish at different times in most cases. Based on this estimate, we check the value of set_2 defined in line 19:

(a) $|set_2| > 0$ means that at least one module, which is in $ids(w_i)$ but not in set_1, starts to execute before any module in set_1 finishes execution. We calculate the partial amount of datasets finished by each module in set_1 until one of the modules in set_2 starts execution, which corresponds to $\delta_w(t)$ in Eq. 3.1.1, and then reset the start times of those partially finished modules in set_1 to be the start time of the first started module in set_2.

(b) $|set_2| == 0$ means that all modules, which are in $ids(w_i)$ but not in set_1, do not start to execute until at least one module in set_1 finishes execution. We calculate the partial amount of datasets finished by each module in set_1 until the first one finishes execution, and then reset the start times of those partially finished modules in set_1 to be the finish time of the first finished module in set_1.

3. *Line 31*: Check if any previously unready modules become ready.

Note that the values of $\alpha(t)$ and $\beta(t)$ are updated at each time step before the partial module execution and data transfer time is calculated (e.g., in lines 22 and 29). According to lines 11, 13, 21 and 25, the time complexity of calculating accurate module execution time is $O(m'^3)$, where m' is the number of modules in the converted workflow $G_w{}'$.

5.3 Mapping Solution to MED

The workflow mapping problem with arbitrary node reuse for MED is formally defined as follows:

Definition 6. *Given a DAG-structured computing workflow $G_w = (V_w, E_w)$ and a heterogeneous computer network $G_c = (V_c, E_c)$, we wish to find a mapping scheme that assigns each computing module to a network node such that the mapped workflow achieves:*

$$\text{MED} = \min_{\text{all possible mappings}} (T_{\text{ED}}). \tag{5.3.1}$$

Note that MED is the minimum sum of time components along the CP. The NP-completeness of the DAG scheduling problem rules out any polynomial optimal solutions [116]. We develop a heuristic approach using a RCP algorithm that recursively chooses the CP based on the previous round of calculation and maps it to the network using a DP-based procedure until the mapping results converge to an optimal or suboptimal point or a certain termination condition is met. For example, the difference in mapping performance between two contiguous rounds is less than a preset threshold.

We significantly improve this algorithm by designing and incorporating a new non-critical module mapping scheme based on A^* Search algorithm and Beam Search algorithm. Moreover, we also provide a decentralized version of the RCP algorithm to accommodate the large-scale applications and versatile resource platforms.

5.3.1 Recursive Critical Path (RCP) Algorithm

As shown in Fig. 5.5, the key idea of the proposed RCP algorithm is described as follows:

1. Assume the network topology to be complete with homogeneous computer nodes and network links, and determine the initial module execution and data transfer times. Thus, we only need to consider the workflow for time cost calculation;

2. Find the CP of the workflow with initial time cost components using a longest path algorithm, defined as $FindCriticalPath(G_w, w_0, w_{m-1})$;

3. Remove the assumption on resource homogeneity and connectivity completeness, and map the current CP to the actual network using the optimal pipeline mapping

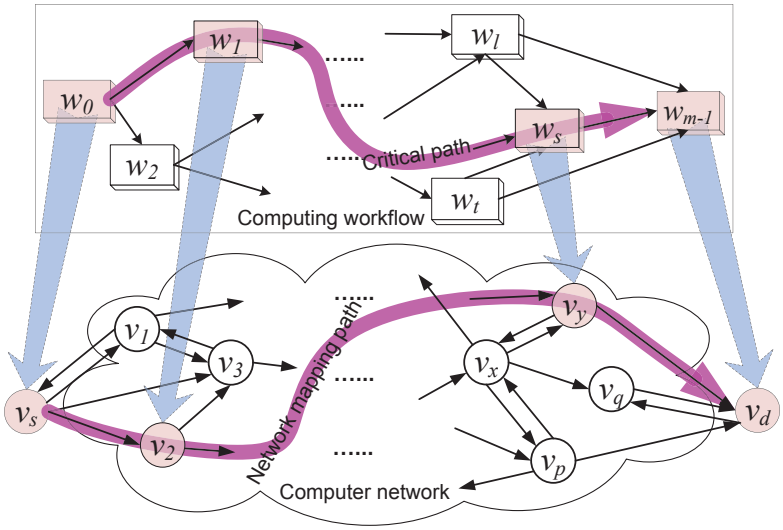


Figure 5.5: Map a CP in the workflow to a path in the computer network.

algorithm based on a DP-based procedure for MED with arbitrary node reuse as proposed in Chapter 4.2.1, defined as $MapCriticalPath(P, G_c, v_s, v_d)$;

4. Map non-critical/branch modules that are not on the CP in the workflow using a greedy approach, defined as $MapNonCriticalModules(P, G_w, G_c, v_s, v_d)$;

5. Compute a new CP using the updated mapping scheme and calculate the new MED.

Steps 2-5 are repeated until a certain condition is met, for example, the difference between a new MED resulted from the current mapping and an old MED resulted from the previous mapping is less than a preset threshold. The pseudocode of the RCP mapping algorithm for MED is given in Alg. 3. The complexity of the RCP algorithm is $O(k(m+|E_w|)\cdot|E_c|)$, where m represents the number of modules in the workflow, $|E_w|$ and $|E_c|$ denote the number of dependency edges in the workflow and communication links in the network, respectively, and k is the number of iterations where CPs are calculated and mapped.

Algorithm 3 RCP(G_w, G_c, v_s, v_d)

1: $MED_0 = MED_{max} = MaxValue$;
2: Create ${G_c}^*$ by assuming resource homogeneity and connectivity completeness in G_c;
3: Calculate initial time cost components for ${G_w}^1$ based on ${G_c}^*$;
4: Find a CP P_1;
5: $MED_1({G_w}^1) = \sum(T_{\text{exec}}(P_1) + T_{\text{tran}}(P_1))$;
6: $i = 1$;
7: **while** $|MED_i - MED_{i-1}| \geq Threshold$ **do**
8: Call $MapCriticalPath(P_i, G_c, v_s, v_d)$ to map the modules on CP P_i to network G_c;
9: Call $MapNonCriticalModules(P_i, {G_w}^i, G_c, v_s, v_d)$ to map the modules not on CP to network G_c;
10: $i = i + 1$;
11: Calculate new time cost for ${G_w}^i$ with the current mapping;
12: Fina a new CP P_i;
13: $MED_i({G_w}^i) = \sum(T_{\text{exec}}(P_i) + T_{\text{tran}}(P_i))$;
14: **return** $MED_i({G_w}^i)$.

Critical Path Calculation

We employ a well-known polynomial longest path (LP) algorithm to find the CP since the CP is essentially the LP in a DAG in terms of MED. The CP is selected with the nodes whose earliest start time (EST) is equal to their corresponding latest start time (LST). The EST is calculated by the LP that starts from the source to the current node, while the LST is obtained from the shortest path (SP) that is back calculated from the destination to the current node, defined as follows, respectively:

$$EST(w_i) = \begin{cases} 0, & \text{if } w_i = w_0 \\ \min\limits_{w_j \in pre(w_i)} \left(EST(w_j) + c(w_j, w_i)\right), & \text{otherwise} \end{cases} \tag{5.3.2}$$

$$LST(w_j) = \begin{cases} EST(w_{m-1}), & \text{if } w_j = w_{m-1} \\ \max\limits_{w_j \in pre(w_i)} \left(LST(w_i) - c(w_j, w_i)\right), & \text{otherwise} \end{cases} \tag{5.3.3}$$

where $pre(w_i)$ is the set of preceding modules of module w_i and $c(w_j, w_i)$ is the total time cost from module w_j to module w_i. For the sake of completeness, we give the pseudocode in Alg. 4 for finding a CP P in G_w.

Algorithm 4 FindCriticalPath(G_w, w_0, w_{m-1})

```
1: for all modules w_i, i = 0,1,...,m-1 do
2:     Initialize the EST_i = 0 and LST_i = ∞;
3: for all modules w_i, i = 0,1,...,m-1 do
4:     Calculate and update EST_i using the LP;
5: for all modules w_i, i = m-1,...,1,0 do
6:     Calculate and update LST_i using the SP;
7: for all modules w_i, i = 0,1,...,m-1 do
8:     if EST_i == LST_i then
9:         Add module w_i to CP P;
10: return P.
```

Critical Path Mapping

We adapt the optimal pipeline mapping algorithm proposed in Chapter 4.2.1 to the CP mapping problem, which uses a 2D table to fill out the final result column-by-column from the substructure. Every cell $T_{\mathrm{ED}}^{j-1}(v_i)$ in the table represents the MED of an optimal mapping solution that maps the first j modules on the CP to a path between the source node v_s and node v_i in the network. Each cell is calculated from the intermediate mapping results up to its adjacent nodes or itself, which are stored in its left column $T_{\mathrm{ED}}^{j-2}(\cdot)$. For details of the mapping algorithm, please refer to Chapter 4.2.1.

Non-critical/Branch Module Mapping

We design a recursive priority-based greedy algorithm to schedule branch modules not located on the CP. We first insert all critical modules into a queue Q. At step i, a module w_i is dequeued from Q and its succeeding modules are assigned to certain nodes in the following way: (i) Sort all unmapped succeeding modules $USM(w_i)$ of module w_i in a decreasing order according to their computation and communication requirements. The modules requiring more computations and communications are assigned higher priorities. (ii) Map all unmapped modules $w \in USM(w_i)$ with decreasing priorities onto either node $v(w_i)$ that runs module w_i or one of its succeeding nodes with the minimum module execution time or data transfer time. (iii) Insert modules $w \in USM(w_i)$ to the end of Q. The

above procedure is recursively performed until Q is empty so that all modules in the workflow are assigned to the network nodes. The pseudocode in Alg. 5 describes the procedure of function $MapNonCriticalModules()$.

Algorithm 5 MapNonCriticalModules(P, G_w, G_c, v_s, v_d)

```
1: Insert all modules on CP P into a queue Q in the same order as they appear on P;
2: i = 0;
3: while Q ≠ ∅ do
4:    w_i = the first module in Q;
5:    Remove w_i from Q;
6:    v(w_i) = the node in G_C that runs w_i;
7:    SN(v(w_i)) = a set of succeeding nodes of node v(w_i);
8:    USM(w_i) = a set of unmapped succeeding modules of module w_i sorted in a deceasing order according to computation and communication requirements;
9:    for all modules w ∈ USM(w_i) do
10:      Map module w to node v(w_i) and calculate time cost C_1;
11:      for all nodes v ∈ SN(v(w_i)) do
12:         Map module w to node v;
13:         Select the node v_min with minimum cost C_2;
14:      Select either v(w_i) or v_min with cost C = Min(C_1, C_2);
15:      Insert module w into the end of Q;
16:   i = i + 1;
17: return the entire mapping assignment;
```

If there is only one module mapped onto one node, we can directly calculate the module execution time. However, if multiple modules are deployed on the same node, the resource is shared among those concurrent modules and the same rule is also applied to bandwidth share. The extED calculation in Chapter 5.2 is designed to calculate the sharing dynamics during execution.

5.3.2 Improved Recursive Critical Path (impRCP) Algorithm

The previous experiments have shown that the mapping performance of RCP largely relies on non-critical module mapping. Thus, we improve the RCP algorithm by designing and incorporating a new non-critical module mapping scheme, referred to as impRCP, based on A^* Search and Beam Search to replace the original naive greedy approach. A^* is a best-first

graph search algorithm that features a path with the least cost from a given source node to a destination node represented by the cost function $f(x) = g(x) + h(x)$, which consists of two cost components: (i) a path cost function, denoted as $g(x)$, computed from the source node to the current node, which may or may not be optimal, and (ii) an admissible heuristic estimation function, denoted as $h(x)$, of the cost from the current node to the destination node. We further employ the Beam Search algorithm to reduce the search complexity of the A^* algorithm by exploring only a predetermined number ρ of least-cost paths in the search tree in the solution space, instead of searching all feasible paths, assuming that the optimal solution is most likely to be found on these paths. An appropriate value of ρ can be decided by an empirical method or based on the actual situation of the application. The pseudocode of the non-critical module mapping algorithm is provided in Alg. 6, whose complexity is $O(\rho \cdot m \cdot n \cdot |E_w|)$, where ρ is the predefined beam width in the Beam Search algorithm, m and $|E_w|$ are the number of modules and edges in the workflow, respectively, and n is the number of nodes in the network.

Algorithm 6 impRCP-NonCriticalModMapping(P, G_w, G_c)
Input: A computing workflow $G_w = (V_w, E_w)$, $|V_w| = m$, a computer network $G_c = (V_c, E_c)$, $|V_c| = n$, and a mapped CP P.
Output: Entire mapping scheme for MED.

1: Insert mapped modules on the CP into a queue Q in a sequential order;
2: **while** $Q \neq \emptyset$ **do**
3: Remove w_i from the head of Q;
4: Prioritize all unmapped direct succeeding modules of w_i from high to low by computational requirements, denoted as $pri(suc(w_i))$;
5: **for all** $w_j \in pri(suc(w_i))$ from high to low **do**
6: Calculate $f(w_j) = g(w_0, w_j) + h(w_j, w_{m-1})$ based on A^* and Beam Search algorithms;
$g(w_0, w_j)$ = the total time cost of mapped modules from w_0 to w_j + the execution time of w_j on v_k;
$h(w_j, w_{m-1})$ = the total time cost of ρ least-cost paths from w_j to w_{m-1} mapped from v_k to v_d based on a greedy procedure;
7: Select the node that produces $f(w_j)_{min}$ to map module w_j;
8: $Q \leftarrow w_j$;

5.3.3 Decentralized RCP (disRCP) Algorithm

Workflow mapping has been traditionally done in a centralized manner. However, as computing, networking, and storage resources are continuously developed and deployed around the world, maintaining the global resource information at a central location becomes prohibitively expensive, and in the worst case, the resource status might have already changed even before the central server gets to run the mapping algorithm. Therefore, the scalability requirement for workflow mapping in large-scale networks calls for a distributed solution. Moreover, a distributed workflow mapping algorithm is able to confine the changes of a mapping scheme in a local area and therefore obviate the need of reproducing and redeploying the entire mapping scheme in the global scope if only a small portion of nodes or links break down.

The original RCP algorithms are centralized algorithms that require a central server. The central server must have full knowledge of the computing workflow and computer network in terms of topology and parameters, based on which, it calculates the best mapping scheme for mapping the workflow to the network. Collecting the global status information and maintaining it at a central location could be prohibitively expensive if not at all possible, especially in shared wide-area networks with unpredictable system dynamics. We propose a distributed version of the RCP algorithm, referred to as disRCP, where each node only needs to identify and exchange information with its neighbor nodes. Each node maintains four local tables:

- *Workflow Table* (WT) is an $m \times m$ 2D table that stores entire workflow topology and parameters.
- *Neighbor Node Table* (NNT) is an array with a varying length between 1 and n that stores neighbor nodes.
- *Critical Module Mapping Table* (CMMT) is an array with a varying length between 1 and m that stores partial mapping results of all critical modules on the current node.

- *Branch Module Mapping Table* (BMMT) has the same size as CMMT and stores the temporary mapping results for non-critical modules mapped on the current node.

Since we assume that a branch/non-critical module cannot be mapped on a node which has already been selected for mapping a critical module to ensure the ED performance of the CP, CMMT and BMMT will not be used simultaneously on the same node. Each mapping node also maintains two variables, namely EST and LST for each of the modules mapped on that node, which are used in the CP calculation.

At the initialization stage, the workflow information is sent to each neighbor node in the network and stored in the WT. Each node sends out a "HELLO" message containing its own node ID to all its neighbors in the network. Once a node receives such a message, it appends its own node ID at the end of the message and sends it back to the sender. Upon the receiving of a reply message, the node updates the neighbor information in its NNT including node ID, computing power, link BW and MLD between them. All other parameters are set to be $NULL$ initially. The workflow mapping process starts at the source node and the decentralizing procedure of disRCP is described as follows.

Finding CP

During this stage, the CP of the workflow is computed based on the previous round of mapping scheme stored in the CMMT or BMMT of each mapping node. Starting from the source node v_s on which the first module w_0 is mapped, each mapping node sets the EST to be 0 and calculates the total time cost (transfer time $t_{s,i}$ and module execution time t_i) to reach each of its succeeding modules w_i using Eq. 5.3.2 and sends the results in a message to the corresponding succeeding nodes. After receiving this message, the node updates the corresponding ESTs of its mapped modules and computes the total time cost for succeeding neighbor modules. This process is repeated until the destination node v_d is reached. Here, a node has two options for updating ESTs: (i) if the EST of the current succeeding module is $NULL$, set the EST to be the received value; (ii) otherwise, compare

the received value and the original one, and set the EST to be the minimum. Similarly, the LSTs of the mapped modules on each mapping node can be computed using Eq. 5.3.3 in a reversed direction from the destination node to the source node.

To identify the critical modules to compose the CP, the source node sends a message to all its succeeding nodes querying "I have the critical module w_0; who has the succeeding critical module mapped?" Each succeeding node compares the ESTs and LSTs of all its mapped modules. If node v_j finds a mapped module w_i with the same EST and LST, it replies to the source node with a message saying "my node ID is j; I have the critical module w_i mapped", and meanwhile sends a message to each of its own succeeding nodes querying "I have the critical module w_i; who has the succeeding critical module mapped?" This query-reply process is repeated until the last module w_{m-1} on the destination node v_d is reached. At this point, the CP has been completely established in the network. Note that an original CP in the workflow is obtained on each node by only considering the workflow under the assumptions of network completeness and resource homogeneity.

The nodes on the CP compute the critical module execution time and data transfer time, and send the results to the succeeding node along the CP for integration. The MED is finally produced on the destination node and compared to the MED of the mapping scheme determined in the previous round. If the termination condition is met, the mapping process stops; otherwise, a new mapping round is performed.

Mapping CP

We adapt the optimal pipeline mapping algorithm based on a DP procedure in Chapter 4.2.1 for MED with arbitrary node reuse by developing a distributed mapping process. The centralized DP procedure employs a 2D table on the central server, where the modules are listed in order along the horizontal dimension and the nodes are arranged along the vertical dimension. A cell in the table represents an optimal mapping solution that maps the first modules in the pipeline (i.e., CP) to a network path between the source and the current node

in the network. The value of each cell is calculated from the intermediate mapping results up to its preceding nodes or itself.

In the disRCP algorithm, the CMMT on each node corresponds to an entire row of the DP table in Fig. 4.6, which stores the partial result for each critical module mapped on the current node. At the beginning, the CMMTs are all initialized to be $NULL$. The first cell of the CMMT on the source node v_s with the first module mapped is filled out and the ED is set to be 0. The rest of the procedure starts from the second module on the CP:

- Each node repeatedly checks its CMMT to find the first $NULL$ cell to fill starting from the second cell.
- If the $NULL$ cell is the second one in the CMMT, go to the next step directly; otherwise, map the current module on itself and compute the ED based on the partial mapping result in the preceding cell of the CMMT on the current node. Fill in the cell with ED and the current mapping node ID. This is corresponding to the horizontal arrows in each row in Fig. 4.6.
- Send a message to all of its preceding neighbor nodes querying "Is the corresponding preceding cell in the CMMT on any node not $NULL$?" If yes, the neighbor node replies to the current node with the partial mapping result in that cell. The current node then maps the current module on itself and computes the ED based on each of the partial mapping results received. It compares the EDs and selects the minimum one among them, which is further compared with the one in the current cell of the CMMT on the current node. The minimum one is selected to fill in the current cell of the CMMT on the current node and update the preceding mapping node ID. This is corresponding to the slanted arrows across different rows in Fig. 4.6.

The above steps are repeated until the last cell of the CMMT on the destination node v_d is filled out. The mapping results of the critical modules can be retrieved by backtracking from the destination node.

Mapping Non-critical Modules

Here, we describe the decentralized process of the priority-based non-critical module mappings. Each critical node with critical module(s) mapped performs the following:

- Assign the priority to the unmapped succeeding modules of each critical module based on WT (modules with higher computation and transmission demands have higher priorities);
- Map those unmapped succeeding modules, from high to low priority, to one of its succeeding neighbor nodes with the minimum ED considering resource share;
- Send a message to update the BMMT on each mapping neighbor node.

After the receiving of this message, a node without critical modules mapped follows the same process as a critical node except for one difference in Step (ii): it not only maps to its succeeding neighbor nodes, but also itself, and selects the minimum one as the final result. All the branch modules are mapped on the nodes eventually. The mapping scheme of the entire workflow is then stored in the CMMTs and BMMTs on those corresponding nodes. The condition in the step of finding CP is checked to continue or terminate the mapping process.

5.4 Mapping Solution to MFR

The workflow mapping problem with arbitrary node reuse for MFR is formally defined as follows:

Definition 7. *Given a DAG-structured computing workflow $G_w = (V_w, E_w)$ and a heterogeneous computer network $G_c = (V_c, E_c)$, we wish to find a mapping scheme that assigns each computing module to a network node such that the mapped workflow achieves:*

$$\text{MFR} = \max_{\text{all possible mappings}} \left(\frac{1}{T_{\text{BT}}} \right). \tag{5.4.1}$$

Note that MFR is achieved by identifying and minimizing the time T_{BT} on a global bottleneck, one module execution time component or one data transfer time component, among all possible mappings. We maximize FR to produce the smoothest data flow in streaming applications when datasets are continuously generated and fed into the workflows.

We conduct a rigorous workflow stability analysis and develop a Layer-oriented Dynamic Programming (LDP) solution based on topological sorting to identify and minimize the global BT, for which, a distributed version is provided as well.

5.4.1 Workflow Stability Analysis

Besides the difficulty of end-to-end performance optimization, many runtime performance issues also remain open due to the complex execution dependencies between computing modules, which are further complicated by the highly random network, host, and user dynamics in distributed environments. Before developing mapping algorithms for MFR, we need to determine whether or not a distributed workflow can stabilize and predict how long it takes to stabilize. A module stabilizes if it reaches a steady state where outputs are produced at a constant rate, and the entire workflow system stabilizes if the last module stabilizes and produces final outputs at a constant rate. Such stability performance guarantee is particularly important for streaming applications that require MFR for achieving smooth data flow.

Without loss of generality, we first convert the data transfer time of each dependency edge to a virtual module execution time as we did in Chapter 5.2.2, where we replace each dependency edge in the computing workflow with a virtual module whose computational requirement is equal to the corresponding transferred data size and replace each mapped network link with a virtual node whose processing power is equal to the corresponding BW. As such, no data transfer time needs to be considered.

In streaming applications, even dependent modules mapped to the same node could run concurrently to process different instances of input datasets continuously entering the workflow. We assume that the computing resources of one node are shared by all the modules mapped to it in a fair manner, and further convert the workflow mapping problem with arbitrary node reuse to an equivalent mapping problem with no node reuse (i.e., one-to-one mapping) by dividing a reused node to several virtual nodes with the processing power divided by the number of modules on that node. Again, we use $G_w{}'(E_w{}', V_w{}')$ to represent the resultant workflow. We have Theorem 5 on the stability of a one-to-one mapped workflow processing time-series input datasets in streaming applications.

Theorem 5. *A one-to-one mapped workflow stabilizes after* $\max(T_i)$ *time steps, where* T_i *is the execution time of module* $w_i \in V_w{}'$.

Proof. The entire workflow system will stabilize if we can prove that any module stabilizes at a certain point when all its preceding (incident or input) modules stabilize, since the system execution starts from the first module, which does not have any input dependency and hence is always in steady state. For convenience, we introduce the following notations:

- sn_i: the first stable dataset sequence number after which module w_i produces outputs at a constant rate.
- st_i: the stable time step (interval) between two adjacent outputs produced by module w_i in steady state.
- $t_{w_i,\,k}$: the time point when module w_i finishes processing the k-th dataset.
- T_j: the execution time cost of module w_j.
- $pre(w_j)$: the set of preceding modules of module w_j (i.e., modules that are incident to module w_j).

For the first dataset, we have $t_{w_j,1} = \max\limits_{w_i \in pre(w_j)} (t_{w_i,1}) + T_j$. For the rest datasets $k > 1$, since the module execution starting time depends on both the completion time of the

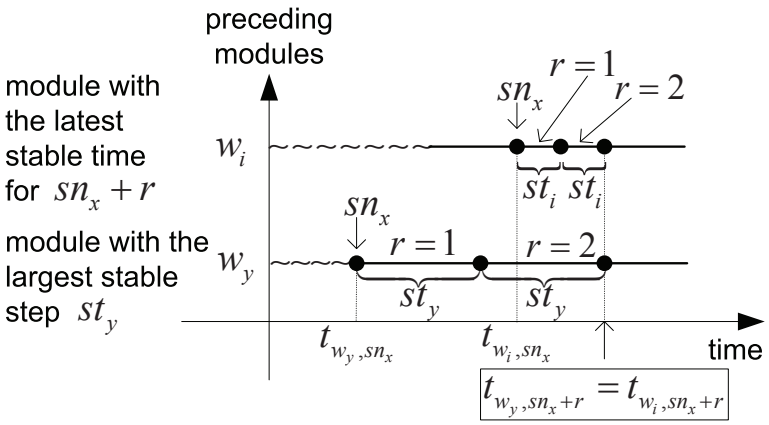


Figure 5.6: Illustration of steady state analysis.

previous dataset on module w_j and the latest arrival time of the current dataset among all preceding modules, we have

$$t_{w_j,k} = \max(t_{w_j,k-1}, \max_{w_i \in pre(w_j)}(t_{w_i,k})) + T_j. \tag{5.4.2}$$

Suppose that we obtain sn_i and st_i for each of these preceding module w_i, $w_i \in pre(w_j)$. The largest steady state dataset sequence number sn_x and the largest steady state time step st_y among all preceding modules are defined as: $sn_x = \max_{w_i \in pre(w_j)}(sn_i)$ and $st_y = \max_{w_i \in pre(w_j)}(st_i)$, respectively. Obviously, starting from dataset sequence number $sn_x + 1$, all preceding modules produce outputs at their own constant rates. For module w_j, we consider two cases:

- Case 1 when $st_y \leq T_j$: The bottleneck time is the execution time of module w_j, hence $sn_j = 1$ and $st_j = T_j$.

- Case 2 when $st_y > T_j$: According to Eq. 5.4.2, the time when w_j finishes processing dataset $sn_x + r$, $r \geq 0$, can be calculated as $t_{w_j,sn_x+r} = \max(t_{w_j,sn_x+r-1}, \max_{w_i \in pre(w_j)}(t_{w_i,sn_x+r})) + T_j$. Since the bottleneck time is the execution time of the slowest preceding module, we have $t_{w_j,sn_x+r} = \max_{w_i \in pre(w_j)}(t_{w_i,sn_x+r}) + T_j = \max_{w_i \in pre(w_j)}(t_{w_i,sn_x} + r \cdot st_i) + T_j$. As shown in Fig. 5.6, after dataset $sn_x + r$ is completed, the preceding module w_y with the largest stable time step st_y will always be the latest (slowest) one to send any new

dataset to module w_j. We derive that module w_j stabilizes with $st_j = st_y$ after the dataset sequence number $sn_x + r$, where $r = \left\lceil \max_{w_i \in pre(w_j)} \left(\frac{t_{w_i,sn_x} - t_{w_y,sn_x}}{st_y - st_i} \right) \right\rceil$.

From the above two cases, we conclude that the stable time step of module w_j is

$$st_j = \max(T_j, \max_{w_i \in pre(w_j)} st_i). \tag{5.4.3}$$

If we recursively apply Eq. 5.4.3 to the entire workflow until we reach the last module, we obtain the stable time step of the workflow system as: $st_{\text{system}} = \max(T_i),\ w_i \in V_w{}'$, which essentially is the global bottleneck time of the entire workflow. □

5.4.2 Layer-oriented Dynamic Programming (LDP) Algorithm

We develop a *Layer-oriented DP* algorithm, referred to as LDP, which maps a topologically sorted computing workflow to a computer network for MFR by identifying and minimizing the global BT. We introduce the following notations to facilitate the description of our mapping algorithm:

- $pre(w_j)$ denotes the set of preceding modules of module w_j (i.e., modules that are incident to module w_j;
- $V_{\text{one-mapping}}(pre(w_j))$ represents the set of nodes that make up one possible mapping of those modules in $pre(w_j)$;
- $suc(v_i)$ denotes the set of succeeding nodes of node v_i;
- $\bigcap_{\text{for all } v \in V_{\text{one-mapping}}(pre(w_j))} (suc(v))$ is an intersection operation that finds the set of common succeeding nodes, denoted as $V_{\text{candidate}}(w_j)$, corresponding to one particular mapping;
- $\bigcup_{\text{for all mappings}}$ represents a union operation on those common succeeding nodes of all possible mapping combinations.

Let $T_{i,j}$ ($0 \leq i \leq n-1,\ 0 \leq j \leq m-1$) denote the global BT of a topologically sorted workflow from module w_0 to w_j mapped to a network from node v_0 to v_i. We have the following recursion leading to the final solution $T_{n-1,m-1}$ by filling out a two-dimensional (2D) DP table:

$$T_{i,j} \underset{j=1 \text{ to } m-1, \text{ and } v_i \in \bigcup_{\text{for all mappings}} \left(V_{\text{candidate}}(w_j)\right)}{=} \max_{\substack{w_u \in pre(w_j), \\ 0 \leq h \leq n-1}} \left(\begin{array}{ll} \max \left(\begin{array}{l} T_{h,u}, \\ \sum\limits_{t=t^s_{e_{u,j}}}^{t^f_{e_{u,j}}} \frac{\beta_{u,j}(t) \cdot \delta_{e_{u,j}}(t)}{b_{h,i}} + d_{h,i}, \\ \sum\limits_{t=t^s_{w_j}}^{t^f_{w_j}} \frac{\alpha_j(t) \cdot \delta_{w_j}(t)}{p_i} \end{array} \right), & \text{if } h \neq i \\ \max \left(\begin{array}{l} T_{h,u}, \\ \sum\limits_{t=t^s_{w_j}}^{t^f_{w_j}} \frac{\alpha_j(t) \cdot \delta_{w_j}(t)}{p_h} \end{array} \right), & \text{if } h = i \end{array} \right). \tag{5.4.4}$$

The base condition, i.e., $T_{0,0} = 0.0$, is determined by the assumption that the first module be mapped to the source node and not perform any computing. Note that Eq. 5.5.6 may not fill up all the cells in the DP table due to the module dependency and network topology constraints.

The key idea of LDP is to first sort a DAG-structured workflow in a topological order and then map computing modules to network nodes on a layer-by-layer basis, while taking into consideration both module dependency in the hworkflow and node connectivity in the network. As illustrated in Fig. 5.7, in a topologically sorted workflow, modules with the same number of hops on their longest paths from the source module v_s are placed in the same layer, where there does not exist any dependency. We further sort the modules in the same layer based on their priority, which is determined by their computational requirements, i.e., the most "needy" module has the highest priority.

We use a 2D DP table to record the intermediate mapping result at each step where a module is mapped layer-by-layer to a strategically selected network node. For a better clarity, we decompose the table into several separate tables, each of which represents one

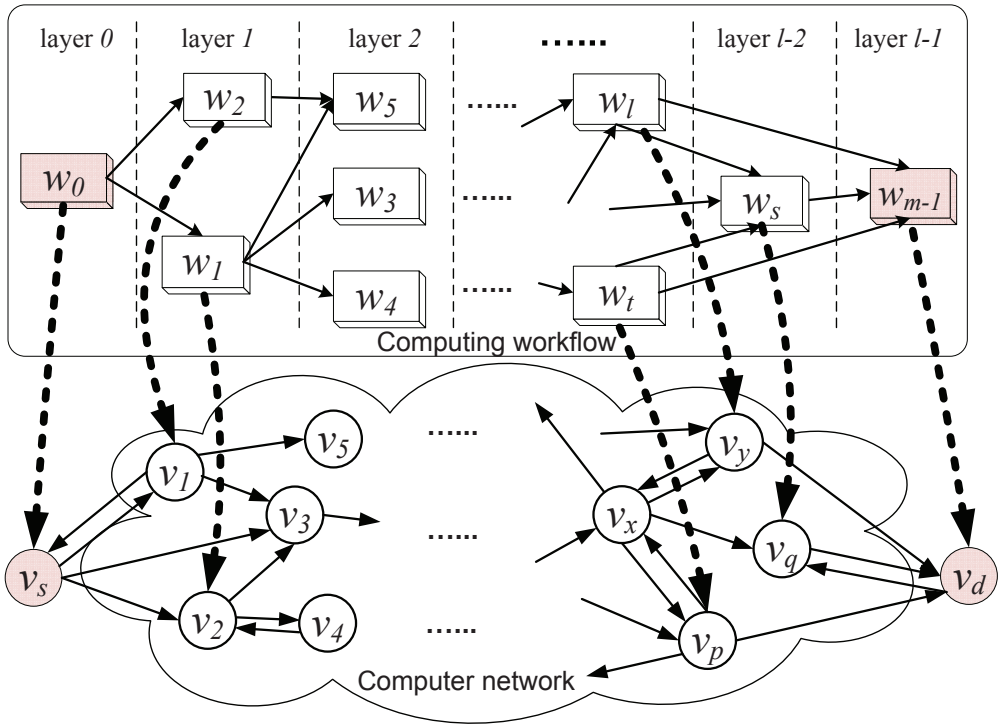


Figure 5.7: Layered workflow in a topological sorting.

topological layer, and calculate the columns corresponding to those modules sorted by their priorities on each layer. Fig. 5.8 illustrates the mapping procedure of LDP for the workflow mapping problem shown in Fig. 5.7. In Fig. 5.8, the horizontal coordinates represent the sequence numbers of topologically sorted modules (using the priority to break a tie within the same layer), and the vertical coordinates represent the labels of network nodes starting from the source node v_s to the destination node v_{n-1}.

