

Florida International University FIU Digital Commons

FIU Electronic Theses and Dissertations

University Graduate School


11-14-2014

Techniques for Efficient Execution of Large-Scale Scientific Workflows in Distributed Environments

Selim Kalayci

Florida International University, skalayci@gmail.com

Follow this and additional works at: <http://digitalcommons.fiu.edu/etd>

 Part of the [Computational Engineering Commons](#), [Computer and Systems Architecture Commons](#), [Other Computer Sciences Commons](#), [Software Engineering Commons](#), and the [Systems Architecture Commons](#)

Recommended Citation

Kalayci, Selim, "Techniques for Efficient Execution of Large-Scale Scientific Workflows in Distributed Environments" (2014). *FIU Electronic Theses and Dissertations*. Paper 1664.
<http://digitalcommons.fiu.edu/etd/1664>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

TECHNIQUES FOR EFFICIENT EXECUTION OF LARGE-SCALE SCIENTIFIC
WORKFLOWS IN DISTRIBUTED ENVIRONMENTS

A dissertation submitted in partial fulfillment of

the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Selim Kalayci

2014

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Selim Kalayci, and entitled Techniques for Efficient Execution of Large-Scale Scientific Workflows in Distributed Environments, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Giri Narasimhan

Jason Liu

Jorge Rodriguez

S. Masoud Sadjadi, Major Professor

Date of Defense: November 14, 2014

The dissertation of Selim Kalayci is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2014

ABSTRACT OF THE DISSERTATION
TECHNIQUES FOR EFFICIENT EXECUTION OF LARGE-SCALE SCIENTIFIC
WORKFLOWS IN DISTRIBUTED ENVIRONMENTS

by

Selim Kalayci

Florida International University, 2014

Miami, Florida

Professor S. Masoud Sadjadi, Major Professor

Scientific exploration demands heavy usage of computational resources for large-scale and deep analysis in many different fields. The complexity or the sheer scale of the computational studies can sometimes be encapsulated in the form of a workflow that is made up of numerous dependent components. Due to its decomposable and parallelizable nature, different components of a scientific workflow may be mapped over a distributed resource infrastructure to reduce time to results. However, the resource infrastructure may be heterogeneous, dynamic, and under diverse administrative control. Workflow management tools are utilized to help manage and deal with various aspects in the lifecycle of such complex applications. One particular and fundamental aspect that has to be dealt with as smooth and efficient as possible is the run-time coordination of workflow activities (i.e. workflow orchestration). Our efforts in this study are focused on improving the workflow orchestration process in such dynamic and distributed resource environments. We tackle three main aspects of this process and provide contributions in each of them. Our first contribution involves increasing the scalability and site autonomy in situations where the mapped components of a workflow span across several

heterogeneous administrative domains. We devise and implement a generic decentralization framework for orchestration of workflows under such conditions. Our second contribution is involved with addressing the issues that arise due to the dynamic nature of such environments. We provide generic adaptation mechanisms that are highly transparent and also substantially less intrusive with respect to the rest of the workflow in execution. Our third contribution is to improve the efficiency of orchestration of large-scale parameter-sweep workflows. By exploiting their specific characteristics, we provide generic optimization patterns that are applicable to most instances of such workflows. We also discuss implementation issues and details that arise as we provide our contributions in each situation.

TABLE OF CONTENTS

CHAPTER	PAGE
CHAPTER 1	1
INTRODUCTION	1
1.1 DECENTRALIZATION OF THE WORKFLOW ORCHESTRATION.....	3
1.2 RUN-TIME ADAPTATION.....	4
1.3 OPTIMIZATION FOR PARAMETER-STUDIES.....	6
CHAPTER 2	7
LITERATURE REVIEW	7
2.1 SCIENTIFIC WORKFLOWS	7
2.1.1 Workflow Design.....	9
2.1.2 Workflow Scheduling.....	13
2.1.3 Fault Tolerance in Workflow Management.....	17
2.1.4 Information Retrieval.....	19
2.2 PEGASUS WORKFLOW MANAGEMENT SYSTEM	20
2.3 CONDOR DAGMAN	24
2.3.1 Condor DAGMan Node.....	25
2.3.2 Condor DAGMan Input File.....	26
2.3.3 Condor.....	28
2.3.4 Condor-G	32
2.3.5 Fault-tolerance in Condor DAGMan – Rescue DAGs	34
2.4 TRIANA.....	34
2.5 TAVERNA.....	35
2.6 ASKALON	36
2.7 KEPLER.....	38
CHAPTER 3	39
DECENTRALIZATION OF WORKFLOW ORCHESTRATION.....	39
3.1 MOTIVATION	39
3.1.1 Multi-site Workflow Mapping Scenarios.....	40
3.2 FEASIBILITY ASSESSMENT FOR MULTI-SITE WORKFLOW ORCHESTRATIONS.....	43
3.2.1 Simulation Setup	44
3.2.2 Synthetic Workflows Simulation Results	46
3.2.3 Real-world Workflows Simulation Results.....	49
3.3 DESIGN OF THE DECENTRALIZATION FRAMEWORK.....	53
3.3.1 DAG Patterns	54
3.3.2 DAG Transformation Patterns	55
3.4 PROTOTYPE IMPLEMENTATION	58
3.4.1 Creation of the Aggregated DAG Specification	59
3.4.2 Transformation Process for the Aggregated DAG Specification.....	60
3.4.3 Case Scenario for the Implementation of Decentralized Orchestration.....	62
3.5 PERFORMANCE EVALUATION	65

3.5.1 Experimental Setup	66
3.5.2 Synthetic Workflow Evaluation Results	68
3.5.2 Montage Workflow Evaluation Results	69
CHAPTER 4	72
RUN-TIME ADAPTATION FOR WORKFLOW ORCHESTRATION.....	72
4.1 MOTIVATION	72
4.2 RUN-TIME ADAPTATION PROCESS	74
4.3 SYSTEM DESIGN	74
4.4 RUN-TIME ADAPTATION FRAMEWORK	75
4.4.1 Monitoring and Detection	76
4.4.2 Planning.....	77
4.4.3 Enactment.....	78
4.5 PROTOTYPE IMPLEMENTATION	83
4.6 PERFORMANCE EVALUATION	85
4.6.1 Experimental Setup	86
4.6.2 Evaluation Results under Constant Overload.....	88
4.6.3 Evaluation Results under Partial Overload	91
CHAPTER 5	93
OPTIMIZATION TECHNIQUES FOR PARAMETER STUDIES	93
5.1 BACKGROUND	93
5.1.1 Single-level Parameter-sweep Workflows	95
5.1.2 Multi-level Parameter-sweep Workflows	96
5.2 OPTIMIZATION PATTERNS	97
5.2.1 Parameter Initialization / Data Distribution	98
5.2.2 Post-Processing / Data Aggregation.....	100
5.3 PROTOTYPE IMPLEMENTATION	102
5.4 PERFORMANCE ANALYSIS	104
5.4.1 Molecular Modeling and Dynamics.....	104
5.4.2 Ensemble Forecasting – Weather Forecasting	107
CHAPTER 6	112
CONCLUSIONS.....	112
REFERENCES	116
VITA.....	126

LIST OF FIGURES

FIGURE	PAGE
Figure 2.1: Overview of the workflow lifecycle.....	9
Figure 2.2: Pegasus software stack	21
Figure 2.3: Basic configuration and interactions among Pegasus components	22
Figure 2.4: Condor DAGMan interactions during DAG execution.....	24
Figure 2.5: Condor DAGMan node	25
Figure 2.6: Diamond DAG	26
Figure 2.7: Diamond DAG input file.....	27
Figure 2.8: Sample Condor submit description file	32
Figure 2.9: Sample Condor-G submit description file.....	33
Figure 3.1: Speedup values simulating multiple domain executions of synthetic workflows	46
Figure 3.2: Average makespan values for the synthetic workflows with 320 tasks	48
Figure 3.3: DAG structure and task types of a Montage workflow [106]	50
Figure 3.4: Speedup values simulating the multiple domain executions of the Montage workflow.....	51
Figure 3.5: DAG structure and task types of a LIGO Inspiral Analysis workflow [106].....	52
Figure 3.6: Speedup values simulating the multiple domain executions of the LIGO Inspiral Analysis workflow.....	53
Figure 3.7: DAG specification stages according to our decentralized orchestration framework	54
Figure 3.8: DAG patterns.....	54
Figure 3.9: DAG transformations for the Sequence DAG pattern.....	56
Figure 3.10: DAG transformations for the Fork/Branch DAG pattern.....	57

Figure 3.11: DAG transformations for the Join DAG pattern	58
Figure 3.12: Sample workflow mapped on two sites.....	62
Figure 3.13: Basic Condor DAGMan specification for the workflow in Fig. 3.12	62
Figure 3.14: Transformed DAG structure at site WFM-1 after the transformation process.....	63
Figure 3.15: Transformed DAG specification to be orchestrated at site WFM-1	64
Figure 3.16: Transformed DAG structure at site WFM-2 after the transformation process.....	65
Figure 3.17: Transformed DAG specification to be orchestrated at site WFM-2	65
Figure 3.18: Speedup values obtained for decentralized versus centralized orchestration of the synthetic workflow	69
Figure 3.19: Speedup values obtained for decentralized versus centralized orchestration of the Montage workflow.....	70
Figure 4.1: System design for the decentralized run-time adaptation framework.....	75
Figure 4.2: DAG adaptations for the Sequence DAG pattern	78
Figure 4.3: DAG adaptations for the Fork/Branch DAG pattern.....	79
Figure 4.4: DAG adaptations for the Join DAG pattern.....	79
Figure 4.5: Patch DAG patterns corresponding to the adapted Sequence patterns	81
Figure 4.6: Patch DAG pattern corresponding to the adapted Join pattern in a scenario where a single site re-maps S_T	82
Figure 4.7: Patch DAG pattern corresponding to the adapted Join pattern in a scenario where two sites re-map the tasks in S_T	83
Figure 4.8: Adaptation process at the originating site	84
Figure 4.9: Adaptation process at the destination site	85
Figure 4.10: DAG structure of the synthetic workflow	86
Figure 4.11: Makespan values for the synthetic workflow under constant overload	89
Figure 4.12: Makespan values for the Montage workflow under constant overload.....	90

Figure 4.13: Average queue wait times for the Montage workflow under constant overload.....	90
Figure 4.14: Makespan values for the synthetic workflow under partial overload	91
Figure 4.15: Makespan values for the Montage workflow under partial overload.....	92
Figure 5.1: DAG structure of a single-level parameter-sweep workflow.....	95
Figure 5.2: DAG structure of a two-level parameter-sweep workflow	97
Figure 5.3: DAG-mapping and centralized orchestration of the Fork/Branch pattern	98
Figure 5.4: Overall view for the decentralized orchestration of the Fork/Branch pattern	99
Figure 5.5: Overall view for the optimized Fork/Branch pattern DAG orchestrated in decentralized manner	100
Figure 5.6: DAG-mapping and centralized orchestration of the Join pattern.....	100
Figure 5.7: Overall view for the decentralized orchestration of the Join pattern	101
Figure 5.8: Overall view for the optimized Join pattern DAG orchestrated in decentralized manner	102
Figure 5.9: WRF Ensemble Forecast	110

CHAPTER 1

INTRODUCTION

Scientific workflows are abstractions that capture the business logic of many complex applications in different scientific disciplines. More specifically, these workflows encapsulate and represent all of the various tasks and data artifacts associated with the application lifecycle. Regardless of the size and complexity of the specific scientific workflow, Directed-Acyclic-Graphs (DAGs) are a powerful and well-established method used by many scientists/developers.

Lifecycle of a typical scientific workflow begins with the specification of individual tasks and artifacts, and dependencies among them. This specification is free of some concrete run-time specific details, and it is mostly referred to as the abstract workflow. Abstract workflows more often than not do not address run-time specific details, such as the exact names and locations associated with data artifacts and details about the exact mapping of tasks on physical resources. For such an abstract workflow to run successfully to completion, those details need to be determined prior to or during the execution of the workflow. This process can be referred to as the concretization of the workflow. However, the concretization of the workflow should not be the responsibility of the scientist, or even the application developer; as a matter of fact, it has to be automated as much as possible based on the criteria provided by the user.

During the concretization process, one essential step is the mapping of workflow tasks onto physical resources. The main goal here is to achieve the minimum makespan possible for the execution of the whole workflow. Makespan metric here can be defined as the total wall-clock time it takes to execute all of the tasks of a workflow. As such,

characteristics of tasks (e.g. estimated runtime) and data artifacts (e.g. estimated size), as well as the availability and characteristics of physical resources, play a major role during this mapping process.

Due to the highly parallelizable nature of large-scale workflows, parallel tasks can be ideally deployed on parallel resources. Sometimes, these parallel resources may span across multiple computational domains, as in the case of a hybrid (public + private) cloud, academic cloud (e.g. FutureGrid [1]), and national/international cyberinfrastructure (e.g. XSEDE [3], Open Science Grid [2]). In such a scenario, mapping and orchestration of the workflow also spans across multiple domains. The key common attributes of such multi-site resources are the heterogeneity and dynamicity of the resources in terms of size and capability, as well as heterogeneous access and priority rights assigned to the users by local administrators. All these factors pose many challenges to the successful and effective execution of workflows conforming to the makespan requirements of the user.

In this dissertation, we tackle three main issues to help improve the orchestration efficiency of large-scale workflows that span across multiple sites of resources. First, we provide a generic decentralization framework for such workflows. This framework also provides the common infrastructure for our efforts, tackling the second and third issues. Second, we provide generic run-time adaptation mechanisms to help alleviate the problems mainly associated with the dynamic nature of resources. Third, we provide additional mechanisms to further optimize the orchestration of large-scale parameter studies.