The pseudocode of LDP is provided in Alg. 7. The workflow is topologically sorted in line 3. In lines 5-8, we select an unmapped module w_j with the highest priority on the current layer, find all its preceding modules $pre(w_j)$ and one possible set of their mapping nodes $V_{\text{one-mapping}}(pre(w_j))$, and determine the set of common succeeding nodes of all nodes $v \in V_{\text{one-mapping}}(pre(w_j))$ using an intersection operation. For the mapping of module w_j, LDP considers all possible combinations of mapping nodes $V_{\text{one-mapping}}(pre(w_j))$ for the set of preceding modules. If a module in $pre(w_j)$ has (i) only one succeeding

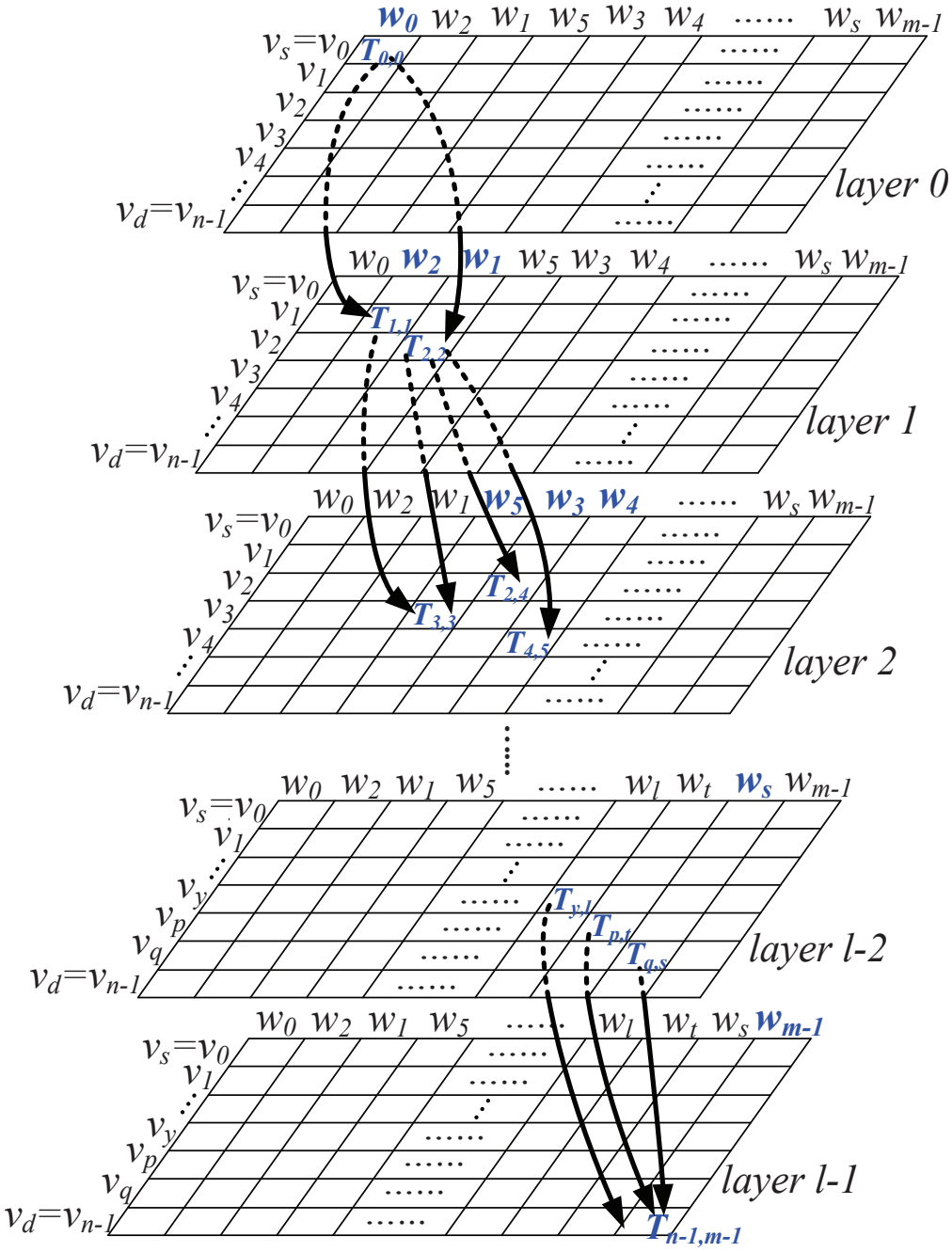


Figure 5.8: The DP table with separated layers in LDP.

Algorithm 7 LDP(G_w, G_c, v_s, v_d)
Input: A workflow graph $G_w = (V_w, E_w)$, $|V_w| = m$, a network graph $G_c = (V_c, E_c)$, $|V_c| = n$, a pair of source and destination nodes (v_s, v_d).
Output: MFR of the workflow.

1: $pre(v_i)$: the set of preceding nodes of node v_i;
2: $v(w_j)$: the node to which module w_j is mapped;
3: Topologically sort workflow G_w into $G_w{}'$ of l layers: lay_k, $k = 0, 1, \cdots, l-1$;
4: **for all** layer lay_k, $k = 1, 2, \cdots, l-1$ **do**
5: **for all** "unmapped" module $w_j \in lay_k$ **do**
6: Select w_j with the highest priority $pri(w_j)$;
7: Find $pre(w_j)$ and $V_{\text{one-mapping}}(pre(w_j))$;
8: Find set $V_{\text{candidate}}(w_j)$ of candidate nodes for module w_j;
9: $T_{x,j} =$ MAX_DOUBLE;
10: **for all** node $v_i \in \bigcup_{\text{for all mappings}} \left(V_{\text{candidate}}(w_j)\right)$ **do**
11: **for all** module $w_u \in pre(w_j)$ **do**
12: h = node ID of $v(w_u)$;
13: **if** $h \neq i$ **then**
14: $T = \max\left(T_{h,u}, \sum_{t=t^s_{e_{u,j}}}^{t^f_{e_{u,j}}} \frac{\beta_{u,j}(t)\cdot\delta_{e_{u,j}}(t)}{b_{h,i}} + d_{h,i}, \sum_{t=t^s_{w_j}}^{t^f_{w_j}} \frac{\alpha_j(t)\cdot\delta_{w_j}(t)}{p_i}\right)$;
15: **else**
16: $T = \max\left(T_{h,u}, \sum_{t=t^s_{w_j}}^{t^f_{w_j}} \frac{\alpha_j(t)\cdot\delta_{w_j}(t)}{p_h}\right)$;
17: **if** $T_{i,j} < T$ **then**
18: $T_{i,j} = T$;
19: **if** $T_{x,j} > T_{i,j}$ **then**
20: $T_{x,j} = T_{i,j}$;
21: $x = i$;
22: **return** MFR of the workflow;

module, we consider all available nodes in the current column to map this module; (ii) more than one succeeding modules, we select the node that results in the minimum BT for mapping this module to ensure mapping feasibility. In lines 10-21, we map module w_j to each of those candidate mapping nodes and compute their corresponding global BTs. The complexity of this algorithm depends on the number of "all possible combinations for $V_{\text{one-mapping}}(pre(w_j))$", and could lead to an expensive search process in some extreme cases, which necessitates a greedy version of LDP to reduce the complexity.

In LDP, at each step, the node considers all possible combinations of the mapping nodes for the preceding modules to determine their common succeeding (candidate) nodes that can be used to map the current module in each column, which could be computationally intensive especially when there are a large number of combinations. To reduce the search complexity, we propose *Greedy LDP*, which selects the best node in each column that yields the minimum global BT for mapping the current module, hence filling one cell only in each column as shown in Fig. 5.8. In Greedy LDP, we recursively calculate the minimum bottleneck of the sub-solution that maps the subgraph (a partial workflow) consisting of the current module w_j and all the modules before the current layer to the network until the last module w_{m-1} is mapped to the destination node v_d, which is represented by the right bottom cell in the DP table. The complexity of Greedy LDP is $O(l \cdot n \cdot |E_w|)$, where l is the number of layers in the topologically sorted workflow, n is the number of nodes in the network, and $|E_w|$ is the number of edges in the workflow.

5.4.3 Decentralized LDP (disLDP) Algorithm

The disLDP algorithm does not require a central server to collect the global information, which is distributed among individual nodes. Each node maintains three local tables:

- *Local Resource Table* (LRT), which stores the neighbor connectivity information of the current node;
- *Sorted Workflow Table* (SWT), which stores the modules in the workflow sorted in a topological order together with their dependencies;
- *Local Mapping Table* (LMT), which stores the local mapping information of the current node during the mapping process.

In the initialization phase, the workflow information is sent to all the nodes and stored in their SWT. Both LRT and LMT on each node are set to be *NULL*. To obtain the neighbor connectivity information, each node broadcasts a "hello" message to the network and

updates its own LRT upon the receipt of its direct neighbors' messages. The LMT of node v_i is essentially the i-th row in the 2D DP table combining all the layers together in Fig. 5.8, where each cell stores (i) the temporary bottleneck time T_{BT} under the current partial mapping scheme and (ii) the incident dependencies from the preceding nodes, on which the preceding modules are mapped.

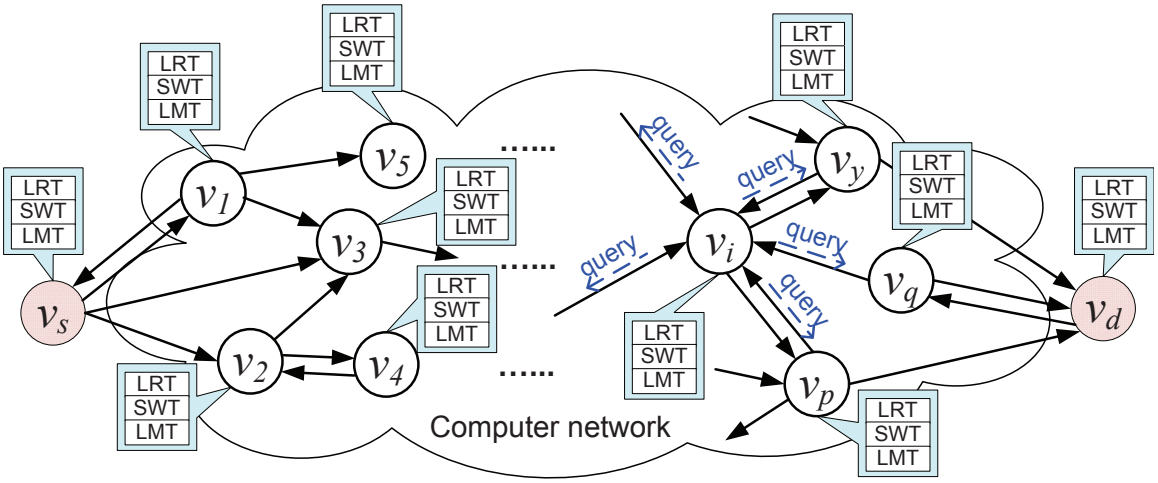


Figure 5.9: Decentralizing the mapping process of disLDP.

The decentralized mapping process starts from the source node to which the first module is mapped. As the base condition, for the first module, the first cell of the LMT on each node is set to be *NULL*, except for the source node where the first cell of the LMT is set to be zero. Once the mapping process begins, each node repeatedly checks its LMT to locate the next unfilled cell. To illustrate this process, let us consider node v_i, $i \in [1, n-1]$, which needs to fill out the j-th cell, $j \in [1, m-1]$, in its LMT (i.e., map the j-th module on node v_i). Suppose that module w_j has two preceding modules w_h and w_k in the workflow. In Fig. 5.9, node v_i sends a query to all its upstream neighbors (including v_i itself) asking "who has w_h mapped" and a similar query asking for the mapping node of module w_k. The node that has either w_h or w_k mapped on it sends v_i a reply message that carries information on the h-th or k-th cell in its LMT. Upon the receipt of all reply messages, v_i considers the following three cases:

1. there is no reply at all to either query;

2. there is only one reply to each of the queries;

3. there are multiple replies to either one query or both queries.

Case 1 means that none of its upstream nodes has either w_h or w_k or both mapped. In this case, module w_j cannot be mapped to node v_i because the module dependency is not satisfied, and node v_i simply sets the j-th cell in its LMT to be $NULL$. Case 2 means that if module w_j is mapped to node v_i, there is only one option to map its preceding modules w_h and w_k. In this case, node v_i calculates the temporary global bottleneck time T_{BT} based on the partial mapping solution from the preceding node(s) on which modules w_h and w_k are mapped, and updates the j-th cell in its LMT using the calculated T_{BT} as well as the dependencies. Case 3 means that there are multiple options to map modules w_h and w_k if module w_j is mapped on node v_i. In this case, node v_i considers all possible mapping combinations for modules w_h and w_k, and for each mapping option $V_{\mathrm{one-mapping}}(pre(w_j))$, node v_i performs a similar calculation as in Case 2. Once all mapping options have been considered, node v_i compares the temporary bottlenecks of all these mapping combinations of modules w_h and w_k, and then selects the minimum one to fill in the j-th cell of its LMT. Each node progressively fills out its corresponding row of cells along the horizontal direction in the 2D DP table by computing the BT of each partial mapping scheme that ends at the current node. Note that some queried cells of the LMTs on the upstream nodes may not have already been filled out when a query is sent. However, after several rounds of queries, the mapping process eventually reaches the last module and then all the cells of the LMT on each node in the network must have been either filled out or marked as $NULL$.

The final mapping scheme for MFR can be established in the network by backtracking the nodes' LMTs, starting from the destination node: it sends a message containing the module sequence number to each of its preceding nodes recorded in the $(m-1)$-th cell of its LMT, and those preceding nodes then check the mapping information in the corresponding

cells of their LMTs and send messages to their preceding nodes. This backtracking process continues until the source node (or the first module) is reached. Once the mapping is completely determined in the network and the modules are deployed on the selected nodes, the workflow can start execution immediately.

We decentralize Greedy LDP based on a similar decentralization process used for LDP. The only difference between them is that in Case 3 of Greedy disLDP, node v_i does not need to consider all mapping combinations for those preceding modules. Instead, it selects the best one for each preceding module to calculate the current cell, which significantly speeds up the search process. Here, the "best one" refers to the computer node that incurs the minimum global bottleneck time among all possible mapping nodes for executing the preceding module under consideration.

5.5 Workflow Optimization under Fault-tolerance Constraint

5.5.1 Overview

In network environments especially with resource sharing, node and link failures are inevitable and may have a detrimental impact on the workflow performance. Fault tolerance in workflow mapping/scheduling has been the focus of research in the literature [35, 59, 76, 86], and several approaches have been proposed including task duplication and checkpoint-restart. Task duplication is a commonly adopted conservative approach where each task is replicated multiple copies and executed on different nodes to minimize the probability of failures [35, 76, 117]. However, this approach significantly increases network traffic and requires more computing resources. Alternatively, a checkpoint-restart approach avoids task duplication by automatically checkpointing the process and restarting it when a failure occurs [28, 39]. Unfortunately, this approach may also increase extra overhead and slow down the execution by the restart mechanism. These approaches consider either remedy

or recovery in the case of resource failures during the course of workflow execution. Orthogonal to the above ones, another approach focuses on finding a good mapping scheme that allocates tasks with or without precedence constraints to relatively more reliable machines [61, 62, 94, 95, 122]. This mechanism attempts to minimize the chance of failure occurrences early in the workflow mapping phase, which is the focus of our work.

Note that a mapping scheme that achieves maximum reliability may lead to an unbearably long execution time. These conflicting requirements make it extremely challenging to tackle the problem of optimizing multiple objectives simultaneously [36, 63, 76, 85]. Some existing research efforts only consider one single objective, either latency minimization [125] or reliability maximization [95]. For multi-objective optimization problems, a commonly used strategy is to either combine multiple objectives into a single criterium, or optimize one objective while treating others as constraints. One solution using the combination approach was presented in [62], where the failure cost function is incorporated into an existing compile time. Another solution using the constrained approach was proposed in [36] to optimize latency under throughput and reliability constraints.

We construct the cost models for workflows and networks subject to probabilistic node and link failures, and rigorously prove that the bi-objective optimization problem cannot be approximated within one constant factor. Considering the Pareto dominance with two potentially conflicting objectives, we further convert this problem to a reliability-constrained optimization problem. The rapid increase in the workflow and network scale discourages centralized management and aggravates node and link failures, which necessitates the decentralization of mapping solutions with fault tolerance for practical deployment. We adapt the proposed disRCP and disLDP workflow mapping solutions in Chapters 5.3 and 5.4, respectively, to the workflow mapping problems under a given reliability constraint.

5.5.2 Fault Models

We further define the fault models for workflow mapping in faulty computer networks based on the fault-free models described in Chapter 3. We consider both nodes and links as resources denoted by $R = V_c \cup E_c = \{r_i | i = 0, 1, \cdots, n + |E_c| - 1\}$. In a faulty network where each resource $r_i \in R$ may fail at a certain probabilistic rate, we assume that the number of failure occurrences of resource r_i in a certain time span follows a Poisson distribution and the probability that r_i is functional (or reliable) during a time period t is modeled by an exponential function:

$$\mathscr{P}(r_i, t) = e^{-c_i \cdot t}, \tag{5.5.1}$$

where coefficients c_i are nonuniform due to the heterogeneity of nodes and links. Conversely, the failure probability of r_i during a time period t is:

$$1 - \mathscr{P}(r_i, t) = 1 - e^{-c_i \cdot t}. \tag{5.5.2}$$

Our fault model assumes that: (i) resources are fail-silent and subject to transient failures, and (ii) resource failures at different locations are statistically independent. Although Poisson process is widely used to model the failures in faulty systems [62,76], it may not always reflect the actual failure dynamics. However, the work in [114] shows that this assumption is reasonable and results in practically useful mathematical models [62].

The reliability $\mathscr{R}$ of a workflow denotes the probability that all mapping nodes and links are functional during execution such that the workflow completes successfully:

$$\mathscr{R} = \prod_{r_i \in R} \mathscr{P}(r_i, t). \tag{5.5.3}$$

A mapping scheme is fault-free if no node or link fails during execution, i.e., $\mathscr{R} = 1$, which is a very strong condition and is not may not always be achievable in a large, shared, distributed, and heterogeneous environment. A small portion of resources are allowed to fail due to some fault-tolerance mechanisms, such as task replication [35,76,86]. Hence, we consider the Overall Failure Rate (OFR), denoted by $\mathscr{F} = 1 - \mathscr{R}$, which is the probability

that the entire workflow breaks down due to the failure of any individual node or link. We assume independent and nonuniform failure rates for both nodes and links.

5.5.3 Non-approximability of Bi-objective Mapping Problems

We first provide a brief introduction to the approximation factor for mono-objective and multi-objective problems.

Definition 8. *For mono-objective optimization, S is an ε-approximation solution to a problem $\mathscr{I}$ for minimizing or maximizing the value/cost function $f(x)$, if and only if the following inequalities hold:*

$$f^*(\mathscr{I}) \leq f(S) \leq \varepsilon \cdot f^*(\mathscr{I}), \text{ when } \varepsilon > 1,$$
$$\varepsilon \cdot f^*(\mathscr{I}) \leq f(S) \leq f^*(\mathscr{I}), \text{ when } \varepsilon < 1,$$

where $f^*(\mathscr{I})$ is the optimal (minimum or maximum) value of $f(x)$ among all the solutions to $\mathscr{I}$.

For multi-objective optimization, the definition is extended in order to take into account multiple value/cost functions. A solution S is an $\varepsilon = (\varepsilon_0, \varepsilon_1, \cdots, \varepsilon_i)$-approximation of a problem $\mathscr{I}$ for minimizing or maximizing the functions $f(x) = (f_0(x), f_1(x), \cdots, f_i(x))$, if and only if for $\forall i$, the following inequalities hold:

$$f_i^*(\mathscr{I}) \leq f(S) \leq \varepsilon_i \cdot f_i^*(\mathscr{I}), \text{ when } \varepsilon_i > 1,$$
$$\varepsilon_i \cdot f_i^*(\mathscr{I}) \leq f(S) \leq f_i^*(\mathscr{I}), \text{ when } \varepsilon_i < 1,$$

where $f_i^*(\mathscr{I})$ is the minimum or maximum value of $f_i(x)$ among all the solutions of $\mathscr{I}$.

We present the non-approximability result of the original bi-objective mapping problem for maximizing FR and reliability. A similar proof can be found in [63], where they tackled a problem of minimizing makespan and maximizing reliability, which is similar to the problem of optimizing MED and OFR in our problem definition.

Theorem 6. *The bi-objective problem of maximizing both FR and reliability is non-approximable within one constant factor.*

Proof. We consider a simple problem with only one module with computational requirements $\lambda(z)$ in the workflow, and two nodes v_1 and v_2 with computing power $p_1 = \lambda(z)$ and $p_2 = \kappa \cdot \lambda(z)$, $\kappa \in \mathbb{R}^+$, and coefficient $c_2 = \kappa^2 \cdot c_1$ in the network. There exist two mapping solutions to this problem instance.

- S1: map the module to v_1, resulting in $MFR_1 = 1/(\lambda(z)/p_1) = 1$, $\mathscr{R}_1 = \mathscr{P}(r_1, t_1) = e^{-c_1 \cdot (\lambda(z)/p_1)} = e^{-c_1}$;
- S2: map the module to v_2, resulting in $MFR_2 = 1/(\lambda(z)/p_2) = \kappa$, $\mathscr{R}_2 = \mathscr{P}(r_2, t_2) = e^{-c_2 \cdot (\lambda(z)/p_2)} = e^{-c_1 \cdot \kappa}$.

Note that S1 is optimal for reliability while S2 is optimal for MFR. $MFR_2/MFR_1 = \kappa$, which leads to infinity. Thus, S1 is not a constant factor approximation solution in this case. Similarly, $\mathscr{R}_1/\mathscr{R}_2 = e^{c_1(k-1)}$, which also leads to infinity, thus S2 is not a constant factor approximation solution, either. Therefore, neither of these two solutions can approximate both objectives within one constant factor. □

Theorem 6 justifies the necessity of converting the original bi-objective workflow mapping problem to a constrained mono-objective workflow mapping problem.

5.5.4 Problem Formulation and Approaches

We investigate a bi-objective mapping problem to optimize ED/ FR and reliability. Due to the Pareto dominance with these two potentially conflicting objectives, we convert it to a reliability-constrained delay/throughput optimization problem:

Definition 9. *Given a DAG-structured computing workflow $G_w = (V_w, E_w)$, a heterogeneous faulty computer network $G_c = (V_c, E_c)$ where each node or link is associated with a certain failure rate, and a bound $\mathbb{F}$ on the OFR, we wish to find a workflow mapping scheme*

such that the mapped workflow achieves MED/MFR under the reliability constraint $\mathscr{F} \leq \mathbb{F}$, represented as:

$$\min_{\text{all possible mappings}} T_{\text{ED}}, \quad \text{such that} \quad \mathscr{F} \leq \mathbb{F}; \quad \text{or} \tag{5.5.4}$$

$$\max_{\text{all possible mappings}} \left(\frac{1}{T_{\text{BT}}} \right), \quad \text{such that} \quad \mathscr{F} \leq \mathbb{F}. \tag{5.5.5}$$

We tackle these two optimization problems by modifying and adapting the disRCP and disLDP algorithms to the faulty network environment.

As for MED of unitary processing applications, we propose a distributed heuristic approach, *Recursive Critical Path with Fault-tolerance Constraint* (disRCP-$\mathscr{F}$), which is similar to RCP in the fault-free environments except that the partial OFR is checked against the global constraint at each DP step as we select an appropriate computer node to map the current module. If the partial failure rate is larger than the bound $\mathbb{F}$, we recalculate the current mapping and search for an alternative node that satisfies the bound to map the module.

As for MFR of streaming applications, we modify the Eq. 5.5.6 in Chapter 5.4 as follows to facilitate the BT calculation under OFR constraint.

$$T_{i,j} = \bigcup_{\substack{j=1 \text{ to } m-1,\ v_i \in \\ \text{for all mappings}}} \left(V_{\text{candidate}}(w_j) \right)$$

$$\max_{\substack{w_u \in pre(w_j), \\ 0 \leq h \leq n-1}} \begin{cases} \max \left(\begin{array}{l} T_{h,u} \\ \sum\limits_{t=t^s_{e_{u,j}}}^{t^f_{e_{u,j}}} \frac{\beta_{u,j}(t) \cdot \delta_{e_{u,j}}(t)}{b_{h,i}} + d_{h,i} \\ \sum\limits_{t=t^s_{w_j}}^{t^f_{w_j}} \frac{\alpha_j(t) \cdot \delta_{w_j}(t)}{p_i} \end{array} \right) \\ \wedge \mathscr{F}_j = 1 - (1 - \mathscr{F}_u) \cdot \mathscr{P}(l_{h,i}, t) \cdot \mathscr{P}(v_i, t) \leq \mathbb{F}, \quad \text{if } h \neq i \\ \\ \max \left(\begin{array}{l} T_{h,u} \\ \sum\limits_{t=t^s_{w_j}}^{t^f_{w_j}} \frac{\alpha_j(t) \cdot \delta_{w_j}(t)}{p_h} \end{array} \right) \\ \wedge \mathscr{F}_j = 1 - (1 - \mathscr{F}_u) \cdot \mathscr{P}(v_i, t) \leq \mathbb{F}, \quad \text{if } h = i \end{cases}, \tag{5.5.6}$$

where, $\mathscr{F}_j$ and $\mathscr{F}_u$ are the partial OFRs after and before the current module w_j is mapped to the selected node v_i, respectively. We present a distributed heuristic algorithm, *Layer-oriented Dynamic Programming with Fault-tolerance Constraint* (disLDP-$\mathscr{F}$) to achieve MFR under a given OFR bound and further implement a Greedy version, referred to as Greedy disLDP-$\mathscr{F}$. disLDP-$\mathscr{F}$ is similar to disLDP except that we consider the satisfaction of the partial OFR during the mapping node selection.

Note that the value of partial OFRs will be compared at each mapping step by both disRCP-$\mathscr{F}$ and disLDP-$\mathscr{F}$ during the mapping process. The relation between the partial OFRs after and before mapping the current module is defined by Lemma 1, which is used to update the partial OFR at each step.

Lemma 1. *Given module w_j mapped to node v_i, its preceding module $w_u \in pre(w_j)$ mapped to node v_h ($i \neq h$), and the partial OFRs $\mathscr{F}_j$ and $\mathscr{F}_u$ calculated after and before mapping the current module w_j to node v_i, the following equation holds between $\mathscr{F}_j$ and $\mathscr{F}_u$:*

$$\mathscr{F}_j = 1 - (1 - \mathscr{F}_u) \cdot \mathscr{P}(l_{h,i}, t) \cdot \mathscr{P}(v_i, t) \tag{5.5.7}$$

Proof. According to the partially mapped workflow $G_w{}'$ and the definition of OFR in Eq. 5.5.3, we denote the partial OFR before mapping module w_j to node v_i as:

$$\mathscr{F}_u = 1 - \mathscr{R}_u = 1 - \prod_{\text{from } w_0 \text{to } w_u,\ r_i \in R} \mathscr{P}(r_i, t). \tag{5.5.8}$$

After module w_j is mapped to node v_i, the failure probabilities of node v_i and link $l_{h,i}$ need to be taken into consideration, and the current OFR rate is calculated as:

$$\mathscr{F}_j = 1 - \mathscr{R}_j = 1 - \left(\prod_{\text{from } w_0 \text{ to } w_u,\ r_i \in R} \mathscr{P}(r_i, t) \right) \cdot \mathscr{P}(l_{h,i}, t) \cdot \mathscr{P}(v_i, t). \tag{5.5.9}$$

Therefore, we have the relation

$$\mathscr{F}_j = 1 - (1 - \mathscr{F}_u) \cdot \mathscr{P}(l_{h,i}, t) \cdot \mathscr{P}(v_i, t) \tag{5.5.10}$$

by replacing $\prod_{\text{from } w_0 \text{ to } w_u,\ r_i \in R} \mathscr{P}(r_i, t)$ in Eq. 5.5.10 with $1 - \mathscr{F}_u$ based on Eq. 5.5.8. □

In faulty network environments, if a computer node or a network link fails, a centralized mapping algorithm has to recompute and redeploy the entire mapping scheme, which could be prohibitively expensive and result in irresponsiveness of the system. The proposed distributed mapping algorithms under fault-tolerance constraint provide a highly scalable and adaptive solution in response to rapid status changes in large-scale networks because each node only requires its neighbor information during the mapping process and mapping recalculation is confined in a local area where a node or a link fails.

Chapter 6

Simulation of Dynamic Execution of Distributed Systems (SDEDS)

In this chapter, we design and implement a simulation program to visually simulate dynamic workflow execution in distributed heterogeneous environments using a graphical user interface.

6.1 Overview

Due to the enormous scale of applications and structural complexity of workflows as well as the massive distribution, vast diversity, and high dynamics of system resources, implementing and deploying a large-scale distributed collaborative application in a real wide-area network environment is formidably expensive, time-consuming, and labor-intensive. Note that configuring, running, and managing hundreds or even thousands of computing jobs with intricate dependencies on various computer nodes across different network domains would be a tremendously tedious and daunting task if not at all possible. Moreover, since the resources and users in collaborative applications are typically distributed in multiple organizations and regions with their own administrative policies and restrictions, such as grid environments, it is extremely challenging to evaluate the performance of distributed computing tasks and validate the effectiveness and robustness of workflow mapping algorithms

in a repeatable and controllable manner. Performance modeling and simulation-based analysis provide a highly cost-effective way to (i) design, evaluate, and tune workflow systems, and (ii) investigate and refine workflow specifications and module functionalities.

Among the work on the development of automated distributed workflow simulation and management systems, closely related to ours is the GridSim program developed by Buyya *et al.* to model and simulate distributed resource management and scheduling for grid computing, which supports modeling and simulation of heterogeneous grid resources (both time- and space-shared), users, and application models [43]. Hatzis *et al.* proposed the DSS tool for the simulation of distributed systems and protocols through communicating finite state machine, in which the computer system is specified as a process that receives a sequence of input signals in some constrained order and performs a set of actions after receiving the input signals based on both the current and previous inputs [87]. NEST is a graphical environment designed by Dupuy *et al.* for distributed networked system simulation and rapid prototyping [64], which enables users to develop and test distributed systems and protocols (from crude models to actual system code) within simulated network scenarios. NEST allows users to modify and reconfigure the simulation during the execution to study the dynamic response of a distributed system to failures or burst-loads. Bajaj *et al.* proposed a multi-protocol network simulator, VINT, to meet various simulation needs in the network research community by providing a rich environment for experimentation at low cost [32]. Racherla *et al.* developed PARSIT that performs simulation and evaluates the performance of dynamically reconfigurable systems [115]. Combined with their earlier work DYRECT, PARSIT simulates parallel and distributed systems and provides architectural mapping of application programs onto the major categories of architectures. Unger *et al.* presented and analyzed experimental results for the algorithms optimizing service response time in a community using a simulation tool in comparison with optimization algorithms [131], where the point-to-point response time is optimized for each client node in the network. Meftali *et al.* designed a SOAP-based distributed simulation environment

for system-on-chip design [109]. In [77], Gonzalez *et al.* proposed a simulation tool to design and implement distributed control architectures for managing large-scale discrete event systems. Each of the aforementioned simulation systems has its own language, design suite, software structure, target application domain, and the systems vary widely in their execution models and the kinds of components they coordinate.