1.1 DECENTRALIZATION OF THE WORKFLOW ORCHESTRATION

Regardless of being in an abstract or concrete form, a scientific workflow is usually crafted and enacted at a single central location. This means, except for a few proprietary solutions, the execution logic of the workflow is handled by a single, central workflow execution manager. This central manager keeps track of the progress of tasks and coordinates the timely execution of each task based on the specifications of the workflow.

The central workflow manager handles the execution of the workflow even if the mapped tasks of the workflow span across multiple sites. Especially in such a scenario, employment of a single central workflow execution manager raises several efficiency and decision-making accuracy issues. First of all, employing a single central manager necessitates each individual activity to be monitored and orchestrated by this central manager. For a large-scale workflow (i.e. comprised of a large number of tasks) that is mapped on multiple and potentially long-distance sites, orchestration efficiency becomes a real issue. Another important issue is the level of information sharing among partnering sites. The more detailed resource and workload information that is shared among partners, the better decisions can be made by the workflow execution manager(s) to take actions in response to changing conditions. However, due to administrative and technical reasons (e.g. size of information, delay), it is not possible for a remote workflow execution manager to have the same level of information about a certain site's resources compared to its local counterpart. This can cause the central workflow execution manager to make non-optimal decisions during run-time adaptation.

To overcome these issues, our proposal is to transfer the responsibility of the orchestration of the whole workflow from a single workflow execution manager to several collaborating workflow execution managers. According to this, each local workflow execution manager is going to be responsible for the orchestration of the tasks that are mapped locally. At the same time, peer local workflow execution managers synchronize among each other when necessary, specifically, to fulfill the requirements of those control/data dependencies that span across them. By collaboratively carrying out these activities, the orchestration of the whole workflow is achieved without affecting the business logic of the workflow. Through this peer-to-peer orchestration approach, we are able to provide (i) improved efficiency for large-scale workflow executions, (ii) better results from run-time adaptation.

We propose a systematic and generic framework for transforming the centralized orchestration to a collaborative orchestration via the utilization of DAG transformation patterns. Also, we provide a prototype implementation of our framework on a standard workflow orchestration tool. However, our framework is generic enough and can be easily incorporated by other orchestration tools.

1.2 RUN-TIME ADAPTATION

Due to the dynamic nature of the execution environment, certain changes may need to be made to the original workflow execution plan at the run-time to meet users' QS requirements. The most common and obvious dynamic change in the execution environment is the availability of hardware resources for the utilization of workflow tasks. The availability of these resources may change basically due to hardware failures, increased workload, and higher priority tasks being deployed in the system. Especially

long-running and large-scale workflows are highly susceptible to these kinds of changes in the execution environment. Under these circumstances, to be able to execute a workflow successfully within QS requirements, proper changes have to be made to the original execution plan.

There are two main issues involved within the run-time adaptation process. The first major issue is the planning phase for the run-time adaptation. This phase includes the continuous monitoring of workflow progress and resources, and detecting a situation that necessitates the run-time adaptation process. After the detection, an appropriate corrective action has to be planned to cope with the situation. The second major issue in the run-time adaptation process is the enactment of the proposed adaptation plan to the ongoing workflow execution process in an efficient and non-intrusive manner. In our studies, we focus only on the second aspect of the run-time adaptation process.

A standard run-time adaptation plan [76 – 78] basically makes changes to the original execution plan by modifying the mapping (hence, the execution) site of tasks. Modifications to the mapping site of tasks need to be reflected and implemented accordingly by the workflow execution manager(s). Here, we provide a run-time adaptation framework that integrates with the decentralization framework briefly explained above. One key aspect of our framework is the low-level of intrusiveness to carry out the adaptation process. Our pattern-based framework has little effect on the ongoing workflow execution process. The peer workflow execution managers implement the re-mapping of tasks without any disruption to the orchestration of the rest of the workflow.

1.3 OPTIMIZATION FOR PARAMETER-STUDIES

Parameter studies enable the conduct of research for a certain experiment on a varying set of data and under various possible circumstances. Through the automated design and execution of such studies, large amounts of data/parameters can be processed and analyzed; and as a result more accurate research outcomes can be acquired. These types of studies are prevalent in a large and diverse group of research fields, such as bioinformatics, earthquake science, weather predictions, and molecular dynamics.

Automated design of parameter studies generally results in a simple and well-defined structure, which can be easily represented as a DAG-based workflow. We will be referring to this type of workflow as a parameter-sweep workflow [79, 80]. Depending on the size and scale of the parameter study, these workflows may contain large numbers of computational tasks, which are generally highly-parallelizable. Built up on our existing decentralized orchestration framework, our optimization mechanisms proposed are targeted especially for large-scale parameter-sweep workflows. However, those optimization mechanisms may also be applicable to more general workflow instances.

By exploiting the specific characteristics of parameter-sweep workflows, we introduce patterns to optimize the decentralized orchestration process of such workflows. Optimization patterns we introduce suggest minor changes in the structure of the workflow and the business logic of certain tasks. As long as they are done properly, we argue that these changes would not affect the validity and integrity of results. To this end, consultation with a scientist and/or a domain expert to verify the appropriateness of such changes may be optional or required depending on the case scenario.

CHAPTER 2

LITERATURE REVIEW

2.1 SCIENTIFIC WORKFLOWS

Scientific workflows [81] are standard abstractions used to represent composition of large-scale scientific applications. Such composite applications are typically comprised of multiple stand-alone tasks brought together to perform complex operations. Different phases of such complex applications are crafted separately and then put together in the form of a workflow. The relationships among tasks that depend on each other are specified in the form of control/data dependencies among them. We can see many examples of such workflows in astronomy [4], climatology [5], bioinformatics [6], and many other scientific disciplines. In general, workflows can be categorized as computation-intensive or data-intensive workflows, depending on the majority of work being performed on computation or data transfer.

Most scientific workflows can be abstracted and represented as a DAG $G = (V, E)$, where $V = \{V_1, V_2, \dots, V_n\}$ is the set of vertices that correspond to the tasks comprising the workflow and $E = \{E_1, E_2, \dots, E_m\}$ is the set of edges that correspond to the dependencies among tasks in V . Any workflow manager's fundamental task is to successfully execute each of the tasks in V complying with the dependency relationships among them. For example, if there is a control/data dependency between tasks V_x and V_y , V_y can start execution only after that dependency has been met after the successful completion of task V_x .

Workflow management systems [82, 83] deal with various phases of workflows throughout their lifetime. A typical workflow basically goes through composition,

deployment, mapping onto resources, and execution phases. A workflow management system typically receives as input abstract workflow specifications and is required to map them onto concrete distributed resources (possibly managed under different administrative domains). A workflow management system decides where to map each individual task among potential resources based on the information available about the workflow application and available resources. Mapping decisions are given in such a way to optimize the desired objectives, such as application performance, utilization of resources, and cost associated with the execution of the application.