There exist a number of complex commercial or open-source simulation tools, which are designed mainly for general-purpose network applications, and unfortunately cannot adequately model distributed control architectures and predict the performance constraints that these distributed architectures impose on the system [77]. To overcome these limitations, we propose a lightweight multi-threaded simulation program, *Simulation of Dynamic Execution of Distributed Systems* (SDEDS) [141, 146], based on realistic cost models to simulate the dynamic execution process of distributed systems with data execution on computer nodes and data transfer along network links. The proposed simulation system measures end-to-end latency or data frame rate of a mapped workflow to evaluate the network performance of various workflow mapping schemes in distributed environments through numerical comparison with theoretical calculations and experimental results collected in real network environments. This simulation program can also be used for tuning QoS metrics [112] of a specifically designed workflow management system before its real deployment. We would like to point out that SDEDS is not meant to compete with or replace any existing network simulators, but provide a simulation-based formal method that enables us to quickly and accurately validate the accuracy of cost models and evaluate the performance of workflow scheduling or mapping schemes in distributed heterogeneous network environments.

6.2 System Design

SDEDS takes two graphs and one mapping scheme as input: (i) a DAG that represents a virtual workflow consisting of a set of computing modules with intricate inter-module dependencies, (ii) an arbitrary directed weighted graph that represents an overlay network consisting of a set of computer nodes interconnected with network links, and (iii) a workflow mapping scheme that assigns each computing module to a computer node. Based on the mapping scheme, SDEDS overlays the virtual computing workflow on the computer network in a Graphical User Interface (GUI), and visually illustrates the dynamic execution process: module execution is visualized by filling an object and data transfer is visualized by moving the object along a communication link in the network. Each computing module in the workflow is designed as an autonomous agent performing three activities: receiving data from its preceding module(s), performing a predefined computing routine, and sending results to its succeeding module(s), which are simulated and coordinated by multiple threads. Furthermore, SDEDS is able to simulate workflow execution in different types of network environments with and without background traffic and workload.

6.2.1 SDEDS Framework

The framework of SDEDS is shown in Fig. 6.1, where we define one class for each of these four components: computing module, dependency edge, computer node, and network link. The design details of these structures are described as follows:

- **Computing Module Structure:** Each module has a static attribute table that contains its attribute data including module ID, computational complexity (CC), a list of input/output edges (IE/OE), aggregate input data size, total number of CPU cycles, and ID of the node to which the current module is assigned. The dynamic information including the number of CPU cycles left to execute, arrival time of input data, and departure time of output data, is also maintained during simulation.

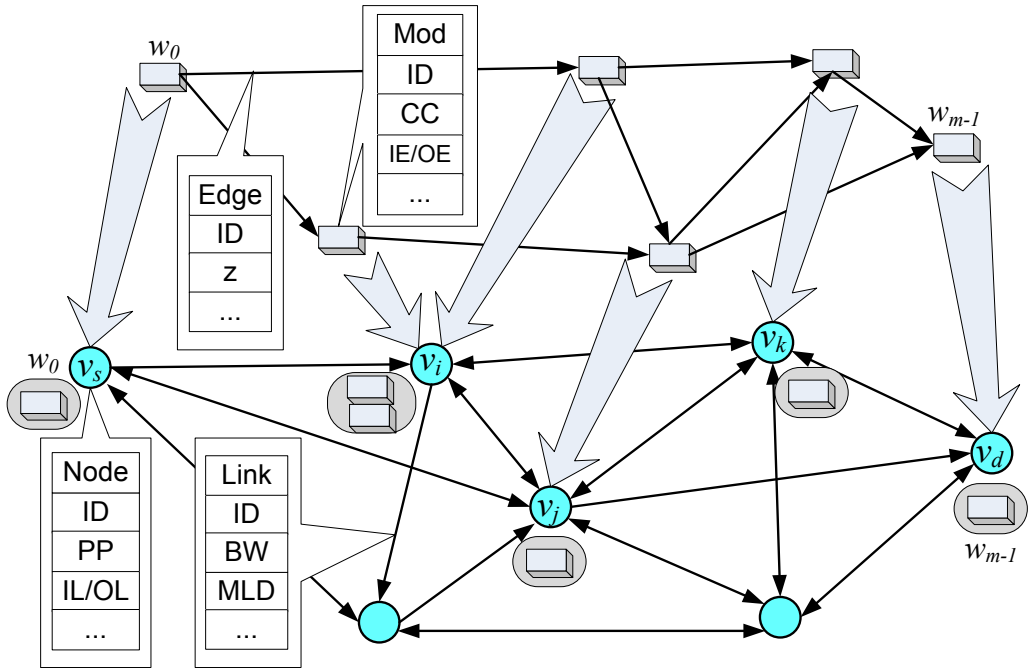


Figure 6.1: The framework of SDEDS.

- **Dependency Edge Structure:** Similarly, each edge has a static attribute table that contains its attribute data including edge ID, transferred datasets, preceding module(s), succeeding module(s), and ID of the network link to which the current edge is assigned. The dynamic information including the number of bytes left to transfer for the current dataset and availability of new data from its preceding module is maintained during simulation.

- **Computer Node Structure:** Each node has a static attribute table that contains its attribute data including node ID, processing power (PP), a list of input/output network links (IL/OL), and a list of modules that are assigned to the current node. The node also keeps track of the set of modules that are concurrently running and allocates an equal share of computing resources (processing power) to them during simulation.

- **Network Link Structure:** Similarly, each link has a static attribute table that contains its attribute data including link ID, link bandwidth, MLD, preceding node, succeeding node, and a list of dependency edges that are assigned to the current link. The link also keeps track of the set of edges that are concurrently transferring and allocates an equal share of network resources (i.e., bandwidth) to them during simulation.

The mapping scheme overlays the virtual workflow on the underlying computer network with the objective to optimize the end-to-end workflow performance in terms of MED for unitary processing applications or MFR for streaming applications with time-series input datasets.

6.2.2 Dynamics Control

Each module in a workflow is designed as an autonomous agent, which performs three activities: receiving data from its preceding module(s), processing data, and sending results to its succeeding module(s), which are simulated using three threads, *RecvThread*, *ProcThread*, and *SendThread*, as shown in Fig. 6.2.

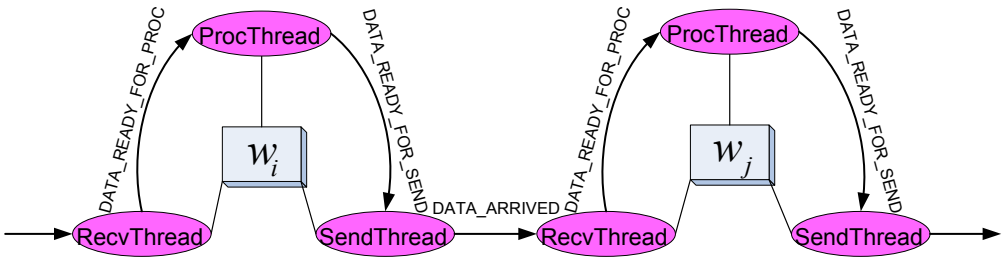


Figure 6.2: Multithreading design for computing modules.

RecvThread checks for arrival of incoming data on the incident edges from its preceding modules. If all input data arrive, it sends a "DATA_READY_FOR_PROC" signal to *ProcThread*. If *ProcThread* is currently idle, it first calculates the needed processing

time based on the data size, computational complexity, number of concurrent modules and available computing resource of the assigned node at the present time, and then simulates the data processing by calling a sleep function. Meanwhile, it registers itself on the underlying node for the use of computing resources. It wakes up at a certain rate to check the node sharing status and updates the amount of left execution time accordingly. Upon successful completion of data processing, *ProcThread* sends a "DATA_READY_FOR_SEND" signal to *SendThread* and removes itself from the list of concurrently running modules on the node. Once receiving a signal from *ProcThread*, *SendThread* starts simulating the data transfer process along each outgoing edge based on the output data size, number of concurrent data transfers, and available link communication resource at the present time. Similarly, the transfer time is also dynamically updated by checking the number of concurrent datasets transferred over the same link during simulation. Upon completion of data transfer, *SendThread* sends a "DATA_ARRIVED" signal to *RecvThread* of the dependency edge leading to the succeeding module. These three threads can run concurrently when multiple instances of input datasets are continuously fed into the system.

Note that multiple independent modules assigned to the same node may or may not run concurrently to share the node's CPU cycles, depending on their different start times when they receive all the input datasets. Similarly, multiple edges assigned to the same link may or may not transfer data concurrently to share the link bandwidth, depending on start times of different data transfers. In SDEDS, the node and link sharing is constantly monitored to provide appropriate computing and communication resources to concurrent module executions and data transfers, respectively.

6.3 Graphical User Interface Implementation

The Graphical User Interface (GUI) of the SDEDS simulator is implemented in Visual C++, which uses .NET components for rich graphical display and enhanced event control.

The overlaid workflow on the network is drawn in a .NET PictureBox, and the entire GUI is designed and built on the core simulation functions presented in Chapter 6.2.

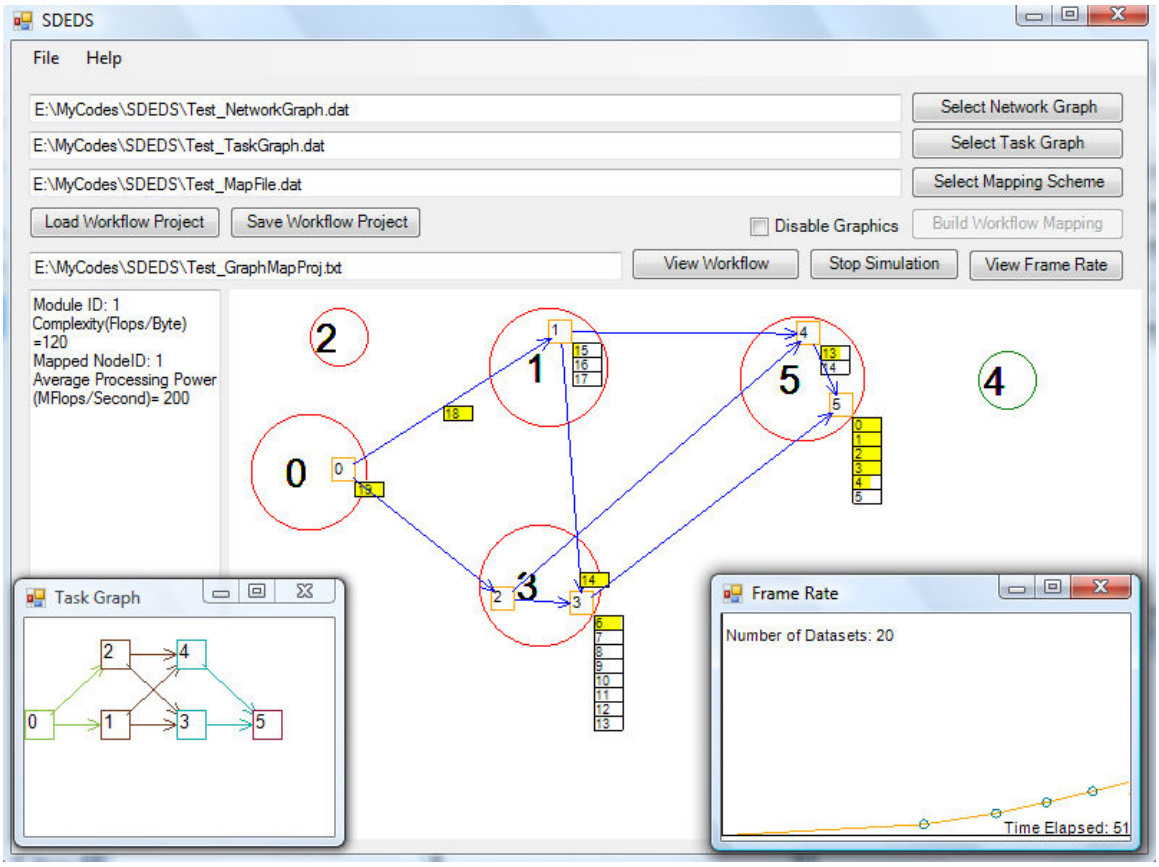


Figure 6.3: A runtime simulation example of a small-scale workflow application.

In the GUI, the user can select a computing workflow, a computer network, and a mapping scheme, and draw computer nodes and overlaid modules on the display as shown in Fig. 6.3. The workflow mapping algorithms are implemented in C/C++ and incorporated into SDEDS for computing mapping schemes. When a simulation starts, the GUI retrieves status information from the workflow periodically to update and repaint the graph. Both the retrieval function during the runtime of a simulation and the repaint function are called by a .NET Timer object. The user is also provided the functionality to select and edit the display properties of nodes and modules. Each dataset processed by a module is represented by a small rectangle, and the module execution is denoted by filling out the rectangle and the

data transfer is illustrated by moving the rectangle along a network link. However, these animations can be disabled to reduce the visualization overhead for a higher simulation accuracy. We design two main window views: one is the display of the topologically sorted workflow, and the other is a two-dimensional plot of MFR during the course of workflow execution, where x axis represents the time step and y axis represents the accumulated number of completed datasets.

A small-scale running example is provided in Fig. 6.3 to illustrate the system execution dynamics. In the main window, each circle represents a computer node in the network and each square within a circle represents a computing module in the workflow mapped to this node: modules 0 and 1 are mapped to nodes 0 and 1, respectively, modules 2 and 3 are mapped to node 3, and modules 4 and 5 are mapped to node 5. Network links are purposefully not shown in the figure for a clear view of workflow mapping.

Chapter 7

Scientific Workflow Automation and Management Platform (SWAMP)

In this chapter, we develop and deploy a workflow automation and management system for end users to conveniently construct, execute, display and monitor workflow applications, where they can focus more on their domain reseach and leave the execution details to the system.

7.1 Overview

A plethora of frameworks and tools have been developed for generating, refining, and executing scientific workflows. Such efforts include P-GRADE [93], Pegasus [57], Kepler [103], Condor/DAGMan [12], Gridbus [13], SimGrid [45], Taverna project [89], GridSim [43], Triana [51], Java CoG Kit [101], Sedna [134], ASKALON [135] and other grid systems such as Open Science Grid (OSG) [8] as well. Each workflow or grid system has its own language, design suite, and software components, and the systems vary widely in their execution models and the kinds of components they coordinate [56]. Some systems attempt to provide general-purpose workflow functionalities while others are more geared toward specific applications and are optimized to support specific component libraries. These existing workflow or grid systems have a primary design goal to provide

services or infrastructures for coordinated application interoperation, distributed job submission, or large data transfer, but generally there lacks a comprehensive workflow solution that integrates workflow mapping schemes for end-to-end performance optimization. Furthermore, most systems using a batch scheduler are not inherently capable of supporting streaming applications that require computational steering, which is a critical activity in explorative sciences where the parameter values of an online simulation or a computing module must be changed and determined on a realistic time scale.

We design and develop a generic *Scientific Workflow Automation and Management Platform* (SWAMP) [145, 150], which contains a set of easy-to-use computing and networking toolkits for application scientists to conveniently assemble, execute, monitor, and control complex computing workflows in distributed network environments. SWAMP is a Condor/DAGMan-based workflow system that automates and manages scientific workflows in network environments through a web-based user interface. SWAMP features a flexible workflow mapper where a third-party workflow mapping algorithm could be plugged in. In the current version, we integrate the proposed mapping algorithms for MED and MFR into the workflow mapper in place of Condor's default mapping scheme to achieve the better end-to-end performance. We conduct two real case study of the workflows for Spallation Neutron Source [2] datasets and Climate Modeling [5] datasets to show the efficacy of the proposed platform.

This is a DOE-funded three-year team-based project, where I am a key participate. My main responsibility in this project is to develop a workflow mapper for SWAMP to optimize workflow performance by incorporating my own workflow mapping schemes based on rigorous algorithm design and performance analysis. This mapper unit significantly improves the overall network performance of workflow applications compared to existing mapping methods.

7.2 System Design

The current SWAMP framework shown in Fig. 7.1 consists of the following functional components: (a) Kepler and Kepler Manager, (b) Web Server, Web Server CGI, and Web Server Manager, (c) DAGMan and DAGMan Manager, (d) Workflow Repository, (e) Network and System Information Management, (f) Workflow Mapper, and (g) Workflow Execution. These components interact with each other to accomplish the tasks of workflow generation, mapping, execution, monitoring, visualization, and steering. Within SWAMP, a user can use either a graphical web interface or the GUI of Kepler to compose abstract workflows. The Kepler Manager converts the abstract workflow in XML format to DAG format, and sends these workflow description files including a meta-workflow and a list of component workflows to the Web Server Manager. The Web Server provides a visual management interface for workflow selection, dispatch, monitoring, and steering. A meta-workflow file may contain one component workflow with a single-input dataset or more with multiple (e.g., time-series) input datasets. A user can select those component workflows of interest, configure their parameter settings, and dispatch them for execution through a web browser.

Upon the receival of an abstract workflow from the Web Server, the DAGMan Manager invokes the Workflow Mapper to map the abstract workflow to the real network based on the availability and capability of computer nodes collected by the Network and System Information Management component. The mapped abstract workflow is then submitted to Condor or Condor-G, which extracts executable modules and source datasets from the workflow repository and composes them into an executable workflow. The mapped executable workflow is dispatched to the real network for execution. The workflow execution status information as well as the final results are collected and sent on the fly to the Web Server Manager for display on the web page. Based on the displayed results, the user may reset the value of a command argument for those steerable computing modules and re-dispatch them. The details of the design and the implementation are provided as below.

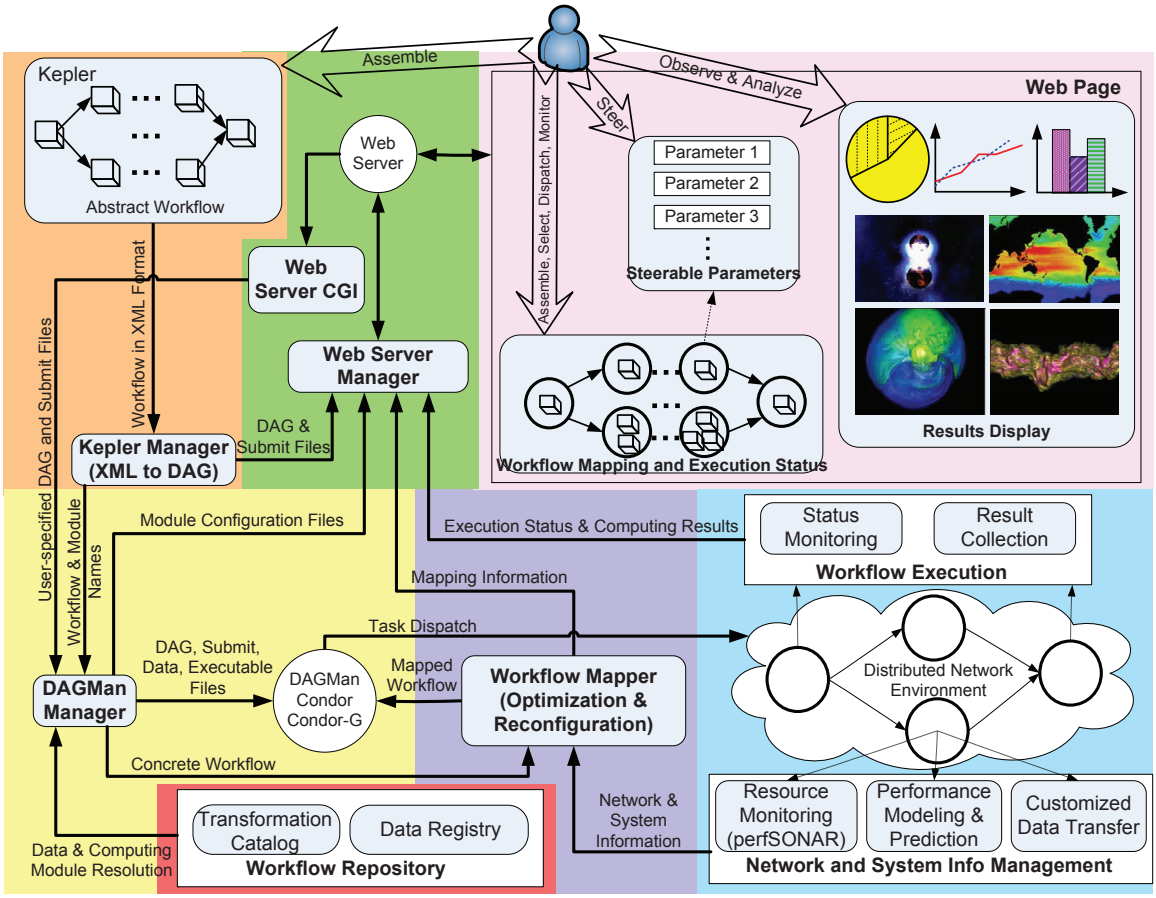


Figure 7.1: SWAMP framework: functional components, control and data flow.

7.2.1 Workflow Generation

In addition to the GUI of Kepler, SWAMP provides a graphical toolkit for visual construction of abstract workflows on a unified web interface, which generates DAG and submit files required by Condor DAGMan. As shown in Fig. 7.2, the web interface for workflow composition consists of a menu bar, workplace, and help window. Users can drag and drop modules in the workplace and edit their parameters required for execution. This web interface also supports the creation of a meta-workflow in XML for easy human interpretation and machine parsing. A meta-workflow describes a list of component workflows that can be submitted as a full set, subset, or individually depending on the needs of the user.

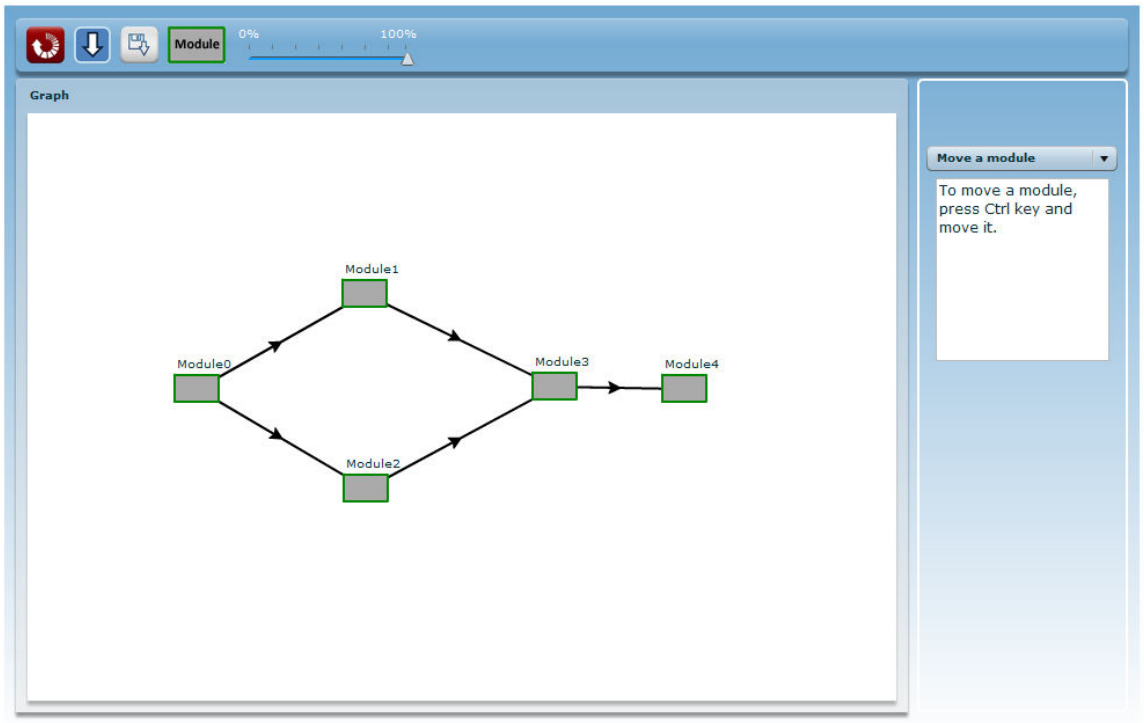


Figure 7.2: The web interface for visual abstract workflow composition.

7.2.2 Workflow Selection, Dispatch and Result Display

We use AJAX web technology to display all workflow layouts that belong to a particular meta-workflow and the user may select one or more component workflows for immediate dispatch in a batch mode. The web component acts as a client and transmits user-selected workflows in succession to the DAGMan Manager, which then proceeds to process and dispatch the workflows to the Condor environment. The interactions between the web site and the DAGMan Manager are accomplished through a socket-based protocol model.

Each component workflow eventually generates an output. The ability to view and manage the output as soon as it is produced is valuable to users for result interpretation. The web interface allows immediate visualization of results as they are transferred directly from the condor environment to the Web Server along the course of workflow execution.

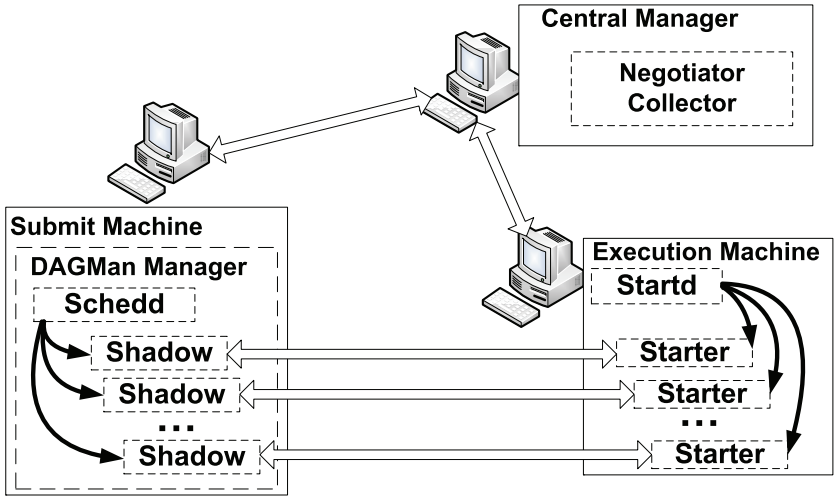


Figure 7.3: Condor pool architecture.

7.2.3 DAGMan Manager

The DAGMan Manager essentially serves as a portal to the Condor batch system. The main goal of the Condor Portal is to simplify the process of workflow submission and execution. The DAGMan Manager listens on a predefined port and establishes socket connections with the Web Server Manager and Kepler Manager. The DAGMan Manager sends the corresponding module configuration files to the Web Server Manager after receiving the workflow information from the Kepler Manager.

The SWAMP system allows users to submit jobs by using a specified repository, which stores all the input data as well as the existing output data. The SWAMP system also allows users to submit jobs by using a shared file system to directly access input and output files. In the latter case, the job or module must be able to access the data files from any machine on which it is deployed through either NFS or AFS. The SWAMP system provides access to the Condor's own native mechanisms for grid computing as well as other grid systems.

Fig. 7.3 shows a local Condor pool architecture. Each execution node in the SWAMP condor pool accepts a new job only if the load of the system is not too high and there is enough memory available for efficient execution. The jobs are submitted to the "schedd"

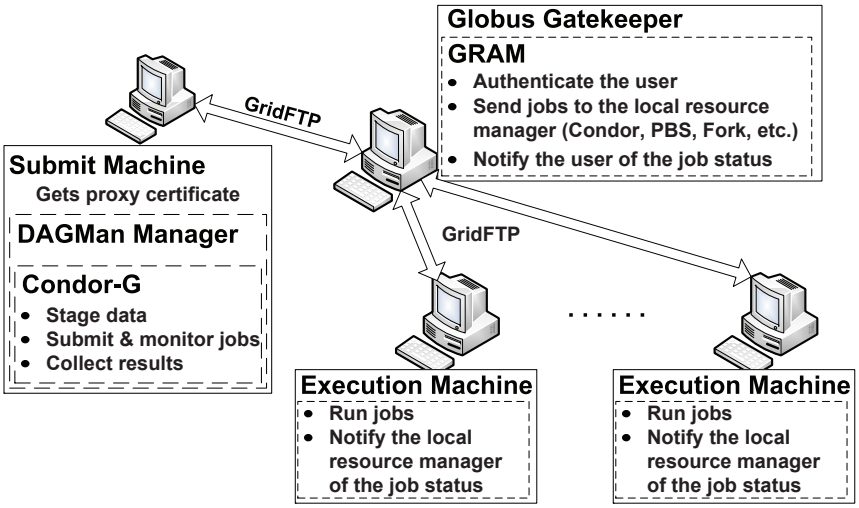


Figure 7.4: The execution procedure of SWAMP in grid environments.

process, which stores them in a permanent storage and advertises their needs. The "startd" process on an execution node advertises its resources to the "collector" process. The "negotiator" process regularly fetches these advertisements (ClassAds) from the "collector" and "schedd", and assigns jobs to execution nodes. For every such association, a shadow and a starter process are created and all further communications take place between these two entities.

Fig. 7.4 shows the execution procedure of SWAMP in grid environments. In order to submit jobs to grid resources, the submit machine needs to apply for a certificate from a trusted Certificate Authority (CA) for authentication. The submit machine with all grid software installed and configured manages the jobs running on the grid. Before running jobs on the grid, we need to ensure that the submit machine is able to obtain the correct user proxy, which is presented to remote grid sites during authentication. Condor-G stages data and submits jobs to the Gatekeeper at a remote grid site. Globus GRAM on the Gatekeeper authenticates users, sends jobs to the local resource manager such as Condor, PBS, and Fork, and notifies users of job statuses. The execution machines run the jobs and notify the local resource managers of job statuses.

Each workflow supported by SWAMP has an output module that runs after all other user jobs are completed. The main task of this output module is to send the final results to the Web Server. Instead of creating a separate daemon, we add this extra module as part of the DAG workflow to avoid checking the Condor log files or invoking Condor's management command in the polling mode.

7.2.4 Network and System Information Management

SWAMP discovers the information of networking and computing resources by querying the Network and System Information Management component. The Resource Monitoring subcomponent interacts with network performance modeling tools such as the One-Way Active Measurement Protocol (OWAMP) [14] and Bandwidth Control (BWCTL) [15] provided by Internet2 to monitor the statuses of networking resources such as link delay and available link bandwidth. It can also interact with information services such as the Globus Monitoring and Discovery Service (MDS) [16], OSG Resource and Site Validation (RSV) system [17], and any other information services to discover available computing resources and their characteristics such as the number of CPUs, CPU frequency, queueing length, available disk space, and so on. In order to improve the performance of large data transfer in wide-area networks, the Resource Monitoring subcomponent queries the aforementioned information services to locate and employ available data movement services such as GridFTP [18], Reliable File Transfer (RFT) [19], Storage Resource Broker (SRB) [20], Storage Resource Management (SRM) [21], and OSCARS [22].

7.2.5 Workflow Mapper

The efficiency of the SWAMP system largely depends on the performance of mapping schemes that map computing workflows to network nodes in Condor, which is determined

by the Condor job dispatch scheme. The mapping scheme currently employed by the Condor scheduler works as a matchmaker of ClassAds. Condor schedules job dispatch by matching the requests for both machine ClassAds and job ClassAds.

SWAMP provides a better mapping performance by incorporating new mapping schemes into the Condor negotiator daemon. One unique feature of SWAMP is the incorporation of a specially-designed mapping engine that automatically maps abstract workflows to underlying networks to achieve better end-to-end performance based on the above real-time network and system status measurements. This mapping engine is built upon a set of workflow mapping methods developed through mathematical performance modeling and rigorous algorithm design. Here, we incorporate the workflow mapping solutions proposed in Chapter 5 to achieve the better performance during execution.

Note that Condor/DAGMan has a centralized execution model where the output of each module is sent back to the submitter for forwarding to its succeeding modules. This centralized model may introduce prohibitively large data traffics in the network, especially for data-intensive workflows with many computing modules. We realize a completely distributed execution model by adopting Stork in SWAMP for direct inter-module data transfer using GridFTP. The workflow execution status information as well as the final results are collected and sent on the fly to the Web Server Manager for display on the web page. Based on the displayed results, the user may reset the value of a command argument in those steerable computing modules and re-dispatch them into the network. SWAMP can be also deployed in the grid environment to provide optimal performance in high-bandwidth high-latency networks.

7.2.6 Data Provenance Tracking

Data management in heterogeneous networks such as OSG [8] is a key part of the architecture and service of the common infrastructure. Large-scale scientific applications have led to unprecedented levels of data collection, which make it very difficult to keep track of the

data generation history. Data provenance provides information about the derivation history of data starting from its original sources. It enables scientists to verify the correctness of their simulations and reproduce them if necessary. SWAMP provides provenance information related to the execution of workflow modules by analyzing the log files of the Condor system. This information contains the execution environment of each module such as the execution location, execution time, and execution output.