Due to the dynamic and heterogeneous nature of distributed resource environments, there is no deterministic solution to the workflow mapping problem [84]. Also, it is possible that mapping decisions made earlier may need to be revised during the execution of the workflow. After the workflow is mapped onto resources, the execution of the workflow has to be orchestrated. Workflow orchestration basically refers to the complete and accurate execution of all the tasks comprising the workflow, respecting the control and data dependencies among them as specified in the workflow specification.

Fig. 2.1 illustrates the basic architecture and functionalities supported by various components of the workflow management systems. We can categorize the basic functionalities of workflow management systems as build-time functionalities and run-time functionalities. Build-time functionalities refer to the process of defining and modeling the individual tasks that comprise the workflow and dependencies among them. Run-time functionalities refer to the process of carrying out the execution of workflow tasks and the process of interacting with resource environment in association with the workflow execution.

During a typical lifecycle of a certain workflow application, users first utilize some kind of a workflow-modeling tool and generate the specification of the workflow. If this specification is abstract, it has to be transformed into a concrete specification. Then, this concrete specification is passed onto the workflow enactment component (a.k.a workflow execution engine) for execution. Basic functionalities provided by workflow enactment component can be listed as scheduling, fault management, and data movement services.

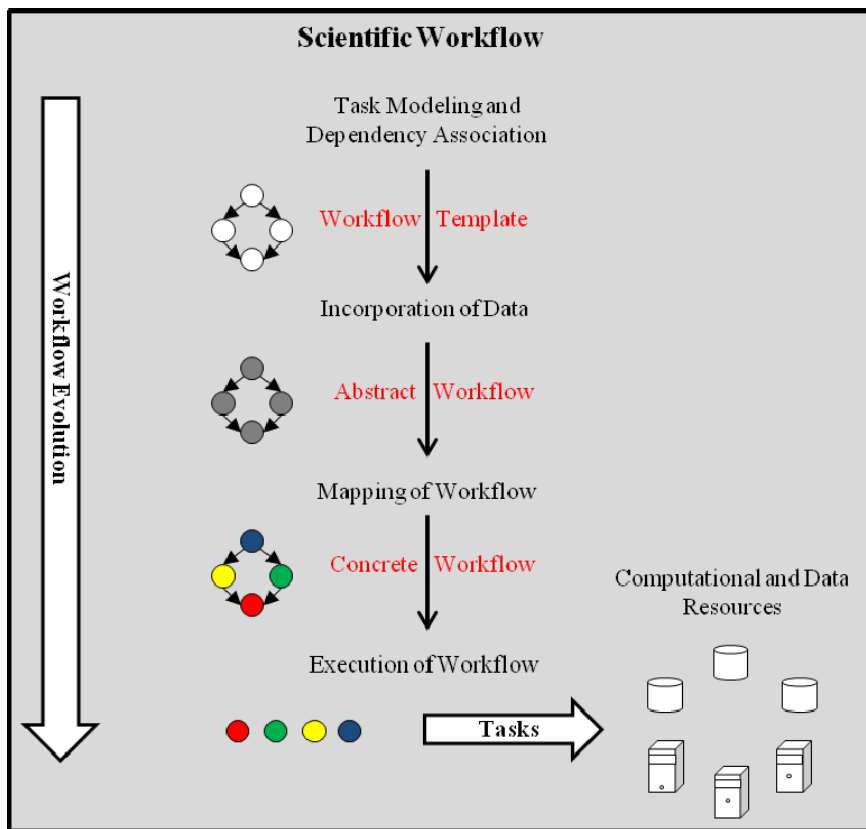


Figure 2.1: Overview of the workflow lifecycle

2.1.1 Workflow Design

Workflow design can be categorized under three main aspects. Those aspects namely are: workflow structure, workflow specification, and workflow composition.

2.1.1.1 Workflow Structure

Connecting multiple tasks according to their dependencies composes a workflow. The structure of the workflow indicates the temporal relationship among tasks. In general, a workflow can be represented as a Directed Acyclic Graph (DAG) or a non-DAG.

In a DAG-based workflow, structure of the workflow can be decomposed into sequence, parallelism, and choice patterns. In a sequence pattern, series of tasks are ordered in succession, with one task starting after a previous task has completed. Parallelism pattern represents tasks which can be performed concurrently, rather than serially. In the choice pattern which is not that common in practice, a task is selected to execute or not at runtime based on the runtime values of its associated conditions.

In addition to all patterns contained in a DAG-based workflow, a non-DAG workflow also includes the iteration pattern. Iteration is also known as loop or cycle. In this pattern, portions of workflow tasks in an iteration block are allowed to be repeated. The iteration structure can be quite handy if used properly, since repeated executions are quite common in scientific applications.

These four workflow structure patterns, namely sequence, parallelism, choice and iteration, can be used to construct many complex workflows. In addition, large-scale workflows can be easily constructed through recurrent usage of these patterns.

2.1.1.2 Workflow Specification

Workflow specification provides the task and dependency information that make up the workflow application. There are two main types of workflow specification,

namely abstract specification and concrete specification. In some literature, concrete specification is also referred as executable specification.

In an abstract specification, a workflow is described in an abstract form in which the workflow is specified without referring to specific resource information that is used for actual task execution. Users/Developers of workflow applications, almost always use abstract specification method to define their workflow application. This way they need not concern themselves with resource-specific details. Those resource-specific details are generally not static information anyways, which makes an otherwise specification (concrete specification) infeasible in those cases. Another advantage of using an abstract specification is the ease of re-use and also the sharing of the workflow specification with others. However, when an abstract specification is used to define a workflow application, it first needs to go through concretization process. Various resource discovery and mapping mechanisms are generally used to help automate this process.

In a concrete workflow specification, tasks are associated with specific resources at build time. In some cases, a concrete specification may need to specify additional tasks to provide data staging in and out of computational resources to facilitate the proper execution of the workflow. In some cases again, it may even be needed to specify the transfer of computational code among resources yet again to facilitate the proper execution of the workflow.

Based on the basic descriptions given about both types of specifications, it is more suitable for users to define workflow applications in abstract form, especially in a diverse and heterogeneous resource environment.

2.1.1.3 Workflow Composition

Workflow composition tools provide capabilities for users to design, create, and assemble components of a workflow. Ideally, these tools need to supply a high-level view to accomplish these tasks as they hide the unnecessary complexities of the rest of the system.

There are two basic categories of tools for workflow composition. In the first category, we find user-directed tools that let users generate workflows through a language-based environment. The second category belongs to the group of composition tools that allow users to generate workflows through graph-based modeling.

As part of language-based composition tools, most provide an Extensible Markup Language (XML) [7] based markup environment (e.g., WSFL [8], BPEL4WS [9], and Gridbus workflow [10]). Some proprietary language formats also do exist (e.g., Condor DAGman [11, 16]). Since, it requires manual composition of workflows through a certain specific syntax; language-based tools are not suitable for most users. Also, even for experts of the syntax, it would be really difficult and time-consuming to compose large workflows. However, these language-based formats are better suited for sharing and manipulation of workflows. So, even though original composition of workflows may be crafted through a graph-based tool, language-based tools bridge the gap between workflow execution engine and the graph-based composition.