7.3 Two Real Use Cases

In order to conduct experiments on SWAMP using real-life workflows and investigate the superior performance of the proposed workflow mapping solutions, we collaborate with the scientists at the national laboratories and execute the real-life workflow applications in SWAMP system for their research.

7.3.1 Spallation Neutron Source (SNS) Workflows

A Brief Introduction to SNS Workflow

The U.S. Department of Energy's Spallation Neutron Source (SNS) is the world's most powerful facility for pulsed neutron scattering science. This one-of-a-kind facility provides the most intense pulsed neutron beams in the world for scientific research and industrial development. SNS was built by a partnership of six U.S Department of Energy laboratories.

One important computing task of SNS is to analyze data from the inelastic scattering instruments deployed at the SNS experimental facility located at Oak Ridge National Laboratory (ORNL). As shown in Fig. 7.5, this data analysis process includes two main computing modules, i.e., data rotation and data rebinning. The process focuses on multiple energy transfer slices from reduced experimental datasets that contain multiple runs, each corresponding to a certain rotation angle at the time of acquisition. The datasets must be rotated according to the corresponding goiniometer settings and converted to reciprocal

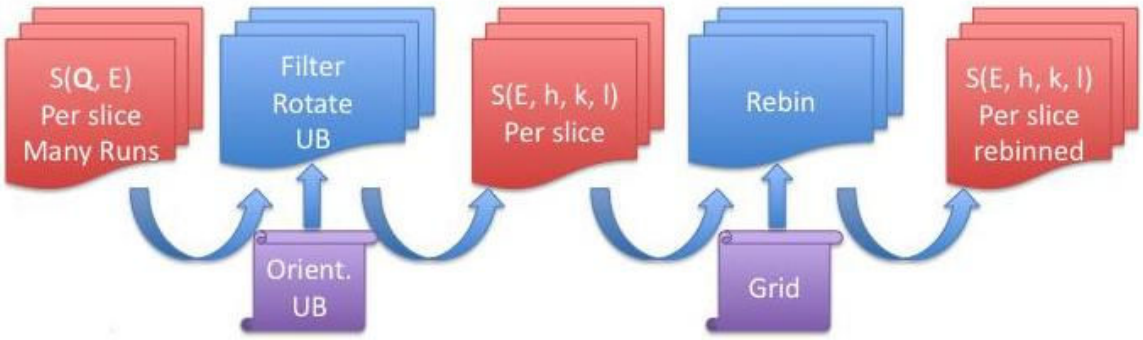


Figure 7.5: SNS data analysis process.

lattice coordinates via the orientation (UB) matrix application. The output is rebinned to place all data on the same final grid with the result of the data analysis process on one energy slice being a file containing 3D histogram information. This file can be used for further data analysis or rendered to produce frames for visual inspection. The process needs to be repeated for each energy slice generated during the preceding data reduction stage.

SNS Workflow Structure

The SNS experimental dataset used in this study contains 160 energy slices, each of which contains 61 runs of reduced data. For each energy slice, we create a separate component workflow that rotates the data of 61 runs in parallel and then performs a concatenation operation on the rotated data. The rebinning result of the concatenated data is eventually converted to an image file, which is sent to the Web Server Manager for display on the web user interface.

There are 160 component workflows in total for the SNS experimental dataset. To manage these component workflows, we create a workflow meta file in XML format that contains a list of component workflow names and other related information. As shown in

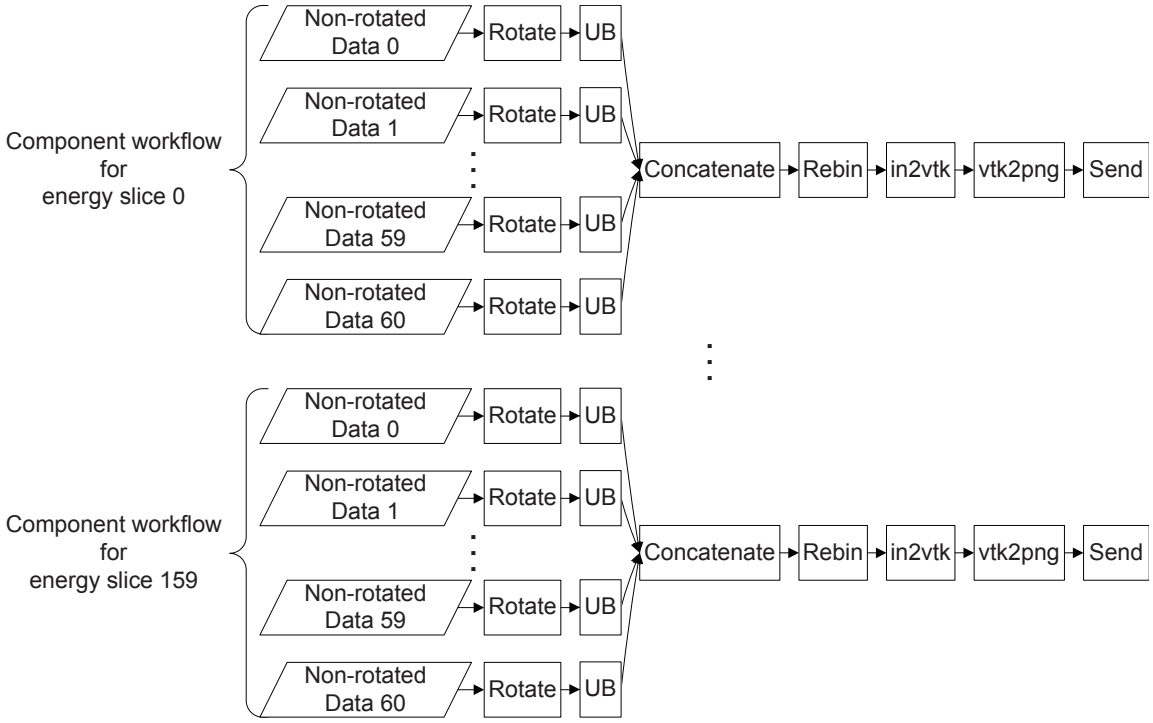


Figure 7.6: SNS workflow structure.

Fig.7.6, each component workflow consists of five modules, namely “rotate”, “concatenate”, “rebin”, “convert”, and “send”, which read and process 61 source data files for a specific energy slice with a different rotation angle.

A source data file contains a list of voxels, one for each detector pixel that has non-zero content, which must be rotated first according to the goiniometer settings. This is achieved by multiplying each voxel by a rotation matrix. Once all the data files have been rotated, they are concatenated into one single file for rebinning, which requires the minimum and maximum extents of three axes and the size (width) of the bins along each axis that are desired. The output of the current rebinning procedure is an ASCII text file, which is first converted to a VTK file [23] and then to the final image file using the VisIt command [24] executed by a Python script. Once all image files are generated and received by the Web Server Manager, the user is provided with a function to produce a movie out of these images at the convenience of one mouse click on the web interface.

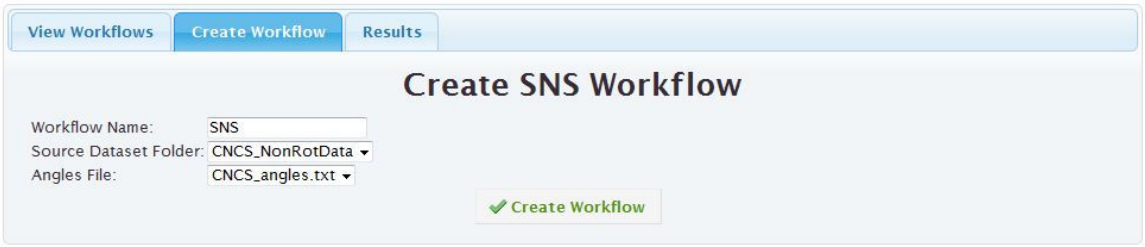


Figure 7.7: SNS workflow generation.

SNS Workflow Generation

The SNS dataset requires generating 160 workflows to process 160 energy slices, each of which contains 61 runs. Since it is very inconvenient for users to manually create such a large number of workflows using Kepler's GUI, we develop a special program, named SNSWorkflow, to automatically generate the DAG and submit files for 160 component workflows according to the source datasets. Each component workflow consists of 5 modules: rotation, concatenation, rebinning, image conversion, and an additional output module that sends the final output .png image file to the Web Server Manager.

The SNS source datasets are organized in the following way: under the dataset directory, there are 61 sub-directories corresponding to different rotation angles, each of which holds 160 energy slice files for a specific rotation angle. SNSWorkflow iteratively reads all the directories, fetches the file names of all the source data, and generates the workflow based on the structure of the given source dataset. A rotation module takes 61 source data file names as input arguments. Each of the input data comes from one of the 61 sub-directories. These arguments are placed in the submit file of the "rotate" module. SNSWorkflow also generates the submit files for other modules. Besides the workflow DAG and submit files, a workflow meta file in XML format is also created to describe the entire set of component workflows. As shown in Fig. 7.7, the user can use the web-based GUI to specify the data location and the rotation angle file to generate the required SNS workflow files. This capability of workflow generation based on a unified web interface is

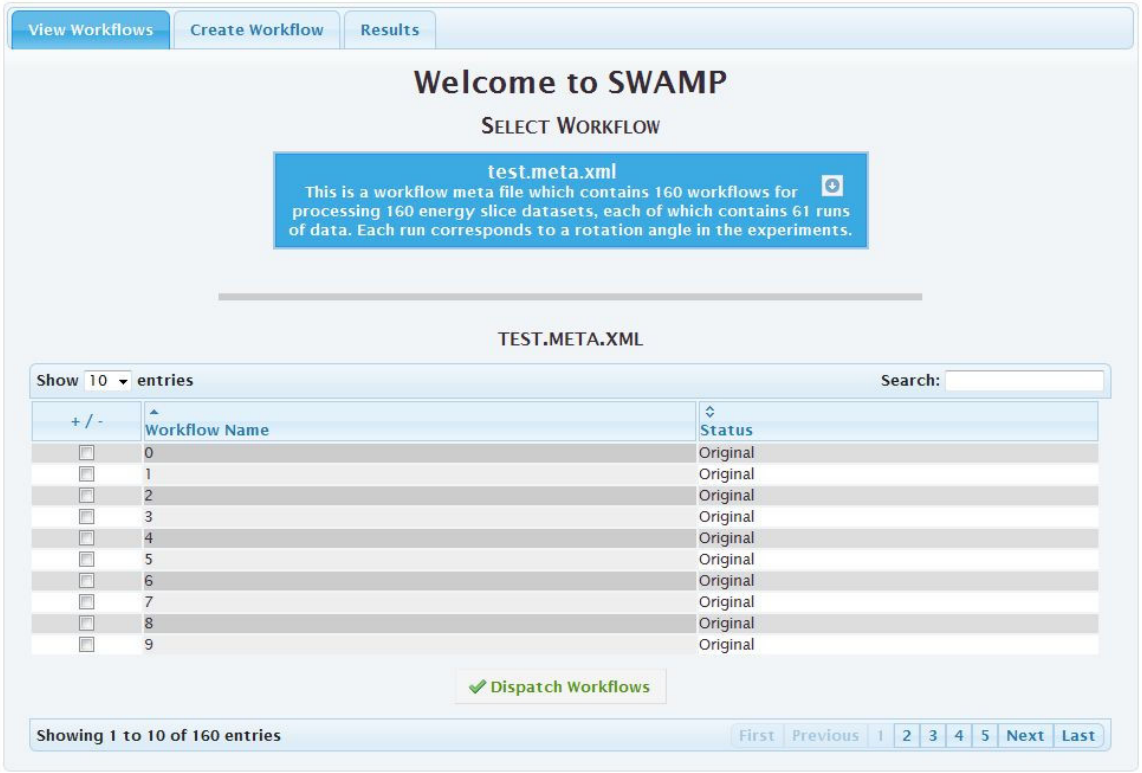


Figure 7.8: SNS workflow selection and dispatch.

currently under development and will be made generic for different applications.

Once the workflow meta file is generated, the user can select one or more SNS component workflows to be dispatched immediately in a batch mode, as shown in Fig. 7.8. The test network environment contains about 10 regularly configured Linux boxes distributed at University of Memphis.

SNS Workflow Result Display

Each SNS workflow eventually generates one image. The web interface has been designed to allow immediate visualization of images as they are transferred from the Condor environment to the Web Server Manager along the course of workflow execution. Our web component can work with many different data formats in concert to create a more meaningful visual output. For illustration purposes, a list of SNS image files are displayed in

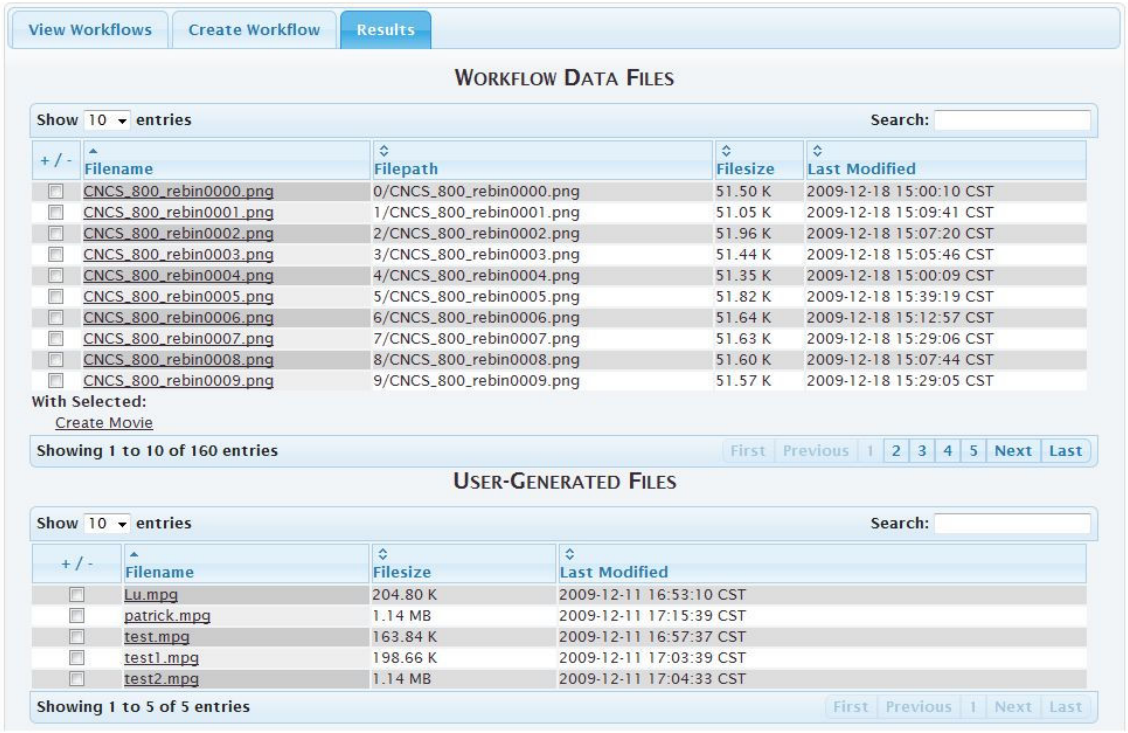


Figure 7.9: SNS workflow image files.

Fig. 7.9 and the final image for a rebinned energy slice is shown in Fig. 7.10.

We also provide the user the ability to create videos from a sequence of image files in an intuitive interface that allows for selection and arrangement of image files into frames of a video, as shown in Fig. 7.11. The user can select a group of image files and create a video using the selected files as individual frames in the video. Videos can be created and displayed on-screen without leaving the website or refreshing the page. All user-generated data such as videos or graphs are stored for later access/playback through a sortable table.

7.3.2 Climate Modeling (CM) Workflows

A Brief Introduction to CM Workflows

To cope with the high computational expense associated with evaluating new parameterizations using a complete atmospheric general circulation model, climate researchers have

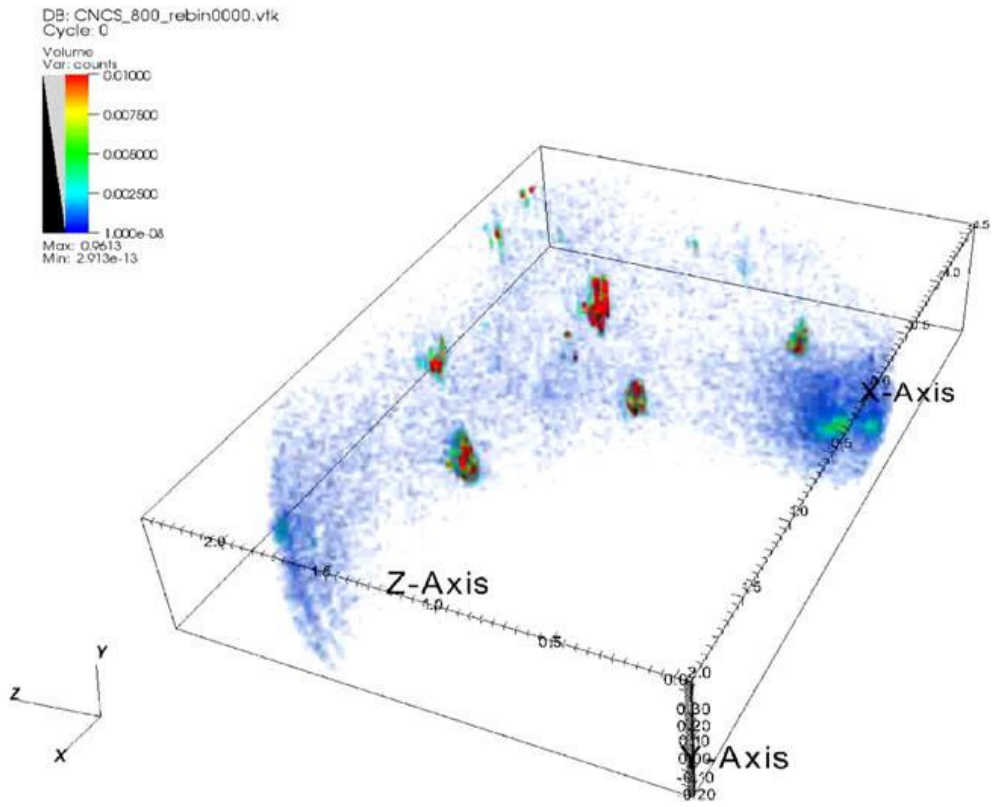


Figure 7.10: The final image of a rebinned slice.

developed a highly flexible and computationally inexpensive single column modeling environment for the investigation of parameterized physics targeted for global climate models. In particular, this framework is designed to facilitate the development and evaluation of physical parameterizations for the NCAR Single column Community Atmosphere Model (SCAM).

We represent SCAM as a workflow with time-series input data that can be executed using SWAMP in grid environments with GridFTP-enabled inter-module data transfer. For each combination of available physics packages, data to be used, and experiment controls, we treat it as a dispatch of one SCAM workflow. There are a varying number of workflow dispatches for the SCAM experiment depending on different combinations of parameter selections. As shown in Fig. 7.12, each SCAM workflow consists of fourteen modules. The

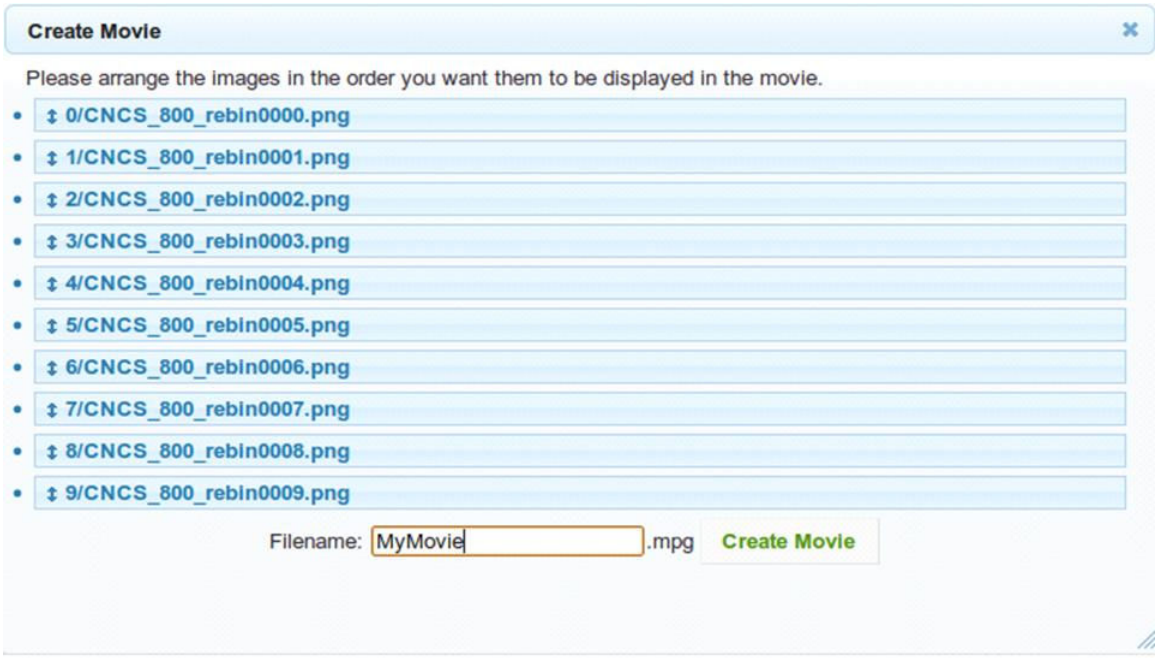


Figure 7.11: Create a movie out of a sequence of images generated by component workflows.

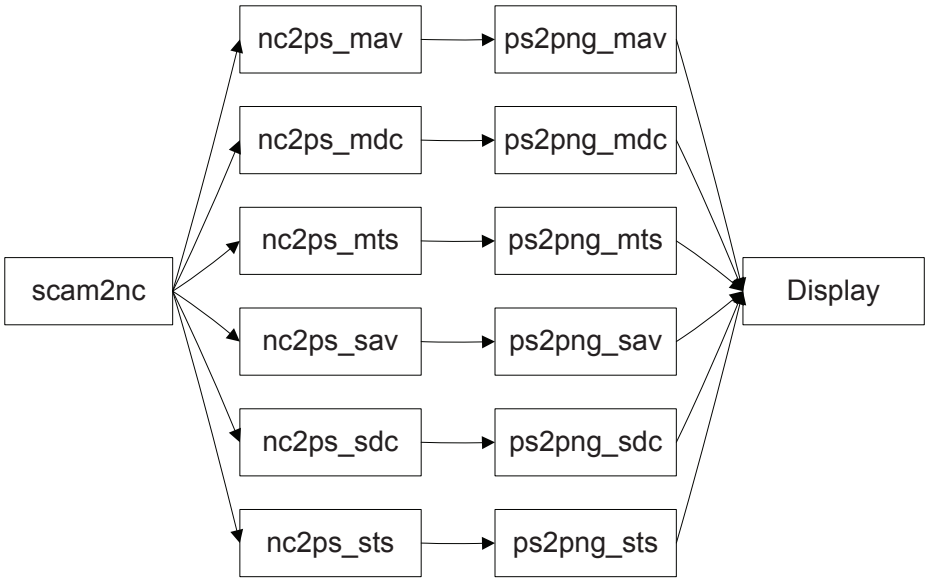


Figure 7.12: SCAM workflow structure.

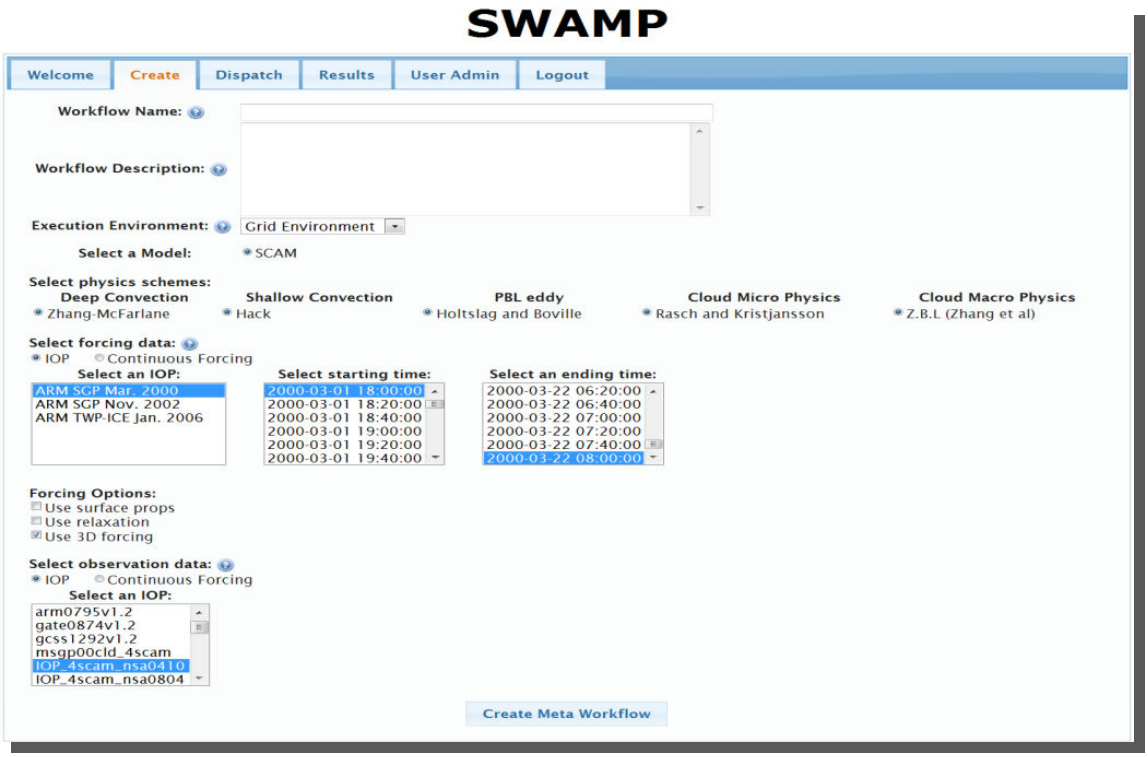


Figure 7.13: SCAM workflow generation.

SCAM workflow uses csh script as a master control, NCAR Command Language (NCL) script to process data and produce plots, and ghostscript to convert the postscript image format to web-ready format (e.g., jpeg, png). We developed a program to automatically generate DAG and submit files for the climate modeling workflow. As shown in Fig. 7.13, all the combinations of parameters are gathered from the web input and passed into the SCAM model through a startup file.

SWAMP provides a web-based interface to automate and manage the SCAM workflow execution and uses a special site-level workflow mapper to optimize its end-to-end performance on OSG. Fig. 7.14 shows the executable SCAM workflow structure where a solid line represents a data flow and a dotted line represents a control flow. Several auxiliary modules are added to the SCAM workflow to stage in data, executable, and libraries required by the modules and remove the data used/produced by modules at the remote sites.

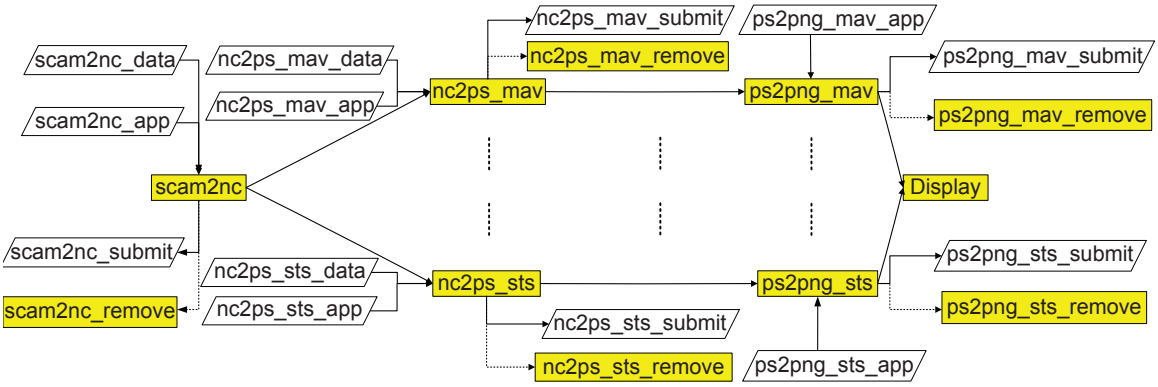


Figure 7.14: SCAM executable workflow structure.

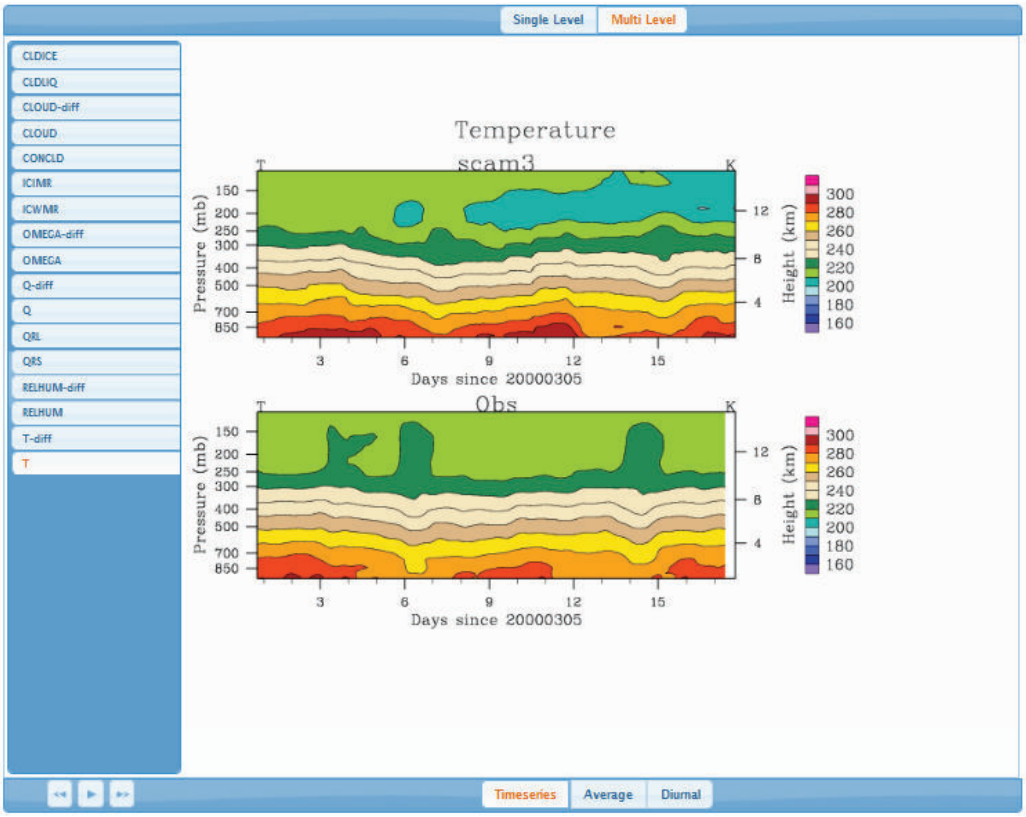


Figure 7.15: A gallery of final images generated by the SCAM workflow.

Each SCAM workflow eventually generates a number of images. A gallery of final images for one SCAM workflow are provided on the web interface for a visual examination, as shown in Fig. 7.15.

Chapter 8

Performance Evaluation and Comparison

In this chapter, we conduct an extensive set of performance evaluation and comparison of the proposed mapping algorithms for both pipeline- and DAG-structured workflows using theoretical calculation, simulation, and real-life workflow applications. For performance comparison purposes, we also adapt and implement several existing workflow mapping algorithms on the same platform. The experimental results illustrate the performance superiority of our mapping solutions over others in terms of MED, MFR, and OFR.

8.1 Implementation Details and Experimental Settings

We first describe the implementation details for both simulations and experiments, including theoretical parameters and physical environments.

The computational complexity of a module in the workflow is an abstract quantity that does not only depend on the computational complexity of the algorithm in the module but also the implementation details such as the specific data structures used in the program. We assume that the first module in the workflow performs nothing but transfer raw data on the designated source node to its succeeding neighbor node(s). The partial result produced by an intermediate module serves as input data to its succeeding modules in the workflow. The node processing power is another abstract quantity that characterizes the general computing

capability of a computer node, which is primarily determined by the processor frequency, memory size, and bus speed.

For each mapping problem, we designate a source node and a destination node to run the first module and the last module of the workflow. This is based on the consideration that the system knows where the raw data are stored and where an end user is located before performing workflow mapping in an existing network.

8.1.1 Simulations

All proposed workflow mapping algorithms are implemented in C/C++. For performance comparison in pipeline mapping, the simulations are conducted on a Windows XP PC equipped with a 3.0 GHz CPU and 3.0 Gbytes memory; while for DAG-structured workflow mapping, the simulations are conducted on a Windows 7 desktop PC equipped with Intel Core 2 Duo CPU E7500 of 2.92 GHz and 3.0 GB memory.

We develop a separate program to generate simulation datasets by randomly varying the parameters of workflows and networks within a suitably selected range of values, represented by a four-tuple (m, $|E_w|$, n, $|E_c|$): m modules and $|E_w|$ edges in the workflow, and n nodes and $|E_c|$ links in the network. For a given problem size, we randomly vary the module complexity and data size within a suitably selected range and create the workflow topology using the following steps:

1. Lay out all modules sequentially along a pipeline;

2. For each module, add an input edge from a randomly selected preceding module and add an output edge to a randomly selected succeeding module (note that the first module only needs output and the last module only needs input);

3. Randomly pick up two modules from the pipeline and add a directed edge between them (from left to right) until we reach the given number of edges.

Here, the workflow topology could be either a pipeline and a DAG, depending on m and $|E_w|$:

- If $m = |E_w| + 1$, only execute Step 1;
- If $m < |E_w| + 1$, execute all three steps;
- Otherwise, return an error message.

To create a network with arbitrary topology, we first start with a complete network and then randomly pick up a link for removal until we reach the given number of links. If the removal of a link results in a disconnected network, we will keep this link and repeat the random link selection until we find a valid removal operation. Similarly, we randomly vary the BW and MLD within a suitably selected range.

8.1.2 Experiments

In the experiments, we set up a wide-area network testbed consisting of 12 PC workstations with different hardware configurations in terms of CPU frequency, memory size, and disk space. The CPU frequency of these computers, which is the most important hardware parameter, falls in a range between 1.2 GHz and 3.4 GHz.

We employ a linear regression method to estimate the effective bandwidth and MLD of a network link [148], and also deploy some daemon nodes in the network to monitor, estimate and measure system resources in real time. If a significant change is detected, those daemons will report the change and trigger a rerunning of the mapping algorithms to find a new mapping scheme.

In the network testbed, we create an arbitrary network topology by configuring a different firewall setting on each computer and using the Linux traffic control command "tc" to allocate a different bandwidth along each overlay link. We deploy a Condor/DAGMan-based workflow management system, SWAMP, in the network testbed to evaluate the performance of different mapping algorithms.

8.2 Algorithms for Comparison

For comparison purposes, we adapt, modify and implement four existing workflow mapping heuristics to both MED and MFR problems, namely *Greedy* A^* [121], *Streamline* [26], *Dynamic Level Scheduling* [125] and *Greedy*. Note that these algorithms might fail to find a feasible mapping solution due to the dependency in the workflows and topology restrictions in the networks.

8.2.1 Greedy A^* Algorithm

A static allocation scheme based on A^* algorithm was proposed by Sekhar *et al.* [121], which maps subtasks onto a large number of sensor nodes. No resource share is considered for multiple subtasks concurrently running on the same node since the module execution time and data transfer time are pre-computed. If the energy constraint and processing capability are considered, the execution of the A^* algorithm itself could drain the resources of a sensor node. A greedy A^* algorithm, which is specifically designed to reduce the complexity of the A^* algorithm, explores only the least-cost path of the search tree in the solution space, instead of searching all feasible paths, assuming that the optimal solution is most likely to be found on this path. The complexity of the greedy A^* algorithm is $O(m^2 + kn(m-k))$, where m is the number of modules, n is the number of sensor nodes, and k is the number of modules in the independent set.

8.2.2 Streamline Algorithm

Agarwalla *et al.* proposed a grid scheduling algorithm, Streamline, for graph-structured dataflow scheduling in a network with n resources and $n \times n$ communication links [26]. The Streamline algorithm considers application requirements in terms of per-stage computation and communication needs, application constraints on co-location of stages (node reuse), and availability of computation and communication resources. Two parameters, *rank* and

blevel, are used to quantitate these features: *rank* calculates the average computation and communication cost of a stage, and *blevel* estimates the overall remaining execution time of a data item after being processed by a stage. Based on these two parameters, the stages are sorted in a decreasing order of resource needs and the nodes are sorted in a decreasing order of resource availability. This scheduling heuristic works as a global greedy algorithm that expects to maximize the throughput of an application by assigning the best resources to the most needy stages in terms of computation and communication requirements at each step. The complexity of this algorithm is $O(mn^2)$, where m is the number of modules in the workflow and n is the number of nodes in the network.

8.2.3 Dynamic Level Scheduling Algorithm

Dynamic Level Scheduling (DLS) [125] is a compile-time scheduling strategy that accounts for inter-processor communication overheads when mapping precedence graphs onto multiple processor architectures. The complexity of DLS is $O(n^3 + n^2 p \cdot f(p))$, where n is the number of nodes (the same as the modules in our cost models), p is the number of processors, and the function used to route a path between two given processors on the targeted architecture is $O(f(p))$. The DLS algorithm uses the classic list scheduling formulation where nodes are assigned priorities, placed in a list, and sorted in order of decreasing priority. Whenever a processor is available, the algorithm assigns it the first ready node in the list for execution, and delete the node and the processor from the priority list and the available processor list, respectively. If the list of ready nodes is empty, or the set of available processors is exhausted, the global time clock is incremented until new ready nodes come or some processors finish execution of their assigned tasks and are available once again.

8.2.4 Naive Greedy Algorithm

A greedy algorithm iteratively obtains the greatest immediate gain based on certain local optimality criteria at each step, which may or may not lead to the global optimum. We design a heuristic mapping scheme based on a naive greedy algorithm that calculates the ED or FR of mapping a new module to the current node or one of its succeeding neighbor nodes and chooses the optimal one based on computation and communication cost. This greedy algorithm makes a module mapping decision at each step only based on the current information without considering the effect of this local decision on the mapping performance in later steps. The complexity of this algorithm is $O(|E_w| \cdot |E_c|)$, where $|E_w|$ is the number of dependency edge in the workflows and $|E_c|$ is the number of links in the computer network.

8.3 Performance Evaluation for Linear Pipelines

We perform pipeline mapping experiments under 20 different random problem sizes using ELPC, Streamline, and Greedy, by varying the mapping constraints, respectively. The measured execution time of these algorithms varies from milliseconds for small-scale problems to seconds for large-scale ones. A set of typical performance measurements in terms of MED and MFR collected in 20 different cases are tabulated in Table 8.1 and Table 8.2 for comparison. Note that we omit $|E_w|$ in the second column because $|E_w| = m - 1$ in pipeline-shaped workflows.

The relative performance differences among these three algorithms observed in other cases are qualitatively similar. In unitary processing applications that minimize ED for fast response, we allow network nodes to be reused but there is only one module executing on a selected node at any time, while in streaming applications that identify and minimize the bottleneck node or link for the smoothest data flow, node reuse will cause resource sharing by a subsequent module allocated to a used node and hence affect the optimality of module mapping carried out in the previous steps in the 2D DP table.

Table 8.1: Performance comparisons of MED among three algorithms under different constraints in pipeline mapping.

Prb Idx	m, n, $\|E_c\|$,	MED-ANR (*milliseconds*)			MED-CNR (*milliseconds*)		MED-NNR(*milliseconds*)		
		ELPC	Greedy	Streamline	ELPC	Greedy	ELPC	Greedy	Streamline
1	4, 6, 29	33.31	33.31	36.44	89.59	105.86	90.12	89.91	103.15
2	6, 10, 86	52.65	58.97	92.08	85.80	111.19	122.83	246.57	340.11
3	10, 15, 207	95.69	95.69	226.96	97.28	139.83	204.82	220.37	439.11
4	13, 20, 376	106.02	147.43	144.48	96.84	96.85	227.39	252.50	557.03
5	15, 25, 597	144.12	144.12	174.37	153.66	219.68	349.35	378.49	548.19
6	19, 28, 753	166.76	166.76	267.88	128.20	128.21	341.78	362.34	931.62
7	22, 31, 927	196.26	196.26	255.04	175.71	181.26	439.21	485.46	925.35
8	26, 35, 1180	194.02	194.02	340.13	205.62	214.21	495.95	528.09	727.95
9	30, 40, 1558	247.92	247.92	505.91	228.67	263.58	551.77	586.55	1346.74
10	35, 45, 1963	309.66	309.66	495.82	255.64	271.54	664.16	712.59	1424.02
11	38, 47, 2153	284.01	284.01	451.73	281.68	401.21	699.79	776.09	1148.48
12	40, 50, 2428	320.03	320.03	377.18	385.86	609.92	716.60	811.62	1161.81
13	45, 60, 3520	361.32	361.32	630.51	394.91	452.03	864.32	924.21	1592.16
14	50, 65, 4155	386.21	386.21	537.89	436.48	486.37	881.78	920.12	2094.98
15	55, 70, 4820	417.29	417.29	949.11	557.42	682.77	960.48	1050.60	1584.84
16	60, 75, 5540	421.76	751.70	834.16	548.25	571.79	1087.21	1232.20	2127.81
17	75, 90, 7990	599.00	599.00	729.48	658.31	706.55	1392.05	1503.61	3206.72
18	80, 100, 9896	648.79	674.36	979.86	744.00	753.38	1564.58	1673.24	3248.07
19	90, 150, 22326	700.54	744.47	1340.25	746.07	805.52	1484.32	1594.28	2811.84
20	100, 200, 39790	752.46	760.73	2057.51	789.81	850.87	1552.42	1604.69	3325.08

Table 8.2: Performance comparisons of MFR among three algorithms under different constraints in pipeline mapping.

Prb Idx	m, n, $\|E_c\|$,	MFR-ANR ($frames/second$)			MFR-CNR ($frames/second$)		MFR-NNR($frames/second$)		
		ELPC	Greedy	Streamline	ELPC	Greedy	ELPC	Greedy	Streamline
1	4, 6, 29	22.10	10.97	4.75	11.36	11.36	33.47	10.97	28.15
2	6, 10, 86	36.50	36.50	26.31	48.54	39.19	47.93	36.50	6.19
3	10, 15, 207	63.33	59.31	4.18	60.15	54.06	54.33	24.81	6.10
4	13, 20, 376	76.01	56.67	26.50	54.21	12.58	56.67	30.44	5.68
5	15, 25, 597	36.98	35.33	26.23	51.25	46.18	37.64	5.07	8.48
6	19, 28, 753	48.92	48.92	48.90	54.21	50.07	48.92	19.77	3.21
7	22, 31, 927	67.95	53.74	57.82	49.52	32.62	47.62	25.85	6.51
8	26, 35, 1180	61.52	57.90	47.41	21.42	14.61	38.70	30.95	10.33
9	30, 40, 1558	96.49	57.90	13.63	40.61	6.13	52.51	32.13	2.42
10	35, 45, 1963	36.46	36.46	27.17	32.19	7.70	36.46	24.09	3.65
11	38, 47, 2153	45.09	15.18	45.09	61.45	57.86	45.09	24.97	7.69
12	40, 50, 2428	27.43	27.43	27.42	60.57	55.57	27.43	25.65	8.07
13	45, 60, 3520	61.01	49.53	55.24	34.34	32.25	52.59	18.61	4.62
14	50, 65, 4155	89.65	59.70	56.56	52.50	16.99	43.92	10.61	2.10
15	55, 70, 4820	50.80	50.80	35.45	40.97	34.13	50.80	22.20	5.85
16	60, 75, 5540	94.00	53.82	45.19	50.23	49.92	43.41	25.19	3.95
17	75, 90, 7990	15.35	15.35	15.35	50.52	50.02	15.35	29.04	1.17
18	80, 100, 9896	32.60	28.90	32.59	54.25	32.38	32.60	18.32	6.96
19	90, 150, 22326	88.95	55.49	24.47	56.07	41.54	55.63	35.58	6.52
20	100, 200, 39790	84.94	55.98	20.68	56.86	34.31	56.70	18.35	4.86

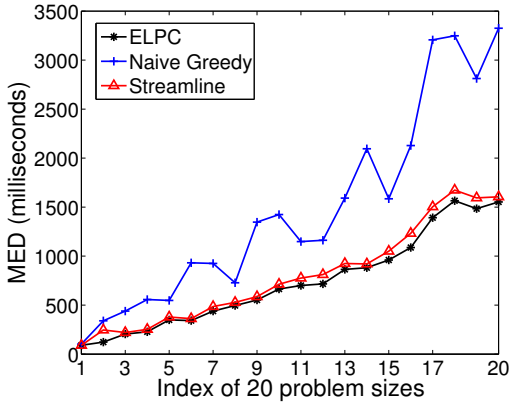


Figure 8.1: Performance comparison for MED-NNR.

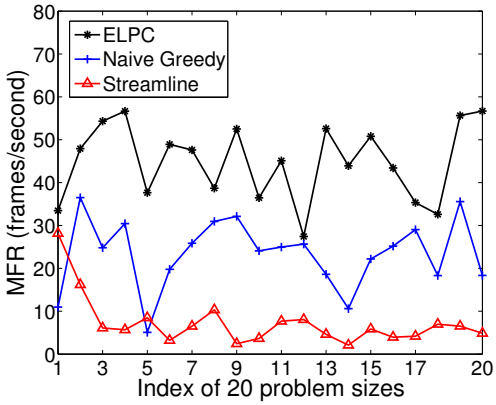


Figure 8.2: Performance comparison for MFR-NNR.

For a visual comparison, we plot the performance measurements of MED and MFR produced by these three algorithms under different constraints in Figs. 8.1, 8.2, 8.3, 8.4, 8.5, and 8.6, respectively. We observed that ELPC exhibits comparable or superior performances over the other two algorithms in all the cases we studied. We did not compare with Streamline in the case of contiguous node reuse because Streamline does not allocate the resources by the modules' sequence number so we may not know the previous module when the current one is being allocated.

Since the MED represents the total delay from source to destination, a larger problem size with more network nodes and computing modules generally (not absolutely, though) incurs a longer mapping path resulting in a longer end-to-end delay, which explains the increasing trend in Figs. 8.1, 8.3, and 8.5. The MFR, the reciprocal of the bottleneck in a selected path, is not particularly related to the path length, and hence the performance curves in Figs. 8.2, 8.4, and 8.6 lack an obvious increasing or decreasing trend in response to varying problem sizes.

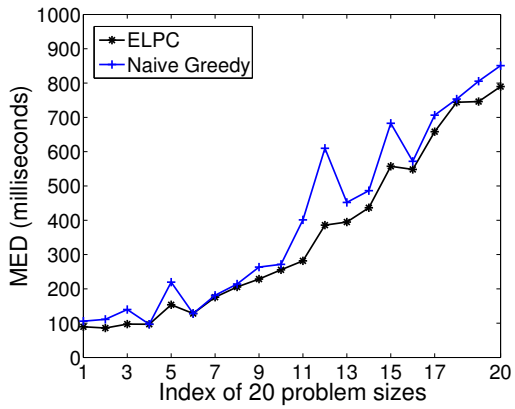


Figure 8.3: Performance comparison for MED-CNR.

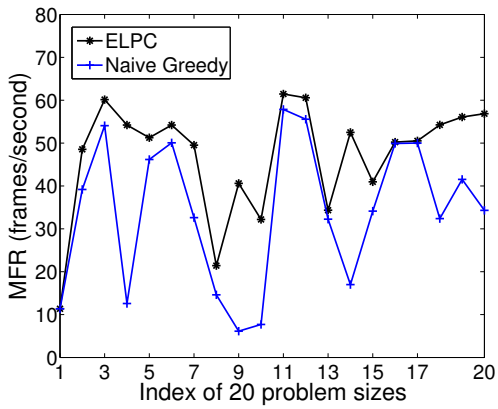


Figure 8.4: Performance comparison for MFR-CNR.

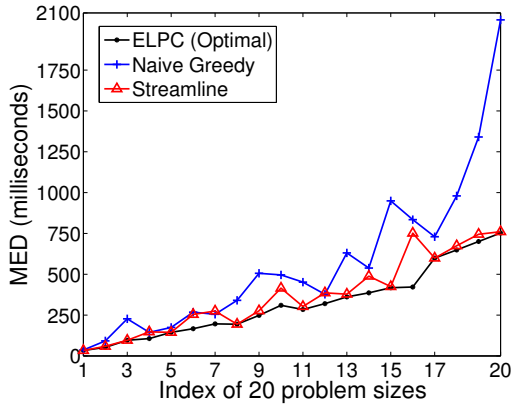


Figure 8.5: Performance comparison for MED-ANR.

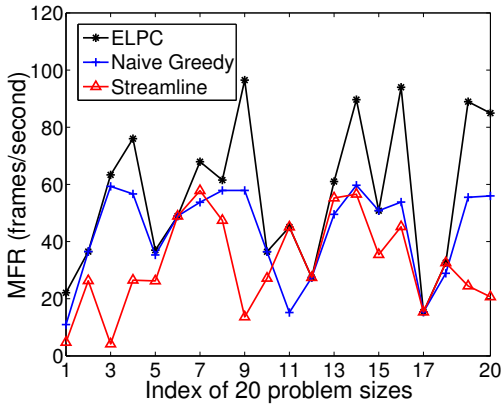


Figure 8.6: Performance comparison for MFR-ANR.

8.4 Performance Evaluation for DAG-structured Workflows

We conduct extensive performance evaluations and comparisons for the proposed mapping algorithms using either simulated workflows and networks or simulated workflows and real networks. The comparisons for real-life workflows and real networks are presented in Chapter 8.6 on the evaluation of the SWAMP system using two use cases.

8.4.1 Performance Evaluation of extED

To verify the accuracy of the extED algorithm, we compare our extED solution with three other approaches for a given mapping scheme:

- appED: An approximate estimate used in [138], where the number of concurrently running modules on any node is assumed to be the total number of independent modules mapped on that node;
- simED: A simulation-based measurement using the SDEDS simulation program in [146], which takes two graphs as input: a DAG that represents a workflow and an arbitrary directed weighted graph that represents a network. Based on a given mapping scheme, SDEDS overlays the computing workflow on the computer network, and visually illustrates the dynamic execution process;
- expED: An experimental result measured in the real computer network environment using SWAMP.

We randomly generate 10 workflow instances with different topologies and sizes as shown in Table 8.3, where the network parameters are not provided because we use one testbed with 10 computer nodes and 88 network links but different network topologies by configuring the firewall settings on the nodes. In each test case, we first run the impRCP

algorithm to compute a mapping scheme, based on which, we calculate the theoretical values of extED and appED. Then, we simulate the execution of the corresponding workflow in SDEDS and also deploy it through SWAMP for execution in the real network under the same mapping scheme. We collect the simED and expED measurements in SDEDS and SWAMP, respectively.

Table 8.3: ED comparison among extED, simED, expED and appED.

Prb Idx	Workflow Size m, $\|E_w\|$	End-to-end Delay (*seconds*)			
		extED	simED	expED	appED
1	4, 5	187.54	203.91	204	271.55
2	6, 10	298.65	299.03	304	298.65
3	10, 18	384.80	402.96	410	517.71
4	15, 30	698.67	710.72	707	1677.02
5	22, 44	1273.26	1290.94	1373	2279.31
6	30, 62	1327.83	1342.20	1405	2626.25
7	35, 70	1390.05	1484.06	1586	2966.77
8	40, 78	1406.47	1490.00	1513	2381.61
9	45, 96	1581.13	1663.38	1689	3340.16
10	50, 102	1748.04	1801.48	1828	4100.91

We tabulate the ED calculations or measurements in Table 8.3 and plot the corresponding performance curves in Fig. 8.7. We observe that the proposed extED is very close to the SDEDS-based simED and SWAMP-based expED, which indicates:

- the accuracy of the extED calculation,
- the validity of the cost models,
- the accuracy of the objective functions, and
- the correctness of the SDEDS and SWAMP implementation.

The slight discrepancy is mainly caused by the visualization and system overhead in SDEDS and SWAMP. The appED is larger than the extED in most cases because in the approximation solution, for each module, we use the number of all mapped modules on the node

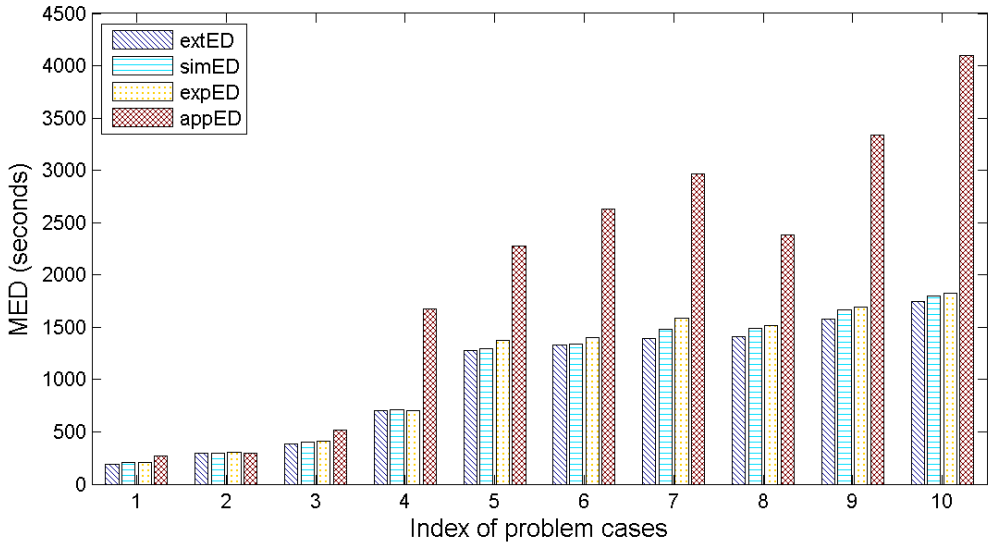


Figure 8.7: Performance comparison among extED, simED, expED and appED results.

to denote the number of concurrent module executions on that node at all times. However, the appED requires much less execution time, especially in large problem sizes, and hence could be applied to applications that do not require a high level of accuracy but a prompt response. Note that the expED results are measured in the unit of seconds due to the time resolution in the Condor log files of SWAMP. Due to the limited availability of physical computer resources, the problems of large scales are not tested in real networks.

8.4.2 Performance Evaluation for MED Algorithms

RCP

We investigate the robustness of the RCP mapping algorithm by randomly generating 10 test cases with different problem sizes, for each of which, we randomly generate 20 problem instances and run four mapping algorithms on them, namely RCP, Greedy A^*, Streamline, and Greedy. We then calculate and plot the mean value and standard deviation over 20 instances for each problem size in Fig. 8.8. Note that the value range along y axis is

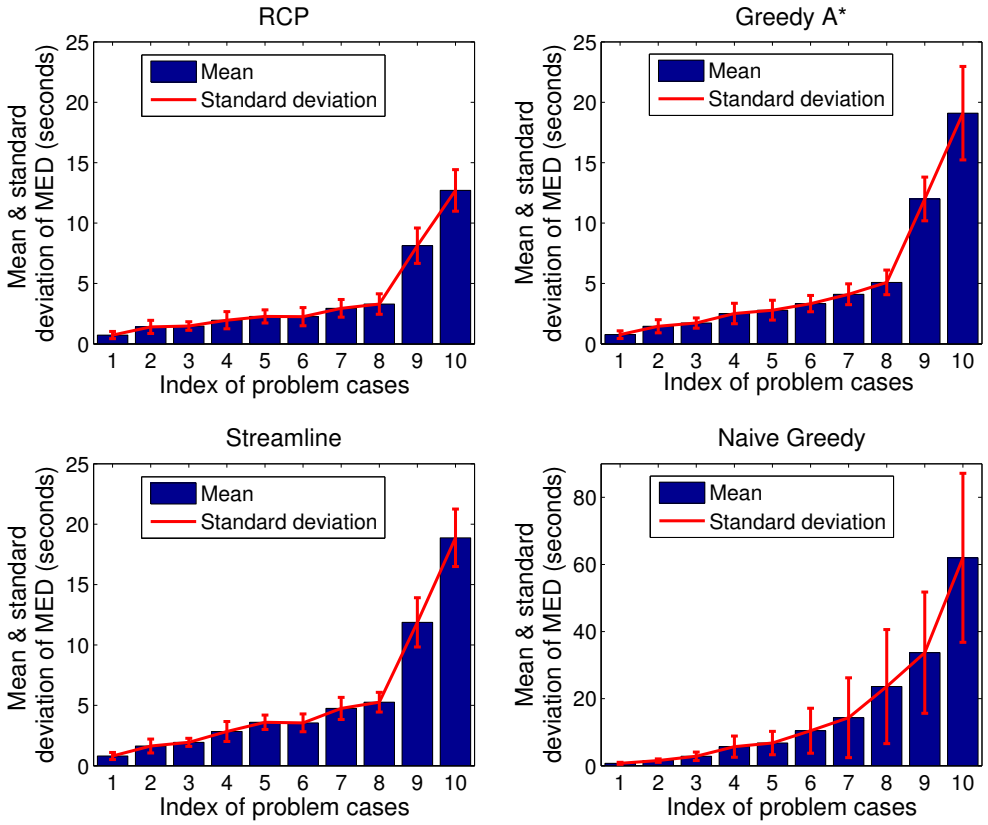


Figure 8.8: Mean and standard deviation of MED performance of four algorithms.

from 0 to 90 seconds for Naive Greedy while it falls between 0 and 25 seconds for others. We observed that RCP achieves the best MED performance in a statistical sense with the smallest standard deviation, which demonstrates the performance robustness and optimization stability of RCP in achieving MED in various workflows and networks of disparate topologies and different scales.

In order to investigate the convergence property of RCP, we plot the MED optimization curve collected over 8 recursions for one large problem size with 100 modules and 200 nodes in Fig. 8.9, where x axis represents recursion steps and y axis represents the corresponding MED measurements. The significant performance improvement in the first recursion is due to the fact that the algorithm's optimization process starts from an initial

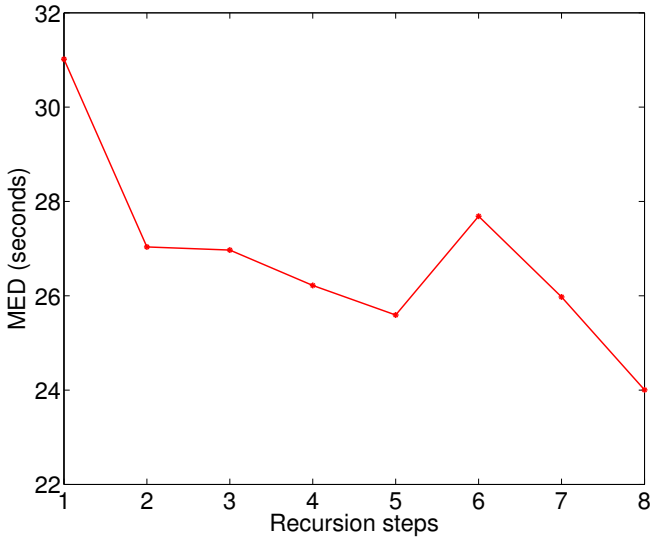


Figure 8.9: MED performance optimization curve of RCP algorithm.

state that assumes uniform resources. In each recursion, the CP calculated in the previous recursion is guaranteed by the optimal pipeline mapping algorithm to have a reduced ED. However, the branch modules mapping may change the CP, hence resulting in variations in the performance optimization curve. The optimization process converges to a certain point where the CP remains the same as the previous recursion. These optimization patterns are typical and those observed in other cases are qualitatively similar.

impRCP

To evaluate the mapping performance of the proposed impRCP algorithm [137] and the original RCP algorithm in [138], we randomly generate 40 problem cases (40 pairs of simulated workflows and networks), indexed from 1 to 40 by varying the topology and problem size (m, $|E_w|$, n, $|E_c|$) from (4, 6, 6, 29) to (55, 124, 70, 4820).

For a visual performance comparison, we plot in Fig. 8.10 the MED performance measurements in these 40 problem cases ranging from small to large scales collected from (i) the proposed impRCP mapping solution that uses A^*- and Beam Search-based algorithms

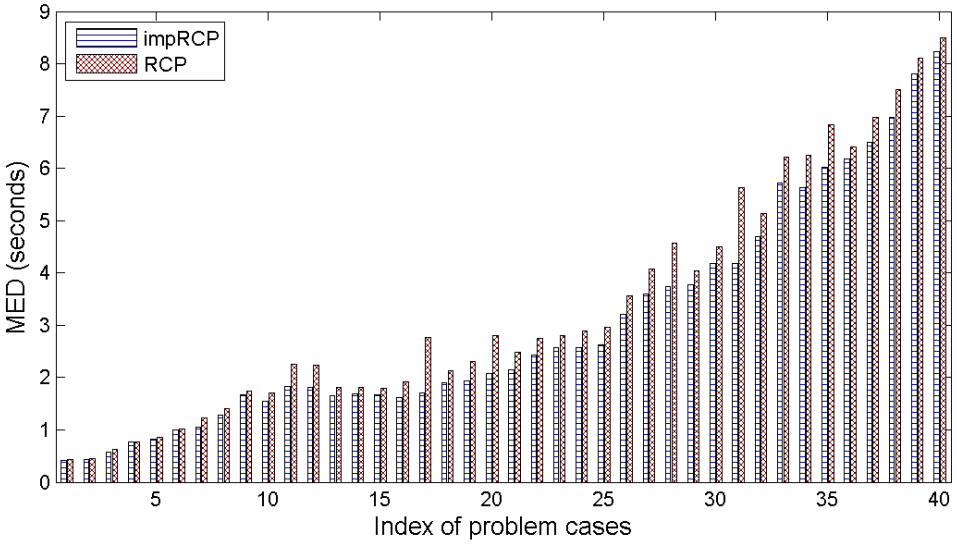


Figure 8.10: MED comparison between impRCP and RCP.

for non-critical module mapping and uses extED for ED calculation, and (ii) the original RCP algorithm that uses a greedy approach for non-critical module mapping and uses extED for ED calculation. We observe that the average MED speedup of impRCP (A^* + Beam Search + extED) over RCP (Greedy + extED) is around 6% in these cases. Here, we define the speedup of solution A over solution B as $|\frac{B-A}{B}| \times 100\%$ (this definition of speedup is applied to the rest of the evaluation). In small problem sizes, all of these algorithms tend to find or approach the optimal mapping schemes and hence their differences are not significant. The slight variation in the curves is caused by the random assignment of different parameter values in a given problem size. For simplicity, the value of ρ used in the simulations is set to be 2. Note that the $h(x)$ part becomes a greedy approach if $\rho = 1$ and an exhaustive search if ρ is equal to the total number of paths from the current step to the destination.

To further illustrate the superiority of the proposed impRCP mapping solution, we also compare it with other three graph-mapping algorithms, namely Greedy A^*, Streamline, and Greedy. The MED measurements of 15 problem sizes indexed from 1 to 15 are tabulated

Table 8.4: MED measurements of four algorithms.

Prb Idx	Prb Size m, $\|E_w\|$, n, $\|E_c\|$	MED (*seconds*)			
		impRCP	GreedyA^*	Streamline	Greedy
1	10, 18, 15, 207	1.15	1.61	1.65	1.94
2	15, 30, 25, 597	1.53	1.86	1.78	1.59
3	19, 36, 28, 753	1.96	2.71	2.72	3.03
4	26, 50, 35, 1180	1.85	2.79	4.00	2.56
5	30, 62, 40, 1558	3.56	5.06	5.22	4.45
6	35, 70, 45, 1963	4.05	4.37	5.79	7.02
7	38, 73, 47, 2153	4.62	5.78	6.89	6.92
8	40, 78, 50, 2428	2.01	2.97	3.13	2.71
9	45, 96, 60, 3520	3.64	5.13	5.92	5.17
10	50, 102, 65, 4155	2.80	5.12	4.37	4.82
11	55, 124, 70, 4820	4.49	6.86	6.63	7.46
12	75, 369, 90, 7990	15.41	18.25	20.53	17.92
13	80, 420, 100, 9896	19.81	25.04	26.20	22.71
14	90, 500, 150, 22346	24.01	28.83	28.18	25.84
15	100, 660, 200, 39790	24.79	34.71	29.78	26.62

in Table 8.4 and plotted in Fig. 8.11, respectively. We observe that the impRCP algorithm consistently outperforms all other mapping algorithms and the average MED speedup over the other algorithms is around 10% in the cases we studied.

Note that the Greedy A^* algorithm is different from the A^* algorithm used for non-critical module mapping. In [121], a Greedy A^* approach is used to map the subtasks of a DAG-structured workflow onto a large number of nodes while in our work, the A^* algorithm is only used for non-critical module mapping and the mapping scheme is produced by a recursive CP mapping procedure. The performance improvements of impRCP over other algorithms become more significant as the workflow and network sizes increase, which is the trend of real-life scientific applications and rapidly growing network environments.