Graph-based tools allow composition of workflows through intuitive graphical elements. Most such tools provide a simple click and drop interface and hides all other details of a workflow specification. Petri Nets [12, 13] and UML (Unified Modeling

Language) [14] formats are used by some of these tools. However, majority of tools are proprietary.

Graph-based modeling allows graphical definition of an arbitrary workflow through a few basic graph elements. It allows users to work with a graphical representation of the workflow. Users can compose and review a workflow by just clicking and dropping the components of interest. It avoids low-level details and hence enables users to focus on higher levels of abstraction at application level. The major modeling approaches are Petri Nets, UML [85], and user-defined component. Graph-based modeling is preferred by users as opposed to language-based modeling.

Since graph-based tools are easy to use and generate results quickly, most users prefer them over language-based tools. However, if the workflow to be designed is large, graphical composition gets challenging. In those circumstances, most graph-based tools allow hierarchical composition mechanisms.

2.1.2 Workflow Scheduling

Casavant et al. [15] categorizes task scheduling in distributed computing systems into global task scheduling and local task scheduling. Global task scheduling is concerned with deciding on which resource to execute a task. On the other hand, local task scheduling is concerned with assigning the time-slice of a certain resource to execute a task. Based on these categories, we have to indicate that the workflow scheduling concept falls under the global task scheduling category.

Especially in distributed resource environments under diverse administrative domains, the process of workflow scheduling gets more complex. Because, on top of the existing complexities of scheduling workflows in a single-domain resource environment,

resource capabilities and policies are now diverse and heterogeneous under these circumstances.

Task scheduling is a very complicated and extensively studied area which is not addressed here in a comprehensive manner. We only provide those major concepts and issues most relevant to the concept of workflow scheduling [86 – 90]. Namely, we discuss various workflow scheduling architectures and the planning schemes for workflow scheduling.

2.1.2.1 Workflow Scheduling Architecture

The architecture of the workflow-scheduling infrastructure is very important for scalability, autonomy, quality and performance of the system. Three major categories of workflow scheduling architecture are centralized, hierarchical and decentralized scheduling architecture.

In a centralized architecture, one central workflow scheduler makes scheduling decisions for all tasks in the workflow. The scheduler should have the information about the entire workflow and periodically collects information of all available resources. Ideally, the centralized architecture results in efficient scheduling, due to the fact that the central scheduler has the up-to-date information about all the resources. However, the major disadvantage of centralized architecture is its scalability. Scalability of the centralized architecture suffers as the size of the workflow and resources grow. Another disadvantage with it is the problem of single-point of failure.

In hierarchical scheduling architecture, there is a central manager and multiple sub-workflow schedulers. The central manager is responsible for controlling the overall workflow execution and it assigns sub-workflows to the low-level schedulers. Lower-

level schedulers are responsible for scheduling tasks in a sub-workflow onto resources owned by the local organization. The biggest advantage of the hierarchical scheduling architecture is that each scheduler can utilize different scheduling policies. However, this architecture also suffers from the single-point of failure problem.

In decentralized scheduling architecture, multiple schedulers coordinate the scheduling of the whole workflow without a central scheduler. Peer schedulers communicate with among themselves and can exchange sub-workflow schedules based on their local loads. Decentralized architecture is even more scalable than the hierarchical architecture; however it requires more complicated effort to establish and sustain the coordination among multiple schedulers.

Both hierarchical and decentralized scheduling architecture are more scalable than the centralized architecture. However, since the scheduling of sub-workflows is done independently, the overall workflow schedule may not be optimal. Another problem is that, if not coordinated properly, multiple schedulers may produce conflicting scheduling decisions.

2.1.2.2 Workflow Scheduling Plans

There are three schemes to plan the scheduling of workflows. These schemes namely are: static scheme, dynamic scheme, and hybrid scheme. In a static scheme, concrete workflow specification is generated prior to the execution of the workflow. In dynamic scheme, scheduling of workflow tasks is done at run-time. Hybrid scheme blends these two schemes and performs the scheduling of workflow tasks prior to run-time (static scheme); however the active scheme is revised and updated at the run-time if needed (dynamic scheme).

Static workflow scheduling scheme is also referred as full-ahead planning. In full-ahead planning [91, 92], resource information is utilized to make scheduling decisions only prior to the execution of workflow. Thus, full-ahead planning is more suitable on resource environments where the availability and capability of resources do not change much over time. Generally, two different strategies are used to perform full-ahead planning. One of them is the user-directed planning. In this strategy, users decide or direct the decision for the scheduling of tasks based on their knowledge and experience regarding both resource environment and task execution. The other strategy is the simulation-based planning. In simulation-based planning, scheduling decisions are made based on the best schedule achieved simulating the execution of workflow tasks.

Dynamic workflow scheduling scheme is also referred as just in-time scheduling [93, 94]. Accordingly, the scheduling of each task or group of tasks is done at run-time right before the task is ready to be executed. This scheme is proposed for and suitable especially for highly dynamic resource environments. In such resource environments, availability and utilization of resources change dramatically over time. Thus, it is very difficult to come up with efficient scheduling decisions with a static plan. The major disadvantage of dynamic scheme is the overhead incurred due to run-time scheduling of tasks.

Finally, the hybrid scheme brings the best of static scheme and dynamic scheme together. As mentioned, using a hybrid scheme [76, 95], first a static scheduling plan is made prior to workflow execution. Then, periodically or when the need arises, scheduling plans are updated at run-time. Hybrid scheme is suitable especially for resource environments where the availability and utilization of resources change occasionally.

2.1.3 Fault Tolerance in Workflow Management

Execution of a task may fail due to myriad reasons in almost all computerized systems; which must be dealt with using proper methods accordingly with the specific environment. Workflows are made up of several tasks, and those tasks are prone to failure. If not configured in a specific way, the failure of even a single task may result in the failure of the whole workflow.

There are many reasons that would induce task failure on a single resource. In a dynamic and heterogeneous distributed resource environment situation gets worse. For this type of resource environments, some common reasons that would cause task failure include: non-availability of require service or components in a resource, variation in the execution environment configuration or policies, resource overload, and failure on network fabric.

Due to the sheer size and volume of computation and data involved, it is very important that a workflow management system to be capable of certain fault-management mechanisms. A workflow management system has to be able to detect failures and provide mechanisms to handle failures properly.

Workflow failure handling can be managed at either the task-level or workflow-level [96]. At task-level, methods are developed to prevent or alleviate the effects of task failures. Workflow-level methods provide mechanisms to deal with failures by modifying the structure of the workflow accordingly.