We conduct another set of performance comparison in Table 8.5 for impRCP with one more workflow mapping/scheduling algorithm, DLS. We randomly generate15 workflow mapping problems indexed from 1 to 15 from small to large scales, for each of which, we

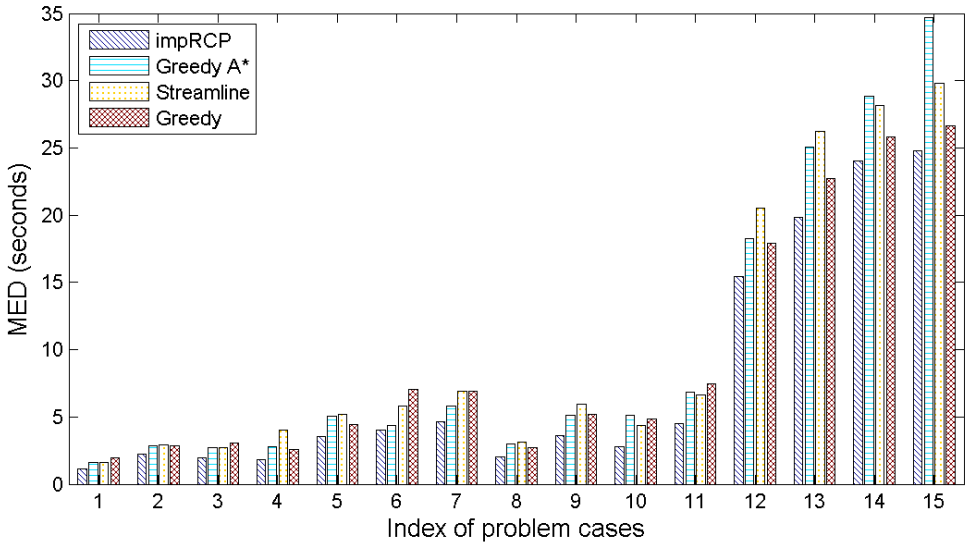


Figure 8.11: MED comparison among four algorithms.

run these five mapping algorithms. We observe that the proposed impRCP algorithm consistently outperforms the other methods in comparison in the above two sets of comparisons we conducted.

disRCP-$\mathscr{F}$

In the evaluation of disRCP-$\mathscr{F}$ in faulty network environments, we set the OFR bound $\mathbb{F}$ to be 5%, i.e., the OFR of each calculated mapping scheme must be less than 5%.

The MED measurements in 18 problem sizes under the fault-tolerance constraint of 5% are tabulated in Table 8.6. We observe that the algorithms in comparison fail to find feasible mapping solutions in some cases (such missing solutions are marked as "——"). In most cases, disRCP-$\mathscr{F}$ is able to find a feasible solution for the mapping problem with a satisfied OFR and consistently exhibits the best performance in terms of MED among all the algorithms in comparison. Moreover, the Greedy A^*, Streamline and Greedy algorithms are more likely to find invalid mappings because they do not take network topology into consideration.

Table 8.5: MED comparison among five algorithms.

Prb Idx	Prb Size m, $\|E_w\|$, n, $\|E_c\|$	MED (*seconds*)				
		impRCP	DLS	GreedyA^*	Streamline	Greedy
1	4,6,6,29	0.64	0.65	0.65	0.65	0.65
2	6,10,10,86	1.08	1.12	1.10	1.08	1.12
3	10,18,15,207	1.53	1.54	1.55	1.56	1.54
4	13,24,20,376	1.06	1.10	1.17	1.23	1.13
5	15,30,25,597	1.22	1.35	1.42	1.69	1.34
6	19,36,28,753	1.77	1.80	1.81	2.87	1.82
7	22,44,31,927	1.99	2.03	2.08	2.18	2.08
8	26,50,35,1180	2.52	2.53	2.71	3.59	2.74
9	30,62,40,1558	2.48	2.52	2.58	5.00	2.60
10	35,70,45,1963	2.80	2.92	2.82	4.13	2.92
11	38,73,47,2153	3.32	3.35	3.38	5.66	3.39
12	40,78,50,2428	3.31	3.33	3.64	4.59	3.33
13	45,96,60,3520	4.55	4.57	5.06	6.18	4.58
14	50,102,65,4155	3.27	3.30	3.47	6.87	3.47
15	55,124,70,4820	7.89	7.93	8.01	9.42	7.95

For a visual comparison, we remove those problem cases with invalid mapping solutions (those 5 cases whose cells with "——" in Table 8.6), and plot the performance curves in Fig. 8.12. The slight variations in Fig. 8.12 are due to the fact that the network topologies and parameters of different problem sizes are randomly generated for the simulated workflows and networks. Therefore, a larger problem size may not always incur a longer delay, which also depends on the module complexity, node computing power, and mapping scheme. However, a larger problem size involves more computing modules and dependency edges, which more likely lead to a longer completion time.

8.4.3 Performance Evaluation for MFR Algorithms

LDP

We investigate the MFR performance of the Greedy LDP algorithm in comparison with the Greedy A^*, Streamline, and Greedy algorithms using a large set of simulated workflows

Table 8.6: MED measurements of four algorithms with $\mathbb{F}$=5%.

Prb Idx	Prb Size $m,\|E_w\|,n,\|E_c\|$	MED (*seconds*)			
		disRCP-$\mathscr{F}$	Greedy A^*	Streamline	Greedy
1	4,6,6,28	0.5659	0.5902	0.5902	0.5902
2	6,10,10,88	0.6630	0.8839	0.8905	0.8839
3	10,18,15,207	1.6848	1.8997	1.8884	2.1225
4	13,24,20,379	1.7745	2.0556	1.9698	2.3033
5	15,30,25,599	1.4719	1.8476	1.8705	1.8673
6	19,36,28,755	2.8010	3.2754	3.0622	3.1373
7	22,44,31,928	2.9538	3.3653	3.7297	3.3780
8	26,50,35,1188	3.0240	3.8249	3.5387	3.8861
9	30,62,40,1557	4.6191	4.6878	4.8902	5.0372
10	35,70,45,1978	3.3974	3.7288	3.5518	4.1925
11	38,73,47,2160	6.9154	—	7.2445	7.0181
12	40,78,50,2448	5.6532	6.1811	—	6.6803
13	45,96,60,3537	4.4895	5.0692	4.7644	5.7309
14	50,102,65,4158	4.7687	6.1577	5.4772	5.8479
15	55,124,70,4828	6.6224	7.8459	8.0041	8.8298
16	60,240,75,5549	12.3642	—	—	—
17	75,369,90,8009	14.3729	—	—	—
18	80,420,100,9897	16.1632	19.6071	19.0434	—

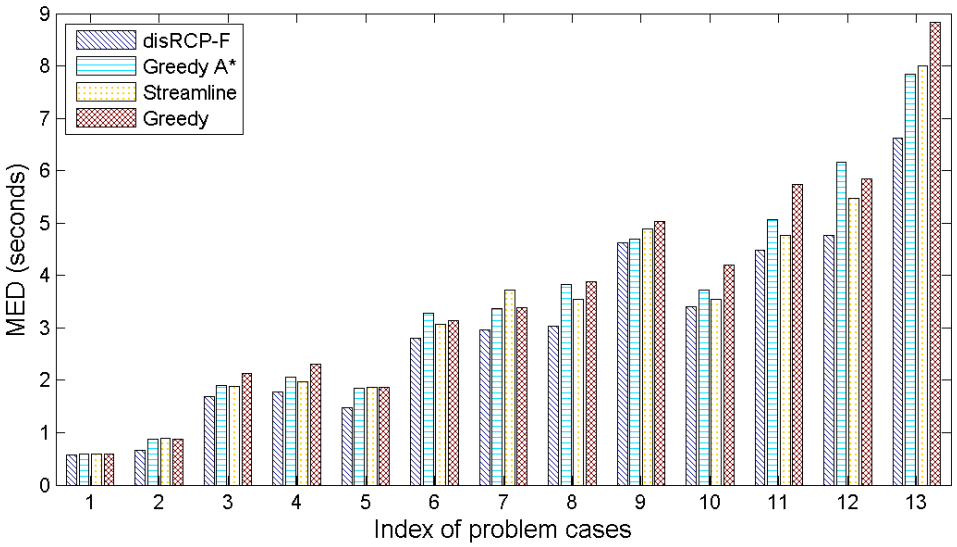


Figure 8.12: MED comparison under fault-tolerance constraint $\mathbb{F}$=5%.

Table 8.7: MFR measurements of four algorithms in comparison.

Prb Idx	Prb Size m, $\|E_w\|$, n, $\|E_c\|$	MFR ($frames/second$)			
		Greedy LDP	Greedy A^*	Streamline	Greedy
1	4, 6, 6, 29	1.025	0.980	0.980	0.980
2	6, 10, 10,86	1.854	1.201	1.201	1.201
3	10, 18, 15, 207	1.544	1.325	1.072	1.325
4	13, 24, 20, 376	1.716	——	1.048	——
5	15, 30, 25, 597	1.724	1.259	0.684	1.154
6	19, 36, 28, 753	1.116	1.050	0.668	——
7	22, 44, 31, 927	1.692	——	1.164	1.674
8	26, 50, 35, 1180	0.946	0.774	0.734	0.801
9	30, 62, 40, 1558	1.393	0.972	0.780	0.736
10	35, 70, 45, 1963	0.812	——	——	0.677
11	38, 73, 47, 2153	0.863	0.683	0.502	0.675
12	40, 78, 50, 2428	1.461	1.230	0.730	1.217
13	45, 96, 60, 3520	0.885	0.800	——	0.752
14	50, 102, 65, 4155	1.911	1.145	1.191	0.774
15	55, 124, 70, 4820	0.836	0.593	0.569	——
16	60, 240, 75, 5540	0.521	0.392	0.391	0.396
17	75, 369, 90, 7990	0.373	0.238	——	0.241
18	80, 420, 100, 9896	0.333	0.205	0.220	0.205
19	90, 500, 150, 22346	0.417	0.278	0.318	0.364
20	100, 660, 200, 39790	0.317	0.187	0.190	0.263

and networks of various scales. The MFR measurements in 20 mapping problems of various sizes and topologies indexed from 1 to 20 are tabulated in Table 8.7. We observe that Greedy LDP outperforms the other methods in all these cases. Since Greedy LDP considers both module dependency and network connectivity in workflow mapping, it is more likely to produce a valid mapping scheme (a mapping scheme is valid, feasible, or successful if both dependency and connectivity constraints are satisfied). Since both Greedy A^* and Streamline are designed for complete networks without considering any topology constraint on mapping, they may miss certain feasible mapping solutions when two dependent modules are mapped to two nonadjacent nodes. This problem can also occur in Greedy when the last selected node does not have a link to the destination node. In Table 8.7, the MFR measurement in an invalid mapping scheme is crossed out using "——".

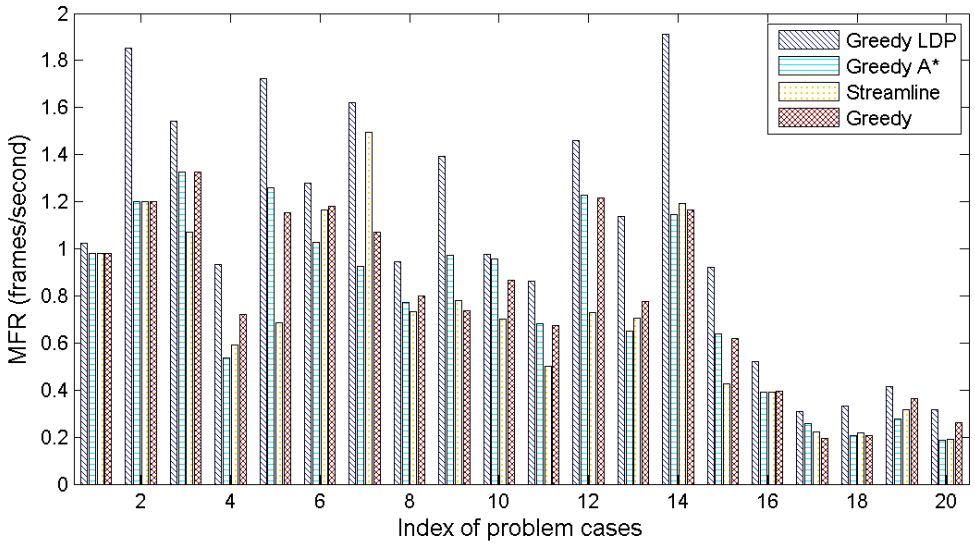


Figure 8.13: MFR measurements among four algorithms.

For an extensive comparison, we recompute those invalid mapping results using new randomly generated instances with the same problem size but different topologies and parameters. We plot the MFR measurements produced by these four algorithms in Fig. 8.13, where we observe that Greedy LDP consistently exhibits comparable or superior MFR performances over the other three algorithms, and the average MFR speedup over the other algorithms is around 25%. Since MFR is not particularly related to the problem size, these performance curves lack an obvious increasing or decreasing trend in response to increasing problem sizes. The slightly decreasing trend in Fig. 8.13 is due to the fact that a node or a link in larger problem sizes is more likely to be shared by more concurrent module executions or data transfers.

To further investigate the robustness of the proposed Greedy LDP algorithm, we randomly generate $N_{\text{total}} = 500$ problem cases of simulated workflows and networks with various topologies and sizes, and then run Greedy LDP as well as the other three algorithms for workflow mapping. For each algorithm, we count the number N_{valid} of valid mapping results and plot the mapping success rate in Fig. 8.14, which is defined as the ratio of

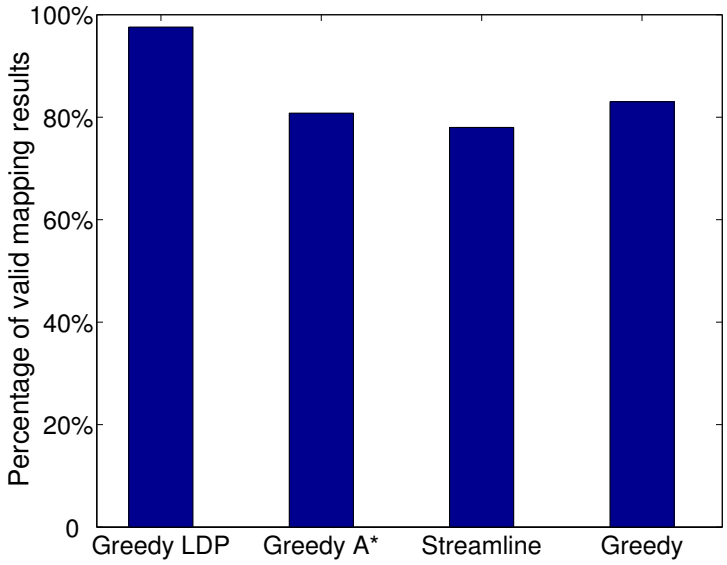


Figure 8.14: Success rate in 500 random test cases.

$\frac{N_{\text{valid}}}{N_{\text{total}}} \times 100\%$. We observe that Greedy LDP has a success rate around 98% while the other three algorithms have a success rate around 80%. These measurements indicate that Greedy LDP has a stable mapping performance due to its consideration of module dependency and network connectivity during workflow mapping.

disLDP-$\mathscr{F}$

In these experiments, we set $\mathbb{F}$ to be 6% in all cases, which means that the OFR of each calculated mapping scheme must be less than 6%.

1) *Simulated Workflows Mapped to Simulated Networks*

We conduct an extensive set of workflow mapping experiments for MFR using a large number of simulated computing workflows and computer networks. We randomly create 20 problem sizes from small-scale to large-scale, for each of which, 10 problem instances of the same size with different workflow and network topologies and parameters are generated. We apply Greedy disLDP-$\mathscr{F}$ and other three mapping algorithms in comparison to

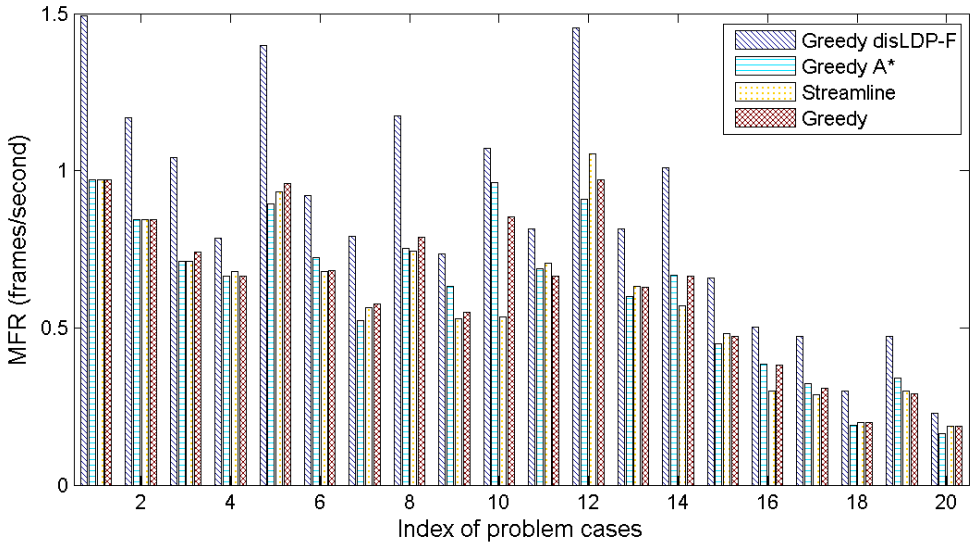


Figure 8.15: MFR comparison based on simulated workflows and simulated networks under the OFR constraint $\mathbb{F}$=6%.

each problem instance and calculate the mean values of MFR in each problem size. The MFR measurements in 20 problem sizes under the fault-tolerance constraint of 6% are plotted in Fig. 8.15, which shows that the Greedy disLDP-$\mathscr{F}$ algorithm outperforms all other algorithms and its throughput speedup ranges from 15% to 60%. The slightly decreasing trend is due to the fact that a node or a link in larger problem sizes is more likely to be shared by more concurrent module executions or data transfers. Note that larger problem sizes with more nodes and links are more likely to have higher failure rates.

For a visual illustration, we plot in Fig. 8.16 different mapping schemes produced by these four algorithms in the smallest problem instance with 4 modules and 6 nodes. In this problem instance, Streamline and Greedy obtain the same mapping scheme, and disLDP-$\mathscr{F}$ outperforms Greedy A^* in terms of MFR with the same OFR of 0.07%. In most of the larger problem instances, these algorithms produce different mapping schemes. We also observe that some algorithms in comparison are not able to produce feasible mapping solutions in some cases. In all the cases we studied, disLDP-$\mathscr{F}$ has found feasible mapping

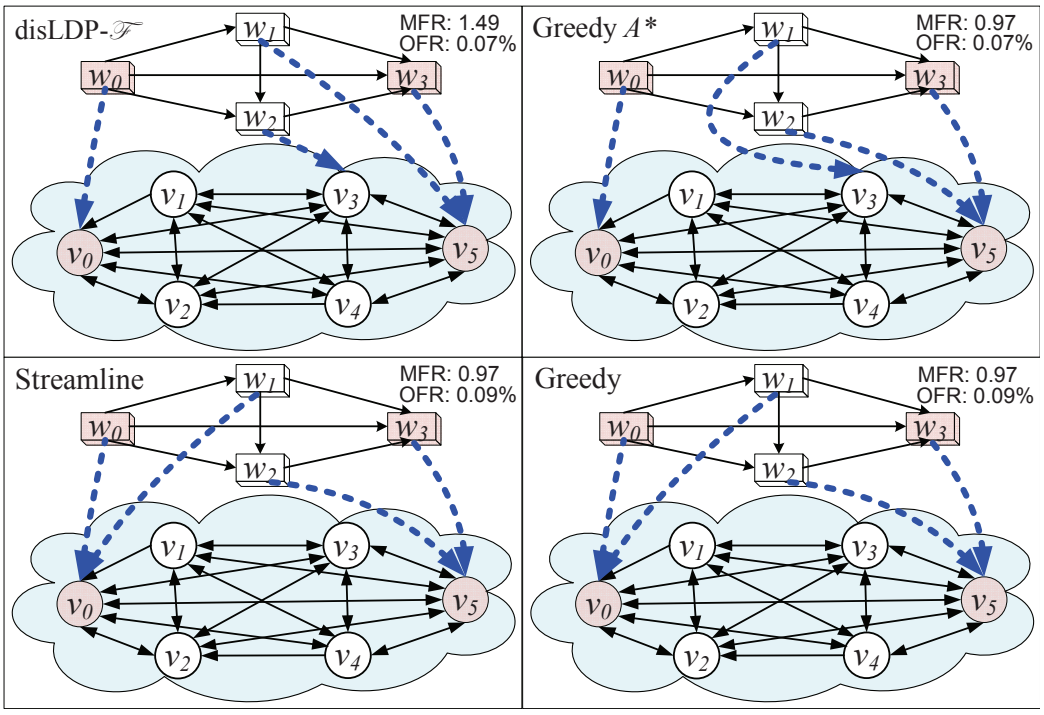


Figure 8.16: Mapping schemes produced by four algorithms in the first problem instance with 4 modules and 6 nodes.

solutions with a satisfied OFR and consistently exhibits the best performance in terms of MFR.

To illustrate the performance superiority of disLDP-$\mathscr{F}$ in terms of OFR, we run these four algorithms on another set of 10 randomly generated problem instances of different sizes and collect both MFR and OFR measurements as tabulated in Table 8.8, which shows that Greedy disLDP-$\mathscr{F}$ also achieves the highest FRs and the lowest OFRs in all these cases. We observe that larger problem sizes with more nodes and links are more likely to suffer from lower FRs and higher OFRs, which is consistent with the failure pattern observed in practical systems.

To further investigate the robustness of Greedy disLDP-$\mathscr{F}$, we randomly generate $N_{\text{total}} = 1500$ test cases of simulated workflows and networks with different topologies and sizes, and then run these mapping algorithms, for each of which, we count the number

Table 8.8: MFR (frames/second) and OFR (%) measurements under the OFR constraint $\mathbb{F}$=6%.

Prb	Prb Size	Greedy disLDP-$\mathscr{F}$		Greedy A^*		Streamline		Greedy	
Idx	m, $\|E_w\|$, n, $\|E_c\|$	MFR	OFR	MFR	OFR	MFR	OFR	MFR	OFR
1	15, 30, 25, 599	1.9089	0.29	1.8194	0.33	1.2882	0.32	1.4822	0.31
2	19, 36, 28, 755	1.5342	0.34	1.3289	0.38	1.4673	0.44	0.9880	0.39
3	26, 50, 35, 1188	1.8427	0.49	1.1989	0.61	1.2502	0.55	1.2133	0.54
4	35, 70, 45, 1979	1.4578	0.73	0.7848	0.76	1.4267	0.77	1.2969	0.75
5	40, 78, 50, 2448	1.1257	0.89	0.7454	0.91	1.0722	0.96	0.8945	0.95
6	45, 96, 60, 3539	1.1319	0.94	0.7019	1.04	0.7161	1.05	1.1043	1.02
7	50, 102, 65, 4158	0.6758	1.11	0.3418	1.22	0.4612	1.27	0.2418	1.22
8	75, 369, 90, 8009	0.3272	3.28	0.3018	3.34	0.2880	3.33	0.2448	3.30
9	90, 500, 150, 22348	0.3517	4.24	0.2750	4.42	0.2227	4.45	0.3058	4.43
10	100, 660, 200, 39799	0.3087	5.33	0.2886	5.38	0.1768	5.44	0.1869	5.51

N_{valid} of valid mapping results and plot in Fig. 8.17 the mapping success rate, which is defined as $\frac{N_{\text{valid}}}{N_{\text{total}}} \times 100\%$. We observe that Greedy disLDP-$\mathscr{F}$ has a mapping success rate of about 98% while that of the other algorithms is below 50%. These measurements indicate that Greedy disLDP-$\mathscr{F}$ has a stable mapping performance due to its consideration of both module dependency and network connectivity in the mapping process.

2) *Simulated Workflows Mapped to and Executed in Real Networks*

We create 10 simulated workflow instances with different workflow sizes from small to large scales by varying the number of computing modules and dependency edges. To make these simulated workflows executable in real networks, we implement each simulated module using a real program with the same complexity as defined for that module. We execute these simulated computing workflow instances through the SWAMP workflow engine in the real wide-area network testbed using different mapping schemes generated by Greedy disLDP-$\mathscr{F}$, Greedy A^*, Streamline and Greedy, respectively.

For each workflow instance, we execute it in the SWAMP system for 10 times, and calculate the mean value of the performance measurements to denote the MFR of that workflow instance. The experimental results in terms of MFR are tabulated in Table 8.9 and also plotted in Fig. 8.18. We observe that Greedy disLDP-$\mathscr{F}$ algorithm produces the best MFR results among all these algorithms in comparison while satisfying the OFR constraint.

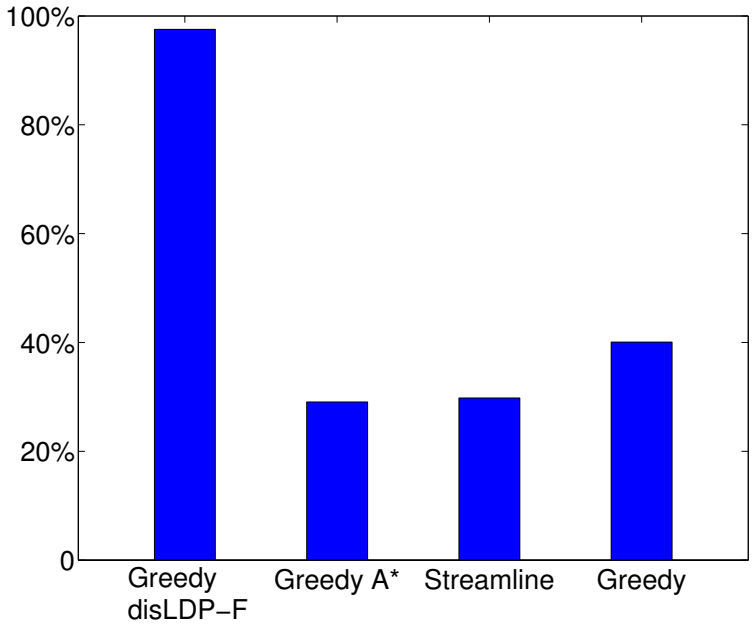


Figure 8.17: Mapping success rate in 1500 test cases under the OFR constraint $\mathbb{F}$=6%.

Table 8.9: Experimental results of simulated workflows executed in the SWAMP system.

Prb Idx	Workflow Size m, $\lvert E_w \rvert$	MFR ($frames/minute$)			
		Greedy disLDP-$\mathscr{F}$	Greedy A^*	Streamline	Greedy
1	10, 18	2.143	1.875	2.000	1.463
2	13, 24	2.142	2.069	1.429	1.154
3	15, 30	1.333	0.952	0.779	0.750
4	22, 44	2.222	0.690	0.845	1.224
5	30, 62	1.500	0.923	0.706	0.480
6	35, 70	1.463	0.811	1.277	1.250
7	40, 78	0.952	0.638	0.321	0.682
8	45, 96	0.632	0.462	0.522	0.287
9	50, 102	0.652	0.500	0.377	0.448
10	60, 240	0.373	0.267	0.211	0.291

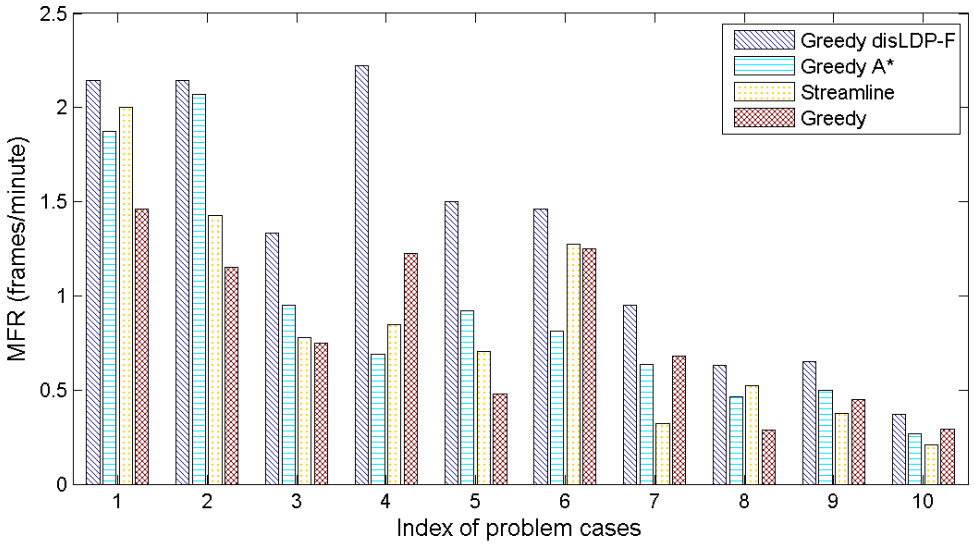


Figure 8.18: MFR comparison based on simulated workflows executed in real networks using SWAMP.

Although the workflow topology is randomly generated and the input data size is also randomly assigned, a larger workflow is still more likely to produce a smaller MFR. This is due to the fact that a larger workflow may have more concurrent module executions on the same node and more concurrent data transfers over the same link, which conforms to the decreasing trend observed in Fig. 8.18.

8.5 Performance Evaluation of SDEDS

We conduct an extensive set of simulations and experiments to evaluate the correctness and accuracy of SDEDS using a large number of workflows with different structures (linear pipelines and DAG-structured workflows) and computer networks with various sizes from small to large scales.

For pipeline mapping, we consider two objectives, i.e., MED and MFR, and three mapping constraints, i.e., NNR, CNR and ANR. For MED, we theoretically calculate the exact ED of a pipeline using Eq. 3.2.1 for a given mapping scheme since there is no resource

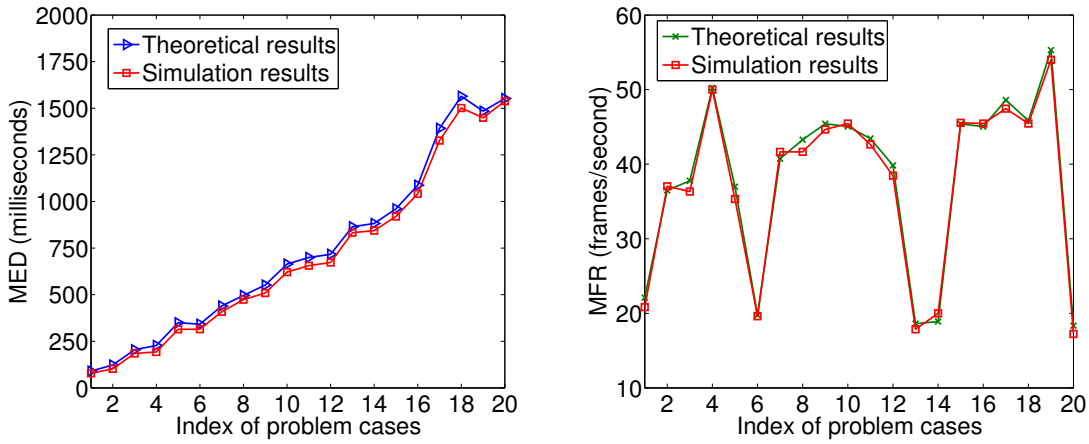
sharing; while for MFR, the theoretical calculation using Eq. 3.2.2 provides an approximate data rate due to the resource sharing of multiple concurrent modules mapped on the same node processing different datasets that are continuously fed into the system.

For DAG-structured workflow mapping, we provide both approximate and exact ED calculation based on theoretical analysis with a unitary-input dataset, and compare the theoretically calculated MFR and the SDEDS simulator for workflows with sequential-input datasets. We also conduct workflow experiments in real networks for comparison-based performance evaluation.