2.1.3.1 Task-level Fault Tolerance

Task-level techniques have been widely studied in parallel and distributed systems. Some common techniques include task retry, alternate resource, application checkpoint/restart and replication.

Task retry technique is a very simple and intuitive failure recovery technique. In this technique, same task is tried to be executed on the same resource after the failure. Alternate resource technique is also simple and similar to the previous technique. In this case, the failed task is submitted to execute on another resource. Application checkpoint/restart techniques transparently creates checkpoint during task execution and in the case of a failure resumes the execution of the task from last stable checkpoint. In the replication technique, the same task is simply run on multiple resources at the same time. This way, the success rate of the application improves as long as one of the replicas completes successfully.

2.1.3.2 Workflow-level Fault Tolerance

Workflow-level fault-handling techniques include alternate task, redundancy, user-defined exception handling and rescue workflow.

In the alternate task technique, another implementation of the failed task is considered. In the redundancy technique, multiple alternative implementations are considered at the same time, similar to the replication technique. In the user-defined exception handling technique, user may specify a certain way to deal with a failure. For example, the user may tag a certain task as optional, so that the workflow execution can continue even if that task fails. In the rescue workflow technique, first developed by Condor DAGMan [11, 16], task failures are ignored and the execution of the workflow is

continued until it cannot progress any further. Then, the system automatically generates a special workflow specification called rescue DAG. This rescue DAG can be submitted later after any necessary actions are taken to mitigate the problem.

2.1.4 Information Retrieval

A workflow management system does not execute the tasks itself; rather it simply coordinates the execution of the tasks by the computational resources. As mentioned earlier, an abstract workflow has to be mapped onto resources, which would result in a concrete workflow. During this mapping process, information about the resources has to be retrieved from appropriate sources. The information retrieval [97, 21] can be categorized as three different dimensions, namely static information, dynamic information, and historical information.

Static information refers to information that does not typically change over time. This type of information includes infrastructure-related information, such as the number of processing cores and the processor clock speed. It also refers to some basic software configuration information, such as the operating system, and certain software libraries. Another type of information includes user-related and accounting information. Generally, the static information is used to make first-level elimination during the mapping process to eliminate those resources that are not suitable to perform task execution.

In a dynamic and distributed resource environment several types of dynamic information is very important to the proper mapping of workflow. Some basic information that changes dynamically on resources include, queue length of a cluster, processor load, network load, and available disk space. Dynamic information is generally used to rank resources that are available to execute a task.

Historical information refers to the information stored about resources and applications based on previous events. This information can be used to predict future behavior of resources and applications. Using this information, more reliable decisions can be made regarding the mapping and execution of workflow tasks.

2.2 PEGASUS WORKFLOW MANAGEMENT SYSTEM

Pegasus [17, 18] is a workflow management system project developed at University of Southern California. At the highest level Pegasus works as a workflow mapper. Accordingly, it takes an abstract workflow specification and generates a concrete workflow by mapping the components of the workflow on local, Grid and/or cloud computing resources.

Pegasus consults various resource information services to find the computational and application resources needed for the workflow. A service called Replica Location Service (RLS) [19] is used to locate the replicas of the data required in the workflow. Similarly, another service called Transformation Catalog (TC) [20] is used to find the location of the logical application components (e.g. application software) in the workflow. Pegasus also queries Globus Monitoring and Discovery Service (MDS) [21] to retrieve information regarding resources and their characteristics.

Default behavior for scheduling of workflow tasks in Pegasus is based on static planning. However, it also provides support for just in-time scheduling and pluggable task scheduling strategies. The concrete workflow is transformed into Condor jobs which are orchestrated by Condor DAGMan metascheduler. Fig. 2.2 illustrates the high-level layout of the software stack for a typical Pegasus workflow management system configuration.

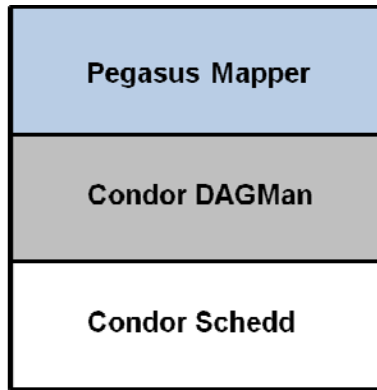


Figure 2.2: Pegasus software stack

Based on this high-level layout, Fig. 2.3 gives a more detailed look at the interactions and configurations of these components. At the highest-level Pegasus mapper transforms an abstract workflow into a concrete workflow. The concrete workflow is then passed onto Condor DAGMan, which acts a metascheduler or more commonly referred as in the literature workflow execution engine. Condor DAGMan is a metascheduler, hence it does not perform the actual scheduling of jobs itself. In fact, it too depends on another layer of software, which is Condor Schedd (scheduler daemon). Condor job scheduler performs the actual scheduling of jobs on resources. More information about Condor DAGMan and Condor will be given in the next sections.

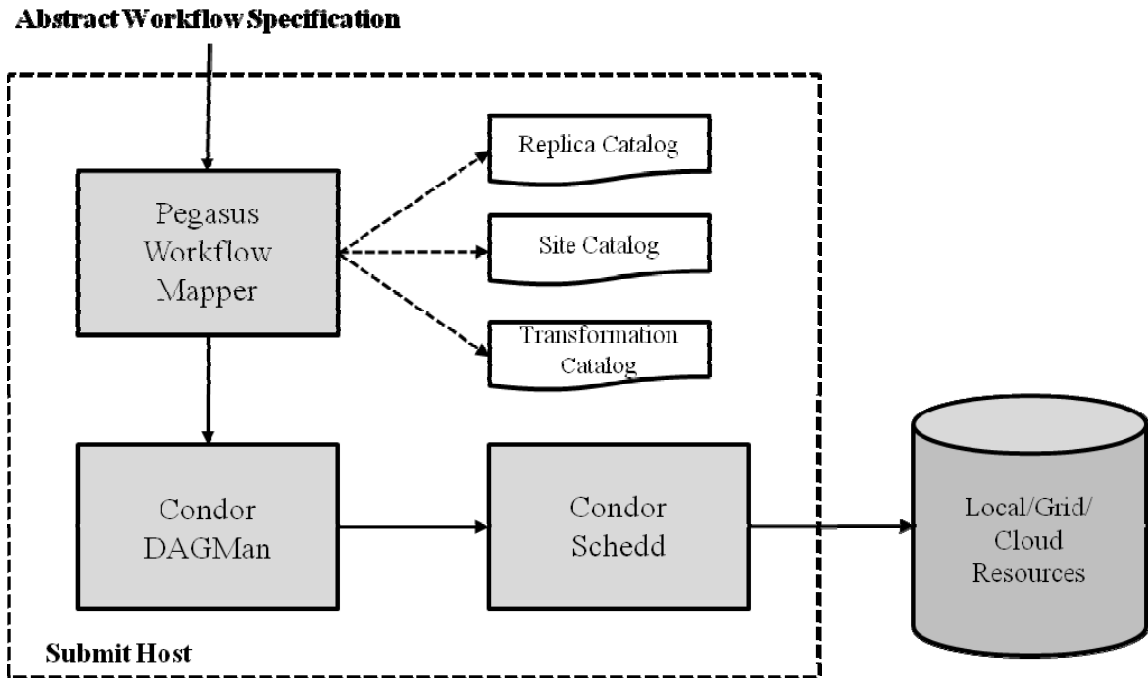


Figure 2.3: Basic configuration and interactions among Pegasus components

After the mapping of the workflow is completed, Pegasus performs possible runtime optimizations. First optimization technique is the workflow reduction, where getting rid of some tasks by accessing the already existing output reduces the components found in the workflow. Second optimization technique is the clustering, where more than one job are clustered into a single cluster of jobs to increase the granularity, so as to decrease the scheduling overhead. Third optimization technique is the use of multi-processor systems.