Performance Evaluation of Pipeline Mappings

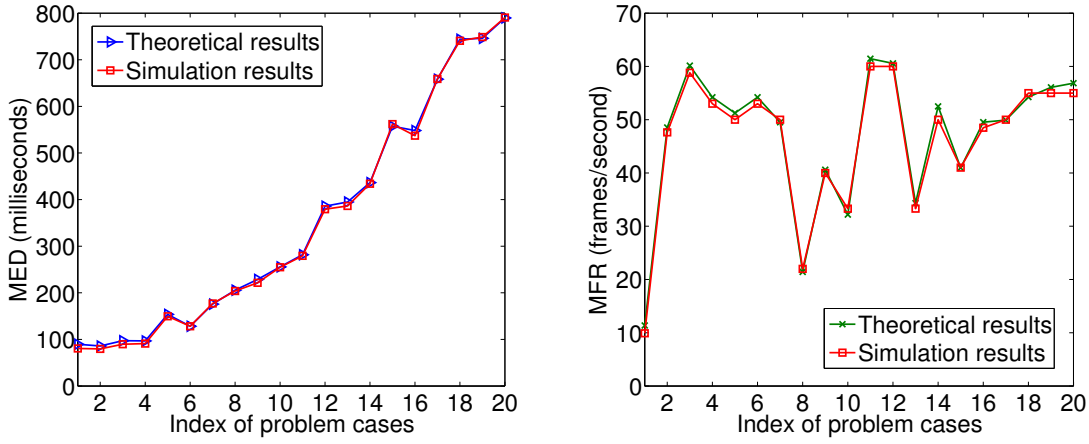
For each mapping objective and constraint, we run 20 pipeline simulation experiments of different problem sizes indexed from 1 to 20. In each experiment, we collect MED or MFR performance measurements and compare them with those theoretical calculations based on our analytical cost models and objectives. The MED performance comparisons for pipeline mapping are plotted in Figs. 8.19(a), 8.20(a), and 8.21(a), and the MFR measurements are plotted in Figs. 8.19(b), 8.20(b), and 8.21(b), under three different mapping constraints. The x axis represents the index of problem cases ranging from small scales with several modules and nodes to large ones with hundreds of modules and nodes, and the y axis represents the theoretical calculations and simulation results in terms of MED or MFR.

We observe that the performance curves from the simulations match very well the theoretical calculations for pipeline mapping, which strongly indicates the validity of our analytical cost models, the accuracy of our objective functions, and the correctness of our SDEDS implementation. The slight discrepancy between the simulation and theoretical results is mostly caused by the host system dynamics including the context switch overhead between multiple threads and limited resolution of the system sleep time.



(a) (b)

Figure 8.19: MED and MFR performance comparison between theoretical calculations and simulation results for pipeline mapping with NNR.



(a) (b)

Figure 8.20: MED and MFR performance comparison between theoretical calculations and simulation results for pipeline mapping with CNR.

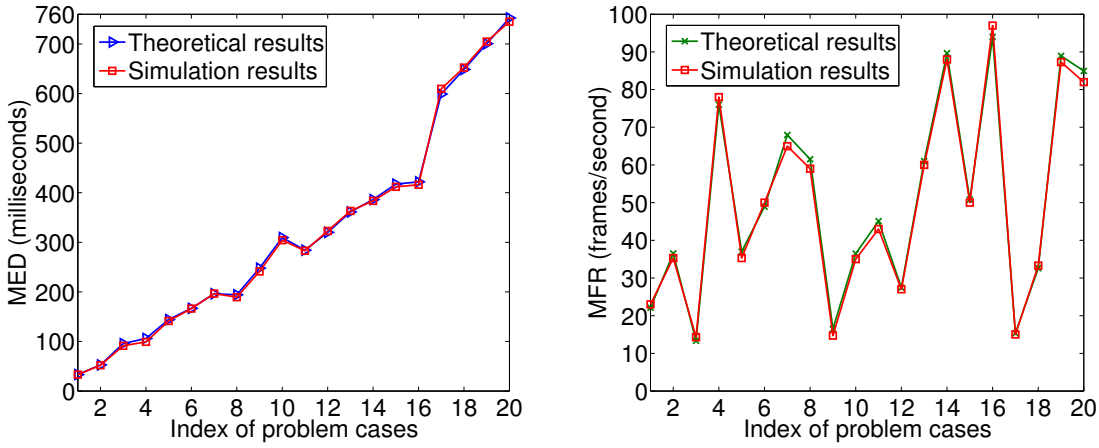


Figure 8.21: MED and MFR performance comparison between theoretical calculations and simulation results for pipeline mapping with ANR.

Performance Evaluation of DAG-structured Workflow Mappings

In view of the complexity of DAG-structured workflow mapping, we conduct an in-depth investigation into the performance evaluation of MED and MFR in three aspects, i.e., SDEDS simulation, theoretical calculation, and a real network deployment using SWAMP.

1) *Comparison with Theoretical Calculation*

For MED in unitary processing applications, we randomly generate 20 mapping problems by varying the topology and size of computing workflows and computer networks, indexed from 1 to 20 as shown Table 8.10. We apply RCP to each problem case and calculate the corresponding mapping scheme. We measure the SDEDS-based simulation results (simMED) using the obtained mapping scheme and compare them with two theoretical calculations under same mapping scheme: (i) appMED: an approximate ED estimation used in [138], where the number of shared modules on any node is assumed to remain constant during the entire course of workflow execution, and (ii) extMED: an exact ED calculation proposed in [137] that counts the accurate number of modules concurrently running on the same node.

Table 8.10: MED and MFR measurements between theoretical calculation and simulation program.

Prb Idx	Prb Size m, $\|E_w\|$, n, $\|E_c\|$	MED (*seconds*)			MFR ($frames/second$)	
		extMED	simMED	appMED	appMFR	simMFR
1	4, 6, 6, 29	0.405	0.440	0.405	0.8761	0.9014
2	6, 10, 10, 86	0.806	1.165	1.578	0.8843	0.9343
3	10, 18, 15, 207	1.603	1.781	5.720	0.9037	1.0000
4	13, 24, 20, 376	1.167	1.262	5.772	1.4538	1.5274
5	15, 30, 25, 597	0.967	1.245	6.265	1.4159	1.5274
6	19, 36, 28, 753	1.558	1.801	9.174	1.3227	1.3646
7	22, 44, 31, 927	3.404	4.015	10.682	1.0891	1.1940
8	26, 50, 35, 1180	1.470	1.603	12.280	1.0308	1.1469
9	30, 62, 40, 1558	3.583	5.459	15.797	0.7190	0.8590
10	35, 70, 45, 1963	3.553	3.892	16.705	0.5225	0.6263
11	38, 73, 47, 2153	4.588	5.011	19.773	0.6211	0.6702
12	40, 78, 50, 2428	3.069	3.383	19.459	0.9505	1.0775
13	45, 96, 60, 3520	3.854	4.226	22.470	0.7010	0.7374
14	50, 102, 65, 4155	4.746	5.189	28.257	0.8190	1.0127
15	55, 124, 70, 4820	7.160	8.364	57.643	0.4741	0.6055
16	60, 240, 75, 5540	12.701	14.178	100.763	0.2228	0.2590
17	75, 369, 90, 7990	16.225	17.810	137.845	0.3764	0.4545
18	80, 420, 100, 9896	19.824	24.395	190.369	0.4494	0.4863
19	90, 500, 150, 22346	28.441	30.010	240.589	0.3079	0.3471
20	100, 660, 200, 39790	29.767	30.390	334.483	0.2940	0.3104

We tabulate the MED performance measurements in these 20 problem cases in Table 8.10, and plot the corresponding performance comparison curves in Fig. 8.22(a). We observe that simMED matches extMED very well in all the cases we studied, which again indicates the validity of our cost models, the accuracy of our objective functions, and the correctness of the SDEDS implementation. The SDEDS simulator provides a more accurate representation of the actual number of shared modules or edges on the same node or link during runtime, hence producing more accurate simMED in performance estimation than appMED. We also observe that in the small problems, e.g., cases 1 to 14, the differences between appMED and extMED are not as significant as in those large ones where extMED is only about 10% of appMED because appMED uses the largest number of shared modules, i.e., the total number of independent modules mapped on the same node,

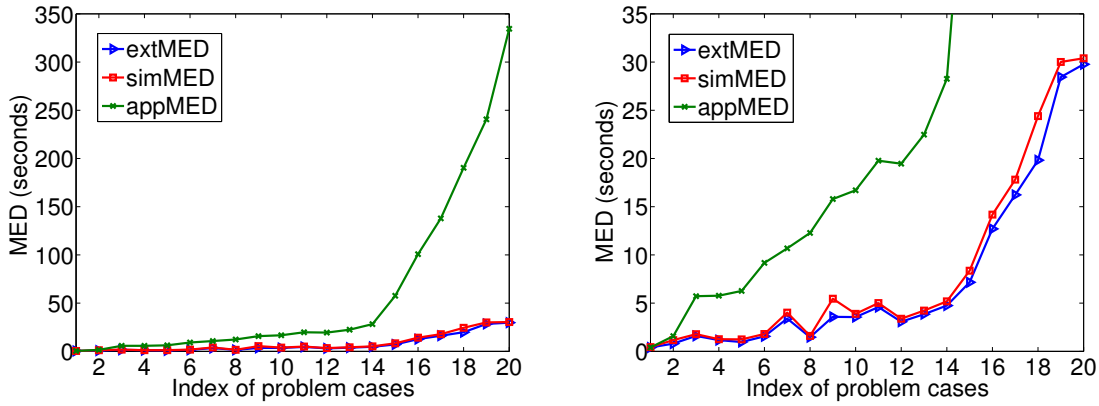


Figure 8.22: Performance comparison among extMED, simMED and appMED results for workflow execution: (a) full range, (b) partial range.

in its calculation without considering sharing dynamics. For a clearer visual comparison, the MED performance curves in Fig. 8.22(a) are re-plotted in Fig. 8.22(b) with a partial performance range.

For MFR workflows, we use the total number of modules on a node to calculate resource sharing since even dependent modules may execute concurrently on different instances of input datasets. We refer to this theoretical calculation of MFR as theMFR and the MFR produced by the SDEDS simulator as simMFR. The MFR performance measurements from theoretical calculations and SDEDS in another set of 20 simulated computing workflows and computer networks, indexed from 1 to 20, are also tabulated in Table 8.10. Note that these workflow mapping instances for MFR have different parameters and topologies from those for MED, but their workflow and network sizes remain the same. For visual comparison, we plot the performance curves of theMFR and simMFR in Fig. 8.23, where we observe that the SDEDS-based simMFR results match very well the theoretical calculations of theMFR, which indicates the validity of our analytical cost models and the correctness of the SDEDS implementation for streaming applications with sequential processing datasets. The slight discrepancy between simMFR and theMFR is due to the fact

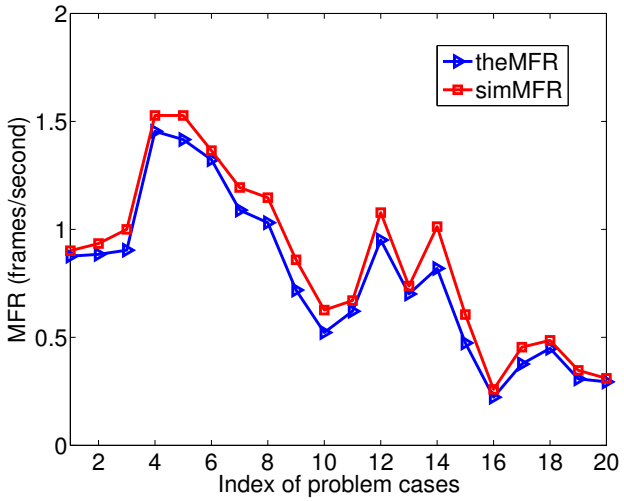


Figure 8.23: MFR performance comparison between simMFR and theMFR for workflow execution.

that the simulator reflects the actual number of concurrent module executions on a node during runtime.

2) *Comparison with Real Network Experiments*

We further conduct workflow experiments for MED using the Condor/DAGMan-based SWAMP system deployed in a local network testbed environment for performance evaluation. Particularly, we consider two different types of network environments for workflow experiments:

- Stable network environments without background traffic and workload: We set up a relatively stable local network testbed environment consisting of 10 workstations located at the University of Memphis. These computers have different hardware configurations in terms of CPU frequency, memory size, and disk space, where the most important hardware parameter, CPU frequency, falls in a range between 1.2 GHz and 3.4 GHz. During the workflow experiments, we only run workflow modules and a minimal set of required kernel-level processes on each computer. We create an arbitrary network topology by configuring a different firewall setting on each computer

and using the Linux traffic control command "*tc*" to allocate a different bandwidth along each overlay link.

- Dynamic network environments with background traffic and workload: We use the same local network testbed environment but with regular user applications and production network traffic. In order to simulate such background traffic and workload, in SDEDS, we use a Poisson distribution to model the arrival rate of other data transfers or user processes, whose duration follows a normal distribution. The parameters of these distributions are empirically determined based on the historical user and system data collected in the real network environment.

We randomly generate a set of 10 workflow instances with different topologies and sizes as shown in Table 8.11. Note that all the workflow experiments are conducted in the same network with 10 workstations as described above but with different network topologies constructed using different firewall settings. The RCP algorithm is executed to find a mapping scheme for MED in each of these test cases. For the workflow experiments without background traffic and workload, we calculate or measure extMED, simMED, expMED and appMED; while for those with background traffic and workload, we measure simMED and expMED for comparison. Please note that the performance comparison for MED without background traffic is the same as the one in Table 8.3 and Fig. 8.7 in Chapter 8.4.1. For the sake of completeness, we include it here as well but present it in a different format.

We tabulate the MED performance comparison in Table 8.11 and plot the corresponding performance curves in Figs. 8.24(a) and 8.24(b). We observe that SDEDS-based simMED matches SWAMP-based expMED very well in stable network environments and achieves a reasonable level of simulation accuracy in dynamic network environments. These measurements show that the SDEDS simulator not only reflects the resource sharing dynamics among multiple concurrent computing modules but also the interactions between computing modules and background processes in shared production networks. We also observe

that the MED measurements in dynamic network environments are in general larger than those in stable network environments. Note that the expMED results are measured in the unit of seconds due to the time resolution in the Condor log files of SWAMP.

Table 8.11: Performance comparison of MED without/with background traffic and workload.

Prb Idx	Workflow Size m, $\|E_w\|$	Without (*seconds*)				With (*seconds*)	
		extMED	simMED	expMED	appMED	simMED	expMED
1	4, 5	187.54	203.91	204	271.55	291.72	314
2	6, 10	298.65	299.03	304	298.65	329.69	467
3	10, 18	384.80	402.96	410	517.71	427.37	547
4	15, 30	698.67	710.72	707	1677.02	1254.14	1302
5	22, 44	1273.26	1290.94	1373	2279.31	1521.24	1646
6	30, 62	1327.83	1342.20	1405	2626.25	1725.25	1965
7	35, 70	1390.05	1484.06	1586	2966.77	1831.49	2050
8	40, 78	1406.47	1490.00	1513	2381.61	1840.86	2071
9	45, 96	1581.13	1663.38	1689	3340.16	1926.30	2051
10	50, 102	1748.04	1801.48	1828	4100.91	2110.55	2458

We conduct a similar set of 10 workflow experiments for MFR in the same testbed network. The Greedy LDP algorithm is executed to compute the mapping scheme in each of these 10 cases. Again, we run SDEDS-based simulations (simMFR) and SWAMP-based experiments (expMFR) in a stable network environment without background traffic and workload, and a dynamic network environment with background traffic and workload. The MFR measurements in these 10 cases are tabulated in Table 8.12 and plotted in Figs. 8.25(a) and 8.25(b) for visual comparison. We observe that simMFR achieves a very high level of simulation accuracy in both stable (compared to both theoretical calculations and real network experiments) and dynamic (compared to real network experiments) network environments. We would like to point out that SDEDS achieves a relatively better simulation accuracy for MFR than for MED in dynamic environments because MFR is only determined by the individual time on the bottleneck node or link while MED is affected by the accumulative time of all the modules along the critical path. Hence, the system overhead and background workload have a more significant impact on MED than MFR.

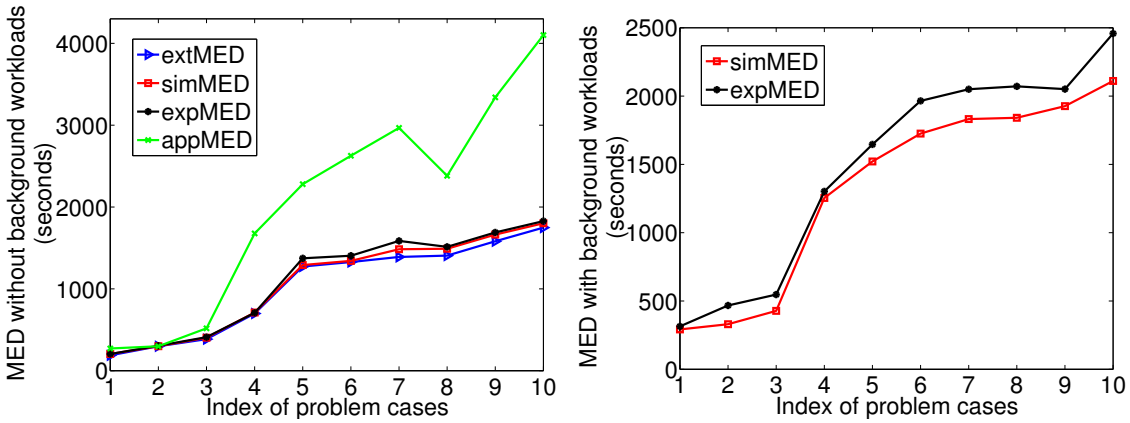


Figure 8.24: Performance comparison among extMED, simMED, expMED and appMED results for workflow execution: (a) in a stable network environment without background traffic and workload, (b) in a dynamic network environment with background traffic and workload.

Table 8.12: Performance comparison of MFR without/with background traffic and workload.

Prb Idx	Workflow Size m, $\|E_w\|$	Without ($frames/second$)			With ($frames/second$)	
		theMFR	simMFR	expMFR	simMFR	expMFR
1	4, 5	2.6372	2.7605	2.6027	2.0370	2.1571
2	6, 10	3.9558	4.1845	3.6803	2.9830	2.7407
3	10, 18	1.4834	1.5070	1.3745	0.9120	0.8237
4	15, 30	2.3735	2.5633	2.2421	1.3270	1.4463
5	22, 44	0.9494	0.9861	0.9377	0.6860	0.6331
6	30, 62	0.8477	0.8734	0.8254	0.6510	0.6340
7	35, 70	0.9494	0.9807	0.8921	0.6280	0.6716
8	40, 78	0.6781	0.7056	0.6451	0.5260	0.4286
9	45, 96	1.1868	1.2988	1.1698	0.8740	0.7964
10	50, 102	0.7912	0.8106	0.7399	0.4280	0.4597

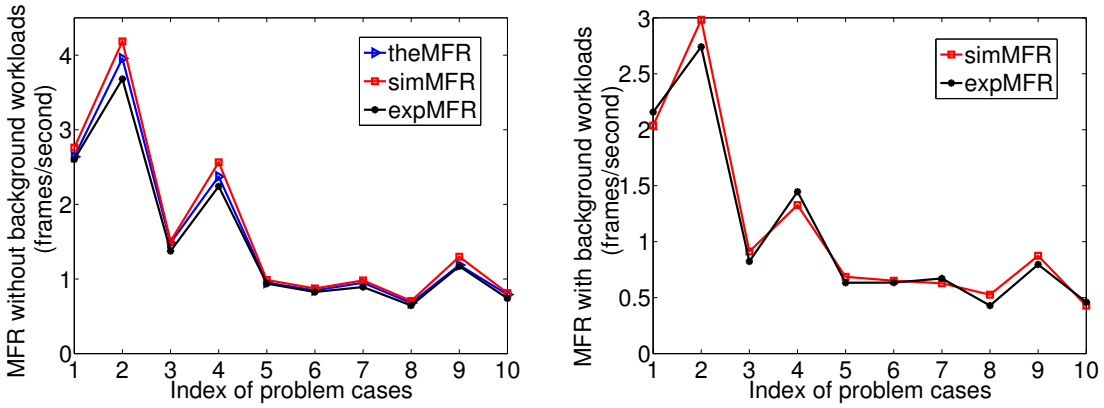


Figure 8.25: Performance comparison among theMFR, simMFR and expMFR results for workflow execution: (a) in a stable network environment without background traffic and workload, (b) in a dynamic network environment with background traffic and workload.

8.6 Performance Evaluation for SWAMP

To further investigate the superiority of the proposed mapping solutions, we conduct comprehensive experiments on two real-life workflow applications through SWAMP in real network environments.

8.6.1 Performance Comparison Using Spallation Neutron Source (SNS) Workflow

The data analysis process in SNS experiments includes two main computing modules, i.e., data rotation and data rebinning. The SNS experimental dataset used in this study contains 160 energy slices, each of which contains 61 runs of reduced data as shown in Fig. 7.6. For each energy slice, we create a separate workflow that rotates the data of 61 runs in parallel and then performs a concatenation operation on the rotated data.

To obtain an accurate performance estimation, we run each SNS module with a certain input data size z on every computer node and measure its execution time t. The time cost coefficient λ per data unit for the module on that particular node is calculated as

$\frac{t}{z}$, which is stored in a 2D table with rows representing modules and columns representing computer nodes. Each module execution is repeated 10 times on every node and the average execution time is used to compute λ. Since there are 61 input files (runs) available in each energy slice, we choose a different number of input files in each experiment to control the input data size. Table 8.13 lists the number of input files and its corresponding data sizes used in 10 SNS workflow experiments.

Table 8.13: Input data sizes in 10 SNS experiments.

Idx of prb sizes	1	2	3	4	5	6	7	8	9	10
Number of runs	6	12	18	24	30	36	42	48	54	60
Data size (MB)	9	18	27	36	45	54	63	72	81	90

To investigate the MED performance in unitary processing applications, we run a single component SNS workflow with 10 different input data sizes shown in Table 8.13 and measure its MED using the proposed impRCP algorithm and the Condor/DAGMan's default mapping scheme. Since the default Condor/DAGMan uses a centralized data forwarding mechanism, we adopt Stork in SWAMP to implement a distributed execution model for impRCP. The MED measurements plotted in Fig. 8.26 show that impRCP consistently achieves a smaller MED than default Condor/DAGMan. The differences between these two methods are not significant for small input data sizes because Stork itself runs as an individual job and introduces extra overhead during file transfer, but the advantage of impRCP becomes more obvious for large input data sizes because impRCP optimizes the entire workflow mapping and avoids redundant file transfers between execution and submission nodes in Condor/DAGMan.

Since Condor/DAGMan is not inherently capable of supporting streaming processing of continuous input datasets, for MFR experiments, we link the 160 component SNS workflows together (160 energy slices) to form a virtual workflow as shown in Fig. 8.27, where

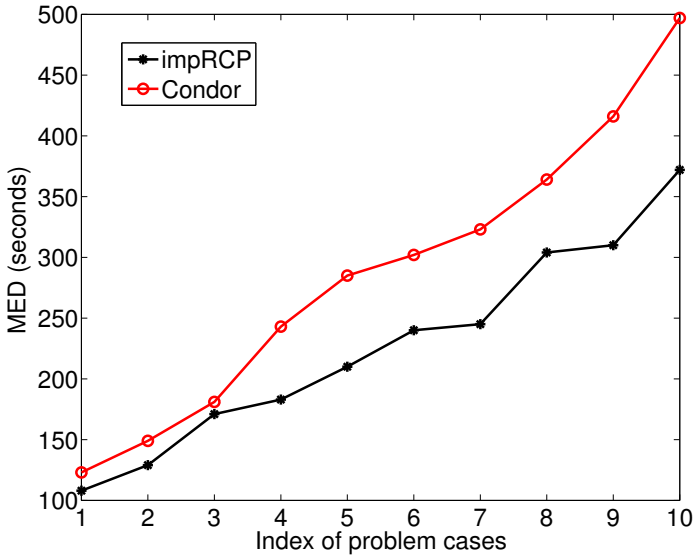


Figure 8.26: MED comparison between impRCP and the Condor default mapping.

each module in the previous component workflow is connected to its corresponding module in the succeeding component workflow using a virtual control flow edge denoted by a dashed arrow. When the current module finishes executing the current dataset, it sends the output along the horizontal data flow edge (a solid arrow) to its succeeding module in the same component workflow, and also sends a signal through the vertical control flow edge to the corresponding module in the succeeding component workflow. When the corresponding module receives the signal, it starts executing the next dataset immediately if all of its input data arrive, and the same is applied to the rest of the modules in the workflows.

We run SNS streaming experiments using the same data sizes in Table 8.13 to compare the MFR measurements of the proposed Greedy LDP algorithm with those of the other workflow mapping algorithms. The performance curves produced by Greedy LDP, Greedy A^* and Greedy are plotted in Fig. 8.28. There is no performance curve for Streamline because it cannot find a feasible mapping scheme in some cases due to its failure to consider the constraints on module dependency and network connectivity. We observe that Greedy LDP outperforms the other two algorithms in comparison in all the cases we studied. Since

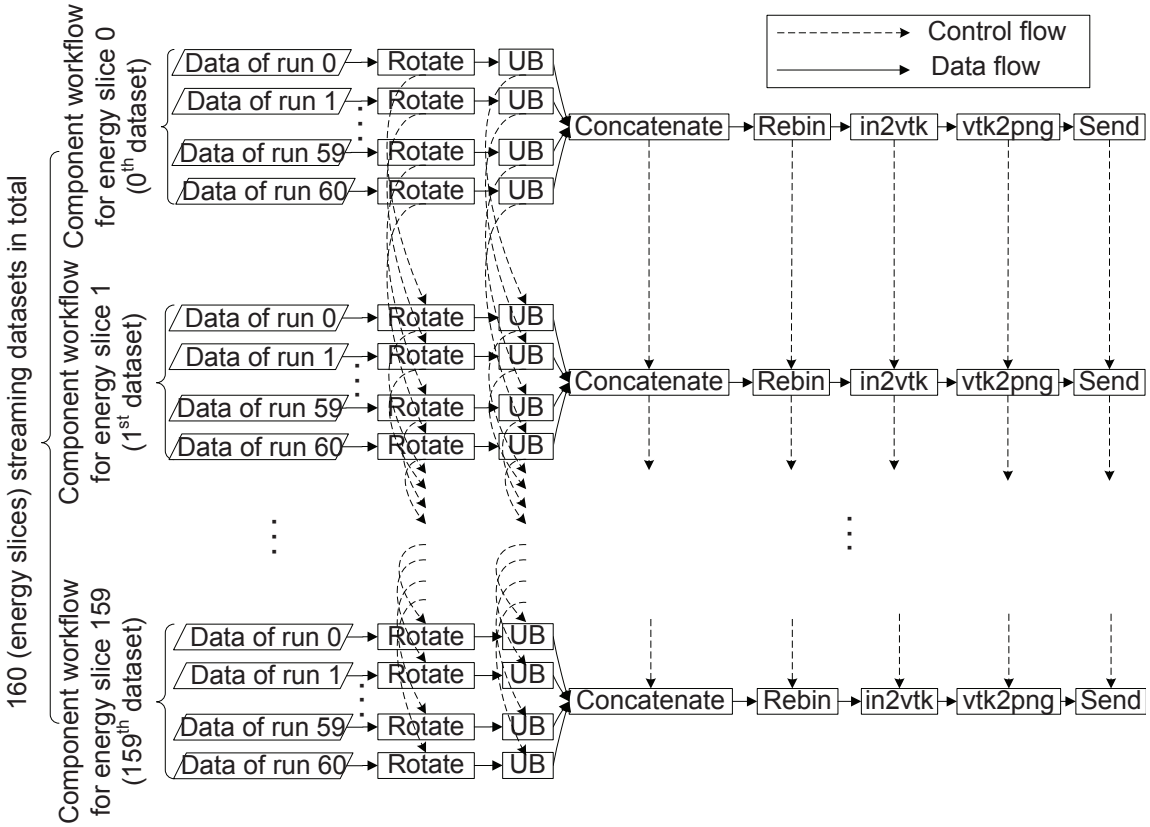


Figure 8.27: Virtual SNS workflow structure for streaming processing.

we use the same workflow structure and mapping scheme for different input data sizes (which is different from the simulation settings), a larger data size obviously results in a longer MED and a smaller MFR, which explains the increasing and decreasing trends in Figs. 8.26 and 8.28, respectively.

8.6.2 Performance Comparison Using Climate Modeling (CM) Workflow

We apply impRCP to a large-scale scientific application on Climate Modeling (CM) at Brookhaven National Laboratory, whose goal is to facilitate the development and evaluation of physical parameterizations for the NCAR Single column Community Atmosphere

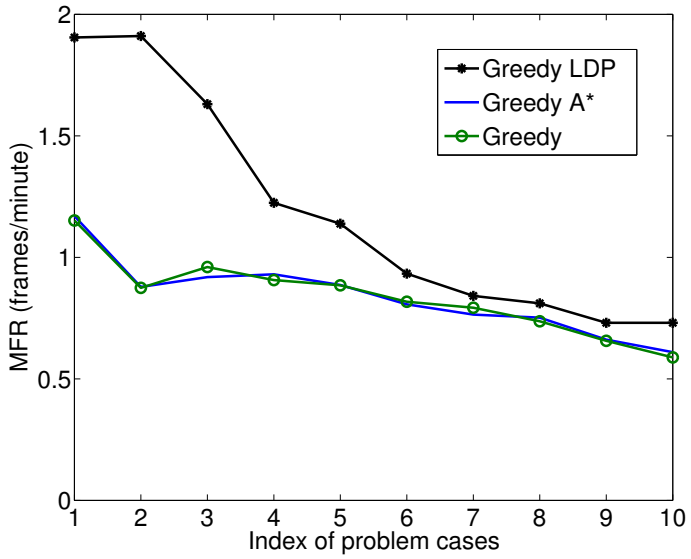


Figure 8.28: MFR comparison among Greedy LDP, Greedy A^* and Greedy.

Table 8.14: Input data sizes in 10 SCAM experiments.

Problem Index	1	2	3	4	5	6	7	8	9	10
No of days	2	4	6	8	10	12	14	16	18	20

Model (SCAM) [5]. We represent SCAM as a workflow and execute it using SWAMP in grid environments with GridFTP-enabled inter-module data transfer. The SCAM workflows use the Intensive Observing Period (IOP) data that provide transient forcing information to SCAM physics. Since there are 20 days of IOP data available in March 2000, we choose a different number of days in each experiment to control the input data size. Table 8.14 lists the number of days used in 10 SCAM workflow experiments.

We run SCAM experiments using the data sizes in Table 8.14 to compare the MED measurements of the proposed impRCP algorithm with the *Random* and *Round Robin* workflow mapping algorithms. The Random algorithm assigns each SCAM module to one of the 12 PC workstations randomly while the Round Robin algorithm assigns these SCAM modules in a round-robin manner. The performance curves produced by impRCP, Random and

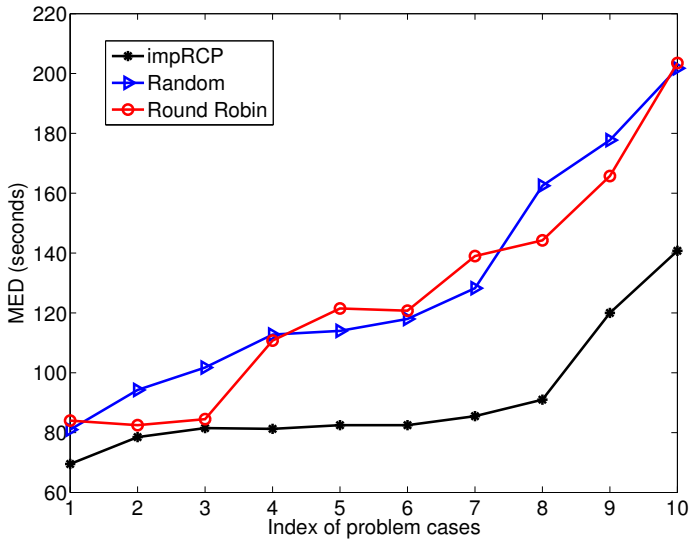


Figure 8.29: MED comparison between impRCP, Random, and Round Robin.

Round Robin are plotted in Fig. 8.29. We observe that impRCP outperforms the other two algorithms in all 10 SCAM workflow experiments. Since there is only one climate modeling workflow structure, the same mapping scheme is produced for different input data sizes (which is different from the simulation setting) in the same network environment. A larger data size is generally more likely to have a longer execution time, resulting in a larger delay, which explains the overall increasing trend in each curve. A few irregulars might have been caused by system and network dynamics at the time of experiments. Here, we also adopt Stork in SWAMP to implement a distributed execution model for impRCP.