Workflow reduction process basically involves skipping the computation of certain tasks if the outcome of those tasks already exist somewhere in the system (found via RLS). The rationale behind this optimization is the assumption that the computation of the data will take longer than transferring the existing data. If such an optimization is

performed, in the concrete workflow some of the computational tasks are replaced by data transfer tasks.

Clustering optimization is performed due to two major concerns. First, by clustering multiple individual tasks into a single task, the overhead for maintaining large number of tasks is lowered by the system. Second, multiple jobs are dispatched at once at the target environment instead of being dispatched one by one. This decreases the waiting time for jobs at the execution environment.

Another possible optimization technique is the use of MPI (Message Passing Interface) clustering. In this technique, multiple independent jobs are wrapped inside a simple MPI program. By this technique, two kinds of optimization benefits mentioned in the previous paragraph are achieved. Also, due to the independent characteristics of these tasks, they can be executed completely in parallel, which further improves the performance.

After the optimization operations are completed, data stage-in and data stage-out jobs are added to the concrete workflow specification. This is a critical step in transforming an abstract workflow specification into a specification that can be actually executed on the resource environment. Because abstract workflow specifications don't include the physical resource and naming information for the data artifacts that will be consumed/generated throughout the lifecycle of the workflow.

As the final step, Pegasus performs the generation of specific artifacts, namely the submit files, that are passed on to the Condor DAGMan. Condor DAGMan takes all these submit files and coordinates the execution of workflow tasks accordingly. The details of these activities will be discussed in the next sections.

2.3 CONDOR DAGMAN

The Condor Directed Acyclic Graph Manager (DAGMan) [11, 16] is basically a meta-scheduler for Condor jobs. As such, it handles the dependencies among jobs. As its name implies, it uses DAG as the workflow structure to represent job dependencies. Each job is a node in the DAG structure and the edges of the graph identify the dependencies among jobs. Each node can have any number of parent or children nodes. Children nodes cannot run until their parent nodes have completed their execution. As the definition of it implies, cycles are prohibited in a DAG. That means two jobs cannot have bidirectional dependencies, or in other words they cannot descend from one another.

Fig. 2.4 illustrates the flow of interactions during the execution of a sample DAG (typically specified in a .dag file). As can be seen, Condor DAGMan acts only as a metascheduler. The actual scheduling of each individual job is handled by another component, Condor or Condor-G (discussed later).

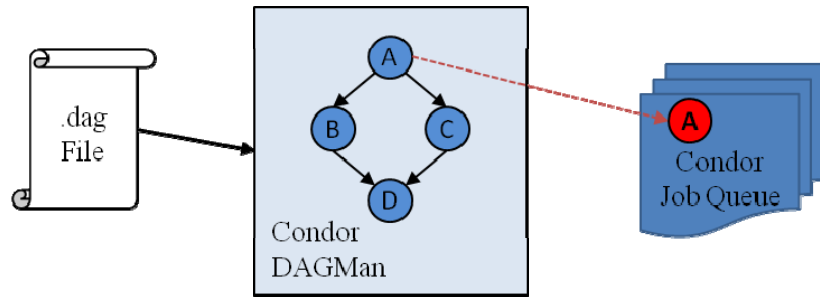


Figure 2.4: Condor DAGMan interactions during DAG execution

Condor DAGMan does not support automatic intermediate data movement. Thus, users are responsible to specify data movement actions through pre-processing and post-processing commands associated with the processing of the job.

2.3.1 Condor DAGMan Node

In Condor DAGMan architecture, a DAG is composed of nodes that define computational and data requirements of the application. Fig. 2.5 illustrates the elements of a Condor DAGMan node. The main component is the Condor Job, which is specified in a specific Condor job submission file (discussed in the next section about Condor). PRE and POST scripts are optional components. PRE script is commonly used to perform data stage-in operations. POST script is commonly used to clean up unnecessary files or to perform data stage-out operations.

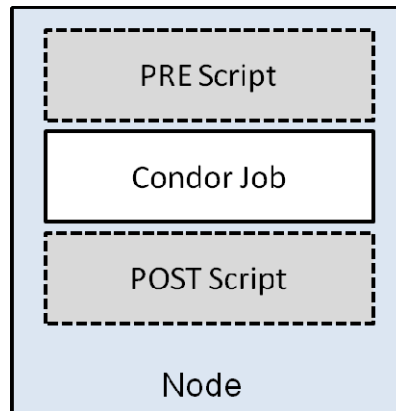


Figure 2.5: Condor DAGMan node

Condor DAGMan monitors the exit values of jobs and scripts, and behaves accordingly with these values. If the PRE script fails then the Condor job is not run. If the PRE script returns an exit value of 0, then the Condor job can be submitted. If the Condor job fails and there is no POST script in the node, then the DAG node also fails. However, if the Condor job fails and there is a POST script in the node, then the node success/failure is determined by the exit value of the POST script. By default, the POST script is run even if the Condor job fails. Thus, it is important to make sure to check the return value of the Condor job to get expected results.

Table 2.1 provides node success/failure outcomes based on all possible values of node components. (S represents success, and F represents failure. (-) symbol indicates the lack of a certain script. The asterisk (*) symbol indicates that the POST script is run.)

PRE	-	-	F	F	S	S	-	-	-	-	S	S	S	S
JOB	S	F	not run	not run	S	F	S	S	F	F	S	F	F	S
POST	-	-	S*	F*	-	-	S	F	S	F	S	S	F	F
node	S	F	S*	F	S	F	S	F	S	F	S	S	F	F

Table 2.1: Node success or failure definition [22]

2.3.2 Condor DAGMan Input File

The input file used by Condor DAGMan is called a DAG input file. All items are optional, other than that there must be at least one JOB item.

The lines starting with the pound character (#) identifies that line as a comment.

A simple diamond-shaped DAG is shown in Fig. 2.6. This DAG contains 4 nodes.