Since Condor is not inherently capable of supporting streaming processing of multiple continuous datasets, for MFR experiments, we link 100 SCAM workflows together to form a virtual workflow as shown in Fig. 8.30, where each module in the previous component workflow is connected to its corresponding module in the succeeding component workflow with a virtual control flow edge denoted by a dashed arrow. When the current module finishes executing the current dataset, it sends the output along the horizontal data flow edge (a solid arrow) to its succeeding module in the same workflow, and also sends a signal through

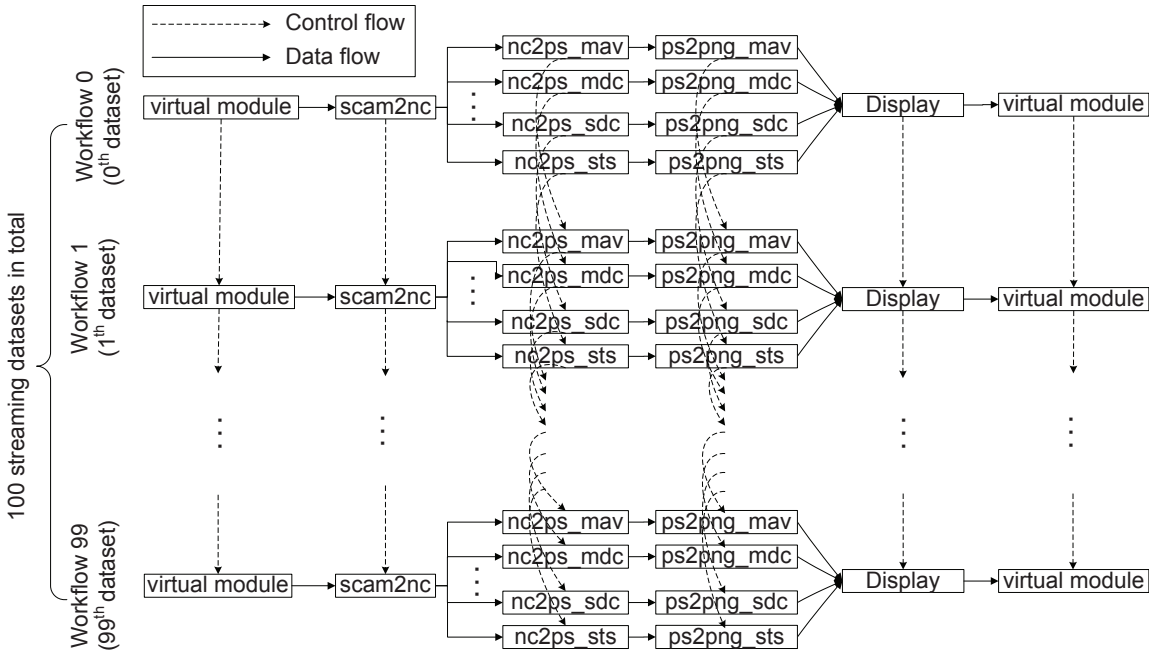


Figure 8.30: The virtual SCAM workflow structure for streaming processing.

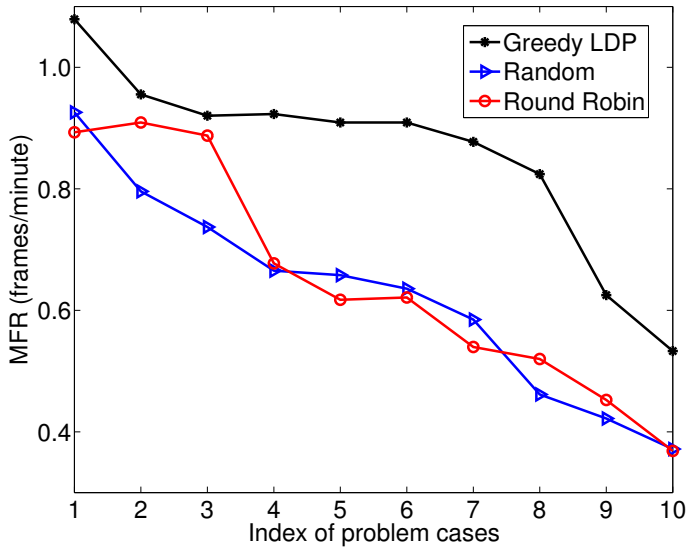


Figure 8.31: MFR comparison between Greedy LDP, Random, and Round Robin.

the vertical control flow edge to the corresponding module in the succeeding component workflow. When the corresponding module receives the signal, it starts executing the next dataset immediately, and the same is applied to the rest of the modules in the workflows.

We run SCAM streaming experiments using the data sizes in Table 8.14 to compare the MFR measurements of the proposed Greedy LDP algorithm with the Random and Round Robin workflow mapping algorithms, whose performance curves are plotted in Fig. 8.31. We observe that Greedy LDP also outperforms the other two algorithms in all the experiments.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

The objective of our work in this dissertation is to conduct a thorough investigation into the optimization of the end-to-end performance of large-scale distributed scientific workflows in heterogeneous network environments from both theoretical and practical perspectives.

We constructed cost models of computing workflows and computer networks to quantitate the characteristics of workflows modeled as simple as linear pipelines or as complex as DAG-structured workflows. We formulated a comprehensive set of workflow mapping problems and provided rigorous complexity analysis as well as NP-completeness and non-approximability proofs. We proposed and implemented optimal and heuristic solutions to optimize the end-to-end performance for fast system response in unitary processing applications and smooth data flow in sequential processing applications. We considered fault tolerance and adapted the proposed solutions to optimize workflow performance in unreliable networks. We designed and implemented a simulation program, SDEDS, to simulate dynamic execution of workflows before real network deployment. We also developed and deployed a workflow automation and management system, SWAMP, to help users construct and execute workflows and monitor their executions. The superiority of the proposed workflow mapping solutions is demonstrated by an extensive set of performance comparisons with existing workflow mapping algorithms through both simulations and experiments.

9.2 Future Work

Workflow mapping/scheduling problems have been the focus of research for years, and have attracted more and more attention from researchers as the scale and data volume of workflows rapidly increase. The research presented in this dissertation focuses on the optimization of large-scale scientific workflows with intricate inter-module dependencies and resource sharing dynamics.

In the current cost models, we used a normalized quantity to represent the processing power and bandwidth for simplicity. However, a single constant is not always sufficient to describe node computing and link transfer capabilities, which highly depend on the type and availability of system and network resources, and could be time-varying in a dynamic environment. It is of our future interest to investigate sophisticated cost models to characterize real-time node and link behaviors in dynamic network environments.

An accurate calculation of resource sharing dynamics is critical to workflow optimization. Besides fair share, we will conduct an analysis of on-node scheduling where the execution order among concurrent modules needs to be determined during execution. We will explore and apply other resource sharing strategies such as proportional and priority-based ones to further improve the workflow performance.

In the current RCP algorithm, the CP mapping is done separately without considering resource sharing, which may cause the mapping path to more likely fall on nodes with more resources. The quality of the branch modules mapping scheme also affects the overall optimization performance.

As for fault tolerance, besides minimizing the failure probability by providing a strategically computed workflow mapping scheme, we would also like to consider some remedy or recovery mechanisms to sustain the workflow execution process in case of failures.

We will further improve the SWAMP system in terms of efficiency, functionality, stability, and user-friendliness, and work with more science groups to conduct more real-life experiments.

Bibliography

[1] Climate and Carbon Research Institute. http://www.ccs.ornl.gov/CCR.

[2] Spallation Neutron Source. http://neutrons.ornl.gov, http://www.sns.gov.

[3] Relativistic Heavy Ion Collider. http://www.bnl.gov/rhic.

[4] NSF Grand Challenges in eScience Workshop, 2001. http://www2.evl.uic.edu/NSF/index.html.

[5] Atmospheric Sciences Division, Brookhaven National Laboratory. http://www.ecd.bnl.gov.

[6] Worldwide LHC Computing Grid (WLCG). http://lcg.web.cern.ch/LCG.

[7] Earth System Grid (ESG). http://www.earthsystemgrid.org.

[8] Open Science Grid. http://www.opensciencegrid.org.

[9] Distributed computing projects. http://en.wikipedia.org/wiki/List_of_distributed_computing_projects.

[10] Condor. http://www.cs.wisc.edu/condor.

[11] Kepler. https://kepler-project.org.

[12] DAGMan. http://www.cs.wisc.edu/condor/dagman.

[13] Gridbus Workflow Engine (GWFE). http://www.buyya.com/gridbus/workflow/.

[14] One-Way Active Measurement Protocol. http://www.internet2.edu/performance/owamp/.

[15] Bandwidth Test Controller. http://www.internet2.edu/performance/bwctl/.

[16] Monitoring and Discovery System (MDS). http://www.globus.org/toolkit/mds/.

[17] OSG Resource and Site Validation. http://vdt.cs.wisc.edu/components/osg-rsv.html.

[18] GridFTP. http://www.globus.org/grid_software/data/gridftp.php.

[19] Reliable File Transfer. http://www-unix.globus.org/toolkit/docs/3.2/rft/index.html.

[20] Storage Resource Broker (SRB). http://www.sdsc.edu/srb/index.php/Main_Page.

[21] Storage Resource Management (SRM). https://sdm.lbl.gov/srm-wg/.

[22] OSCARS: On-demand Secure Circuits and Advance Reservation System. http://www.es.net/oscars.

[23] Visualization Toolkit. http://www.vtk.org.

[24] VisIt. https://wci.llnl.gov/codes/visit.

[25] F.N. Afrati, C.H. Papadimitriou, and G. Papageorgiou. Scheduling DAGs to minimize time and communication. In *Proc. of the 3rd Aegean Workshop on Computing: VLSI Algorithms and Architectures*, pages 134–138. Springer-Verlag, 1988.

[26] B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramachandran. Streamline: a scheduling heuristic for streaming application on the grid. In *Proc. of the 13th Multimedia Comp. and Net. Conf.*, San Jose, CA, 2006.

[27] I. Ahmed and Y. Kwok. On exploiting task duplication in parallel program scheduling. *IEEE Trans. on Para. and Dist. Sys.*, 9:872–892, 1998.

[28] T. Angskun, G. Fagg, G. Bosilca, and J. Dongarra. Scalable fault tolerant protocol for parallel runtime environments. In *Proc. of Euro PVM/MPI*, Bonn, Germany, 2006.

[29] S.W. Annie, H. Yu, S. Jin, and K.-C. Lin. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Trans. on Para. and Dist. Sys.*, 15:824–834, 2004.

[30] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer Verlag, 1999.

[31] R. Bajaj and D.P. Agrawal. Improving scheduling of tasks in a heterogeneous environment. *IEEE Trans. on Parallel and Distributed Systems*, 15:107–118, 2004.

[32] S. Bajaj, L. Breslau, D. Estrin, K. Fall, S. Floyd, P. Haldar, M. Handley, A. Helmy, J. Heidemann, P. Huang, S. Kumar, S. Mccanne, R. Rejaie, P. Sharma, K. Varadhan, Y. Xu, H. Yu, and D. Zappala. Improving simulation for network research. Technical report, University of Southern California, 1999.

[33] A.F. Bashir, V. Susarla, and K. Vairavan. A statistical study of the performance of a task scheduling algorithm. *IEEE Trans. on Computers*, 32(12):774–777, Dec. 1975.

[34] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, and F. Zini. OptorSim - a grid simulator for studying dynamic data replication strategies. *Int. J. of High Performance Computing Applications*, 17(4), 2003.

[35] A. Benoit, M. Hakem, and Y. Robert. Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. In *Proc. of IEEE Int. Symp. on Para. and Dist. Proc.*, pages 1–8, Miami, FL, April 2008.

[36] A. Benoit, M. Hakem, and Y. Robert. Optimizing the latency of streaming applications under throughput and reliability constraints. In *Proc. of the 2009 Int. Conf. on Para. Proc.*, pages 325–332, 2009.

[37] A. Benoit and Y. Robert. Mapping pipeline skeletons onto heterogeneous platforms. *JPDC*, 68(6):790–808, 2008.

[38] C. Boeres, J.V. Filho, and V.E.F. Rebello. A cluster-based strategy for scheduling task on heterogeneous processors. In *Proc. of 16th Symp. on Computer Architecture and High Performance Computing*, pages 214–221, 2004.

[39] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello. MPICH-V: a multiprotocol fault tolerant MPI. *J. of High Perf. Comp. and App.*, 2005.

[40] D. Bozdag, U. Catalyurek, and F. Ozguner. A task duplication based bottom-up scheduling algorithm for heterogeneous environments. In *Proc. of the 20th Int. Parallel and Distributed Processing Symposium*, Apr. 2006.

[41] T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, 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. *JPDC*, 61(6):810–837, June 2001.

[42] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid. In *Proc. of the 4th Int. Conf./Exhibition on the High Performance Computing in the Asia-Pacific Region*, volume 1, pages 283–289, 2000.

[43] R. Buyya and M. Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, pages 1175–1220, 2002.

[44] J. Cao, S.A. Jarvis, S. Saini, and G.R. Nudd. GridFlow: workflow management for grid computing. In *Proc. of the 3rd IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, pages 198–205, May 2003.

[45] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a generic framework for large-scale distributed experiments. In *Proc. of the 10th IEEE Int. Conf. on Computer Modeling and Simulation - EUROSIM/UKSIM*, pages 126–131, Cambridge, UK, Apr. 2008.

[46] S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: end-to-end analysis of heterogeneous, multi-resource real-time systems. In *Proc. of the 15th Int. Conf. on Distributed Computing Systems*, pages 204–211, May 1995.

[47] V. Chaudhary and J.K. Aggarwal. A generalized scheme for mapping parallel algorithms. *IEEE Trans. on Parallel and Distributed Systems*, 4(3):328–346, May 1993.

[48] L. Chen and G. Agrawal. Resource allocation in a middleware for streaming data. In *Proc. of the 2nd Workshop on Middleware for Grid Comp.*, Toronto, Canada, Oct. 2004.

[49] S.Y. Choi, J. Turner, and T. Wolf. Configuring sessions in programmable networks. In *Proc. of IEEE INFOCOM*, pages 60–66, 2001.

[50] S.Y. Choi, J. Turner, and T. Wolf. Configuring sessions in programmable networks. *Int. J. of Computer and Telecommunications Networking*, 41:269–284, 2003.

[51] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with triana services. *Concurrency and Computation: Practice and Experience, Special Issue: Workflow in Grid Systems*, 18(10):1021–1037, 2006. http://www.trianacode.org.

[52] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *Proc. of the 3rd IAPR-TC-15 Int. Workshop on Graph-based Representations*, Italy, 2001.

[53] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger. *Workflows for e-Science*, chapter Workflow in Condor. Springer Press, 2007.

[54] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny. Pegasus: mapping scientific workflows onto the grid. In *Proc. of the European Across Grids Conference*, pages 11–20, 2004.

[55] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T.H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, and L. Zhao. Managing large-scale workflow execution from resource provisioning to provenance tracking: the cybershake example. In *Proc. of the e-Science Conf.*, Amsterdam, Netherlands, Dec. 2006.

[56] E. Deelman, D.B. Gannon, M. Shields, and I.J. Taylor. Workflows and e-Science: an overview of workflow system features and capabilities. *J. of Future Generation Computer Systems*, 25(5):528–540, May 2009.

[57] E. Deelman, G. Singh, M. Su, J. Blythe, A Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13:219–237, 2005.

[58] M.K. Dhodhi, I. Ahmad, and A. Yatama. An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. *J. Parallel and Distributed Computing*, 62:1338–1361, 2002.

[59] T. Do, D. Nguyen, H. Nguyen, T. Nguyen, and W. Shi. Failure-aware scheduling in grid computing environments. In *Proc. of the 2008 Int. Conf. on Grid Comp. and App.*, Las Vegas, Nevada, July 2009.

[60] The office of science data-management challenge, Mar.-May 2004. Report from the DOE Office of Science Data-Management Workshops. Technical Report SLAC-R-782, Stanford Linear Accelerator Center.

[61] A. Dogan and F. Özgüner. Optimal and suboptimal reliable scheduling of precedence-constrained tasks in heterogeneous distributed computing. In *Proc. of 2000 Int. Workshops on Para. Proc.*, pages 429–436, Toronto, Ontario, Canada, 2000.

[62] A. Dogan and F. Özgüner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. on Para. and Dist. Sys.*, 13(3):308–323, 2002.

[63] J.J. Dongarra, E. Jeannot, E. Saule, and Z. Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Proc. of the 19th Annual ACM Symp. on Para. Alg. and Arch.*, pages 280–288, San Diego, CA, 2007.

[64] A. Dupuy, J. Schwartz, Y. Yemini, and D. Bacon. NEST: a network simulation and prototyping testbed. *ACM Communication*, 33(10):63–74, 1990.

[65] M.F. Ercan and C. Oğuz. Performance of local search heuristics on scheduling a class of pipelined multiprocessor tasks. *J. of Computers and Electrical Engineering*, 31(8):537–555, 2005.

[66] P. Foggia, C. Sansone, and M. Vento. A performance comparison of five algorithms for graph isomorphism. In *Proc. of 3rd IAPR-TC-15 Int. Workshop Graph-based Representations in Pattern Recognition*, pages 188–199, 2001.

[67] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[68] I. Foster, J. Vockler, M. Wilde, and Y. Zhao. Chimera: a virtual data system for representing, querying, and automating data derivation. In *Proc. of the 14th Int. Conf. on Scientific and Statistical Database Management*, pages 37–46, 2002.

[69] I.T. Foster. Globus toolkit version 4: software for service-oriented systems. *J. of Comp. Sci. and Tech.*, pages 513–520, July 2006.

[70] I.T. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *Int. J. of Supercomputer Applications*, 15(3), 2001.

[71] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[72] P. Garcia, C. Pairot, R. Mondejar, J. Pujol, H. Tejedor, and R. Rallo. PlanetSim: A new overlay network simulation framework. *Software Engineering and Middleware*, pages 123–136, 2005.

[73] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, San Francisco, 1979.

[74] B. Gedik, H. Andrade, K.-L. Wu, P.S. Yu, and M. Doo. Spade: The system S declarative stream processing engine. In *Proc. of the 2008 ACM SIGMOD Int. Conf. on Management of Data*, pages 1123–1134, Vancouver, Canada, 2008.

[75] A. Gerasoulis and T. Yang. A comparison of clustering heuristics for scheduling DAGs on multiprocessors. *JPDC*, 16(4):276–291, Dec. 1992.

[76] A. Girault, *É*. Saule, and D. Trystram. Reliability versus performance for critical applications. *JPDC*, 69(3):326–336, 2009.

[77] O. Gonzalez, W.J. Davis, F.G. Gonzalez, and W.J. Davis. A new simulation tool for the modeling and control of distributed systems. *Society for Computer Simulation International*, 78(9):552–567, 2002.

[78] Y. Gu and Q. Wu. Maximizing workflow throughput for streaming applications in distributed environments. In *Proc. of the 19th Int. Conf. on Comp. Comm. and Net.*, Zurich, Switzerland, August 2010.

[79] Y. Gu and Q. Wu. Optimizing distributed computing workflows in heterogeneous network environments. In *Proc. of the 11th Int. Conf. on Distributed Computing and Networking*, Kolkata, India, Jan. 3-6 2010.

[80] Y. Gu, Q. Wu, A. Benoit, and Y. Robert. Brief announcement: Complexity analysis and algorithm design for optimal pipeline configuration in distributed network environments. In *Proc. of the 28th Annual ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (PODC)*, Calgary, Canada, Aug. 10-12 2009.

[81] Y. Gu, Q. Wu, A. Benoit, and Y. Robert. Optimizing end-to-end performance of distributed applications with linear computing pipelines. In *Proc. of the 15th Int. Conf. on Para. and Dist. Sys.*, Shenzhen, China, Dec. 8-11 2009.

[82] Y. Gu, Q. Wu, X. Liu, and D. Yu. Improving throughput and reliability of distributed scientific workflows for streaming data processing. In *Proc. of the 13th Int. Conf. on High Performance Computing and Communications*, Banff, Canada, Sep. 2-4 2011.

[83] Y. Gu, Q. Wu, and N.S.V. Rao. Analyzing execution dynamics of scientific workflows for latency minimization in resource sharing environments. In *Proc. of the 7th IEEE World Congress on Services*, Washington DC, Jul. 4-9 2011.

[84] Y. Gu, Q. Wu, M. Zhu, and N.S.V. Rao. Complexity analysis of pipeline mapping problems in distributed heterogeneous networks. *International Journal of Distributed Sensor Networks*, 5(1), 2009.

[85] F. Guirado, A. Ripoll, C. Roig, and E. Luque. Optimizing latency under throughput requirements for streaming applications on cluster execution. *J. of Cluster Computing*, pages 1–10, 2005.

[86] K. Hashimito, T. Tsuchiya, and T. Kikuno. Effective scheduling of duplicated tasks for fault-tolerance in multiprocessor systems. *IEICE Tran. on Info. and Sys.*, 85(3):525–534, 2002.

[87] K. Hatzis, C. Papanikolaou, G. Pentaris, P. Spirakis, and B. Tampakas. DSS: A tool for simulation of distributed systems and protocols through communicating finite state machines. Technical report, CTI Internal Report, 1994.

[88] J. Hopcroft and J. Wong. Linear time algorithm for isomorphism of planar graphs. In *Proc. of the 6th Annual ACM Symp., Theory of Computing*, pages 172–184, 1974.

[89] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006. http://www.taverna.org.uk.

[90] E. Ilavarasan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *J. of Computer Sciences*, 3(2):94–103, 2007.

[91] M. Jelasity, A. Montresor, G.P. Jesi, and S. Voulgaris.

[92] W.E. Johnston. Computational and data grids in large-scale science and engineering. *J. of Future Generation Computer Systems*, 18(8):1085–1100, 2002.

[93] P. Kacsuk, Z. Farkas, G. Sipos, A. Toth, and G. Hermann. Workflow-level parameter study management in multi-Grid environments by the P-GRADE Grid portal. In *Int. Workshop on Grid Computing Enviornments*, 2006.

[94] S. Kartik and C.S.R. Murthy. Improved task-allocation algorithms to maximize reliability of redundant distributed computing systems. *IEEE Trans. Reliability*, 44:575–586, 1995.

[95] S. Kartik and C.S.R. Murthy. Task allocation algorithms for maximizing reliability of distributed computing systems. *IEEE Trans. Computers*, 46(6):719–724, 1997.

[96] R.M. Khandekar, K.W. Hildrum, S. Parekh, D. Rajan, J.L. Wolf, K.-L. Wu, H. Andrade, and B. Gedik. Cola: Optimizing stream processing application via graph partitioning. In *Proc. of the 10th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 308–327, 2009.

[97] T. Kosar and M. Livny. Stork: making data placement a first class citizen in the grid. In *Proc. of 24th IEEE Int. Conf. on Distributed Computing Systems*, Tokyo, Japan, Mar. 2004.

[98] Y. Kwok and I. Ahmad. Dynamic critical-path scheduling: An effective technique for allocating task graph to multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 7(5):506–521, May 1996.

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

[100] D. Kyriazis, K. Tserpes, A. Menychtas, I. Sarantidis, and T. Varvarigou. Service selection and workflow mapping for Grids: an approach exploiting quality-of-service information. *J. of Concurrency and Computation: Practice and Experience*, 21(6), April 2009.

[101] G. Laszewski and M. Hategan. Workflow concepts of the Java CoG Kit. *Journal of Grid Computing*, 3(3-4):239–258, 2005.

[102] C. Lin, S. Lu, X. Fei, D. Pai, and J. Hua. A task abstraction and mapping approach to the shimming problem in scientific workflows. In *Proc. of the IEEE Int. Conf. on Services Computing*, pages 284–291, Bangalore, India, 2009.

[103] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1605, 2006.

[104] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. of Computer System Science*, pages 42–65, 1982.

[105] T. Ma and R. Buyya. Critical-path and priority based algorithms for scheduling workflows with parameter sweep tasks on global grids. In *Proc. of the 17th Int. Symp. on Computer Architecture on HPC*, pages 251–258, 2005.

[106] C. McCreary, A.A. Khan, J.J. Thompson, and M.E. McArdle. A comparison of heuristics for scheduling DAGs on multiprocessors. In *Proc. of the 8th Int. Symp. on Parallel Processing*, pages 446 – 451. IEEE Computer Society, 1994.

[107] C.L. McCreary, M.A. Cleveland, and A.A. Khan. The problem with critical path scheduling algorithms. Technical report, Auburn University, 1996.

[108] W.J. McDermott, D.A. Maluf, Y. Gawdiak, and P.B. Tran. Airport simulations using distributed computational resources. *Journal of Defense Software Engineering*, pages 7–11, June 2003.

[109] S. Meftali, A. Dziri, L. Charest, P. Marquet, and J.L. Dekeyser. SOAP based distributed simulation environment for system-on-chip (SoC) design. Research paper, Universite des Sciences et Technologies de Lille, France, 2003.

[110] B.T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. PhD thesis, Institute of Computer Science and Applied Mathematics, University of Bern, 1996.

[111] A. Mezzacappa. SciDAC 2005: scientific discovery through advanced computing. *J. of Physics: Conf. Series*, 16, 2005.

[112] J.A. Miller, J. Cardoso, and G. Silver. Using simulation to facilitate effective workflow adaptation. In *Proc. of the 35th Annual Simulation Symposium*, page 177, 2002.

[113] T. Oinn, M. Addis, J. Ferris, D. Marvin, T. Carver, M.R. Pocock, and A. Wipat. Taverna: A tool for the composition and enactment of bioinformatics workflows. In *Bioinformatics*, volume 20, pages 3045–3054, 2004.

[114] J.S. Plank and W.R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In *Proc. of the 28th Int. Symp. on Fault-Tolerant Computing*, pages 48–57, 1997.

[115] G. Racherla, S. Killian, L. Fife, M. Lehmann, and R. Parekh. PARSIT: A parallel algorithm reconfiguration simulation tool. In *Proc. of International Conference on High Performance Computing*, Decemeber 1995.

[116] M. Rahman, S. Venugopal, and R. Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *Proc. of the 3rd IEEE Int. Conf. on e-Sci. and Grid Comp.*, pages 35–42, 2007.

[117] A. Ranaweera and D.P. Agrawal. A task duplication based algorithm for heterogeneous systems. In *Proc. of IEEE Int. Parallel and Distributed Processing Symp.*, pages 445–450, 2000.

[118] K. Ranganathan and I. Foster. Decoupling computation and data scheduling in distributed data-intensive applications. In *Proc. of the 11th IEEE Int. Symposium on High Performance Distributed Computing*, pages 352–358, Washington, DC, 2002.

[119] D. Saha, A. Samanta, and S.R. Sarangi. Theoretical framework for eliminating redundancy in workflows. In *Proc. of the IEEE Int. Conf. on Services Computing*, pages 41–48, Bangalore, India, 2009.

[120] D.C. Schmidt and L.E. Druffel. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *J. of the ACM*, 23:433–445, 1976.

[121] A. Sekhar, B.S. Manoj, and C.S.R. Murthy. A state-space search approach for optimizing reliability and cost of execution in distributed sensor networks. In *Proc. of Int. Workshop on Dist. Comp.*, pages 63–74, 2005.

[122] S.M. Shatz, J.P. Wang, and M. Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. Computers*, 41:1156–1168, 1992.

[123] B. Shirazi, M. Wang, and G. Pathak. Analysis and evaluation of heuristic methods for static scheduling. *J. of Parallel and Distributed Computing*, (10):222–232, 1990.

[124] P. Shroff, D.W. Watson, N.S. Flann, and R.F. Freund. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In *Proc. of Heterogeneous Computing Workshop*, pages 98–104, Apr. 1996.

[125] G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Trans. on Para. and Dist. Systems*, 4(2):175–187, 1993.

[126] M.P. Singh and M.A. Vouk. Scientific workflows: scientific computing meets transactional workflows. In *Proc. of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions*, pages 28–34, Univ. Georgia, Athens, GA, 1996.

[127] I.J. Taylor, E. Deelman, D.B. Gannon, and M. Shields (Eds.). *Workflows for e-Science: scientific workflows for grids*. Springer-Verlag, 2007.

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

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

[130] J.R. Ullmann. An algorithm for subgraph isomorphism. *J. of the ACM*, 23:31–42, 1976.

[131] H. Unger, S. Yuen, P. Kropf, and G. Babin. Simulation of communication of nodes in a wide area distributed system. Eurosim Congress, Delft, Netherlands, June 2001.

[132] P. Velho and A. Legrand. Accuracy study and improvement of network simulation in the SimGrid framework. In *Proc. of the 2nd Int. Conf. on Simulation Tools and Techniques*, 2009.

[133] L. Wang, H.J. Siege, V.P. Roychowdhury, and A.A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *J. of Parallel and Distributed Computing*, 47:8–22, 1997.

[134] B. Wassermann, W. Emmerich, B. Butchart, N. Cameron, L. Chen, and J. Patel. *Workflows for e-Science: Scientific Workflows for Grids*, chapter Sedna: a BPEL-based environment for visual scientific workflow modeling, pages 427 – 448. Springer, London, 2007.

[135] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the ASKALON grid environment. *ACM SIGMOD Record*, 34(3):56–62, Sep. 2005.

[136] J.L. Wolf, N. Bansal, K. Hildrum, S. Parekh, D. Rajan, R. Wagle, K.-L. Wu, and L. Fleischer. SODA: An optimizing scheduler for large-scale stream-based distributed computer systems. In *Proc. of the 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 306–325, 2008.

[137] Q. Wu and V.V. Datla. On performance modeling and prediction in support of scientific workflow optimization. In *Proc. of the 7th IEEE World Congress on Services*, Washington DC, Jul. 5-10 2011.

[138] Q. Wu and Y. Gu. Supporting distributed application workflows in heterogeneous computing environments. In *Proc. of the 14th IEEE Int. Conf. on Parallel and Distributed Systems*, pages 3–10, Melbourne, Australia, Dec. 2008.

[139] Q. Wu and Y. Gu. A distributed workflow mapping algorithm for minimum end-to-end delay under fault-tolerance constraint. In *Proc. of the 16th Int. Conf. on Para. and Dist. Sys.*, Shanghai, China, Dec. 8-10 2010.

[140] Q. Wu and Y. Gu. *Computational and data grids: principles, applications and design*, chapter Performance analysis and optimization of linear workflows in heterogeneous network environments. IGI Global, isbn13: 9781613501139 edition, Sep. 2011.

[141] Q. Wu and Y. Gu. Modeling and simulation of distributed computing workflows in heterogeneous network environments. *SIMULATION: Transactions of the Society*

for Modeling and Simulation International, Special Issue: Performance Evaluation, Modeling and Optimization of Computer Systems, Mar. 2011. First published online.

[142] Q. Wu and Y. Gu. Optimizing end-to-end performance of data-intensive computing pipelines in heterogeneous network environments. *J. of Parallel and Distributed Computing*, 71(2):254–265, 2011.

[143] Q. Wu, Y. Gu, L. Bao, W. Jia, H. Dai, and P. Chen. Optimizing distributed execution of WS-BPEL processes in heterogeneous computing environments. In *Proc. of the Quality of Service in Heterogeneous Networks (QSHINE)*, volume 22, page 770C784, 2009.

[144] Q. Wu, Y. Gu, Y. Liao, X. Lu, Y. Lin, and N.S.V. Rao. Latency modeling and minimization for large-scale scientific workflows in distributed network environments. In *the 44th Annual Simulation Symposium (ANSS11), part of the 2011 Spring Simulation Multiconference (SpringSim11)*, Boston, MA, Apr. 4-7 2011.

[145] Q. Wu, Y. Gu, X. Lu, M. Zhu, P. Brown, W. Lin, and Y. Liu. On optimization of scientific workflows to support streaming applications in distributed network environments. In *Proc. of the 5th Workshop on Workflows in Support of Large-Scale Science (in conjunction with SC 2010)*, New Orleans, LA, Nov. 14 2010.

[146] Q. Wu, Y. Gu, and Z. Wang. Simulation-based analysis of performance dynamics of distributed applications in heterogeneous network environments. In *SpringSim'09: Proc. of the 2009 Spring Simulation Multiconference*, pages 1–8, San Diego, CA, March 22-27 2009.

[147] Q. Wu, Y. Gu, M. Zhu, and N.S.V. Rao. Optimizing network performance of computing pipelines in distributed environments. In *Proc. of the 22nd IEEE Int. Parallel and Distributed Processing Symp.*, Miami, Florida, Apr. 14-18 2008.

[148] Q. Wu and N. S. V. Rao. On transport daemons for small collaborative applications over wide-area networks. In *Proc. of the 24th IEEE Int. Performance Computing and Communications Conf.*, pages 159–166, Phoenix, AZ, Apr. 7-9 2005.

[149] Q. Wu, M. Zhu, Y. Gu, and N.S.V. Rao. System design and algorithmic development for computational steering in distributed environments. *IEEE Trans. on Parallel and Distributed Systems*, 21(4):438–451, April 2010.

[150] Q. Wu, M. Zhu, X. Lu, P. Brown, Y. Lin, Y. Gu, F. Cao, and M.A. Reuter. Automation and management of scientific workflows in distributed network environments. In *Proc. of the 6th Int. Workshop on Sys. Man. Tech., Proc., and Serv.*, Atlanta, GA, April 19 2010.

[151] Y. Zhu and B. Li. Overlay network with linear capacity constraints. *IEEE Trans. on Parallel and Distributed Systems*, 19:159–173, Feb. 2008.