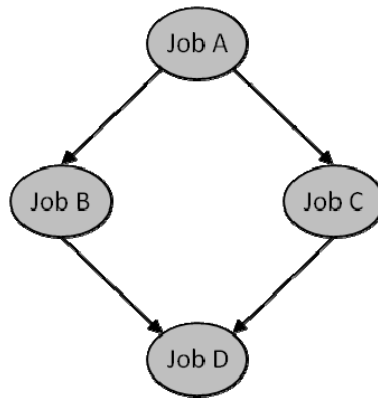


Figure 2.6: Diamond DAG

A sample DAG input file for this diamond-shaped DAG is illustrated in Fig. 2.7.

```
# diamond.dag input file
# contains 4 nodes

JOB A A.condor
JOB B B.condor
JOB C C.condor
JOB D D.condor
PARENT A CHILD B C
PARENT B C CHILD D
```

Figure 2.7: Diamond DAG input file

Here, we briefly explain the basic keywords and the structure of a DAG input file description. First of all, every DAG input file must contain at least one Condor job that is indicated by the JOB keyword. Each JOB must be associated with a unique name (JobName). With each JOB definition, the name of the associated Condor submit file must also exist (e.g. ‘A.condor’). The requirements and details of the actual Condor job is specified within this Condor submit file. There is also an optional DONE keyword that can be used in a JOB line. DONE keyword indicates that the node has already completed. This is mainly used by rescue DAGs that are generated automatically by Condor DAGMan when the execution of the workflow fails for any reason. By associating those nodes that have already completed with the DONE keyword, Condor DAGMan avoids those tasks to be run again when the rescue DAG is submitted at a later time.

The PARENT and CHILD keywords specify the dependencies among jobs within the DAG. Each node has to be specified as either parent and/or a child in a DAG. A parent node must be completed successfully before any of its children may be started. A child node may be started only after all its parents have completed successfully.

To specify a PRE script associated with a certain Condor Job, SCRIPT PRE keyword is used together with the associated JobName. Similarly, to specify a POST

script associated with a certain Condor Job, SCRIPT POST keyword is used together with the associated JobName. In both cases, the name of the specific script to be run must be provided.

There are several other optional keywords that can be used in a DAG input file. However, the keywords described above provide sufficient capability for most of the typically used DAG workflows.

2.3.3 Condor

Condor [23] is a specialized resource management system project developed at the University of Wisconsin-Madison. Condor provides a High Throughput Computing (HTC) environment from large collections of distributed computing resources ranging from desktop computers to supercomputers.

Condor also performs as a batch scheduler, and accordingly provides a queuing mechanisms, scheduling policies, and priority schemes. End-users submit their computational jobs through an interface, which are placed in one of the Condor job queues. Condor then takes care of the handling and execution of the job. Job is dispatched to a specific resource and executed, after which the user is informed regarding the results of their job.

Job dispatch in Condor occurs according to the matchmaking mechanism in Condor. Through matchmaking mechanism jobs and available resources are matched with each other according to their job requirements and resource classified advertisements. When more than one resource satisfies the requirements of job, the resource with higher rank value is chosen.

Condor provides support for checkpoint and migration mechanisms. This way, should there be a fault or change in the utilization of a resource, execution of a task can be continued on a different resource without going over the same calculations again.

Condor also makes it easy for users to run the same job many different times with varying input parameters/data. Condor makes it really simple to construct such job submissions, as well as organizing the outputs of such calculations. This is a very important feature for scientific experimentation where parameter studies are quite commonly used.

2.3.3.1 Submitting and Running Jobs with Condor

Following are the steps needed to run a job using Condor.

- Code preparation: A job run under Condor must be able to run as a background batch job. As such, applications that require interactive input/output are not suitable to run over Condor. However, making the interactive portions of the application to progress via using files may make them Condor-ready.
- Choosing the Condor universe: A universe in Condor defines an execution environment for the application. Condor supports several different universes for user jobs: standard, vanilla, grid, java, scheduler, local, parallel, and VM. The universe under which a job runs is specified in the submit description file. If a universe is not specified, vanilla universe is selected by default. The standard universe provides checkpointing and job migration support; however it enforces some restrictions on the applications before they can be eligible to use this universe. The vanilla universe provides less reliability, however it is the most simple and straightforward environment that can be chosen by most applications.

The grid universe allows users to submit jobs hassle-free on a remote resource management system. The java universe allows users to run jobs using the Java Virtual Machine (JVM) as Condor take care of specific details regarding java configuration (e.g. locating JVM binary, setting the 'classpath'). The scheduler universe is typically used to run the Condor DAGMan metascheduler. As a consequence, Condor DAGMan itself runs as a Condor job on the submit host. The parallel universe is aimed for distributed memory programs (e.g. MPI jobs). The VM universe allows users to run jobs on a virtual machine by facilitating the proper disk image and infrastructure for the application.

- Preparing submit description file: All the requirements and details of a job has to be specified in a submit description file. Some of the basic information found in this file includes the executable/binary to be run for the job, input parameters, file information regarding input/output data, resource-related requirements, ranking method for eligible resources, and user notification information (e.g. email information). In this file, user can also specify how many times to run the job, and where to put the associated data for each individual run.
- Job submission: Job is submitted to the Condor queue for execution via a simple 'condor_submit' command.

2.3.3.2 Condor Submit Description File

Condor submit description file specifies all the requirements and details of the job that is expected to be run by Condor. Accordingly, it contains various keywords and parameters that define information about the job such as the executable/binary to be run,

command-line arguments, working directory, input/output files, resource requirement specifications, and so on.

Condor allows easy and convenient approach to run multiple copies of the same program. Users can easily specify in submit description file for each run to use different data sets and to read/write in their own files. Condor simply allows each run to have its own working directory, input/output/error files, and command-line arguments.

The sample submit description file illustrated in Fig. 2.8 submits two copies of the application 'prime' found under '/home/selim' directory. The first copy runs under directory prime_1, and the second runs under directory prime_2. Output, error, and log files are associated with each run separately. This means, first copy will generate prime.out, prime.err, and prime.log files under prime_1 directory, whereas the second copy will generate the same files under prime_2 directory. However, two copies get different command-line arguments for the execution. First copy will find prime numbers between 2 and 1000000, whereas the second copy will find prime numbers between 1000000 and 2000000. For both copies, the standard universe is selected as the execution environment which provides additional reliability for the execution of 'prime'. Both copies request a minimum memory of 4 GB RAM to exist in a resource that is going to execute the application.

```

# Two copies of 'prime' is executed
# Results are stored in different files

Executable      = /home/selim/prime
Universe        = standard
Output          = prime.out
Error           = prime.error
Log             = prime.log
Request_memory  = 4 GB

Initialdir      = prime_1
Arguments       = 2      1000000
Queue

Initialdir      = prime_2
Arguments       = 1000001 2000000
Queue

```

Figure 2.8: Sample Condor submit description file

2.3.4 Condor-G

Condor-G [24] is a special component within Condor and serves as a uniform interface to heterogeneous batch management systems. Through Condor-G, users can utilize resources beyond their own Condor pool of resources. In the meantime, usage of Condor-G is transparent to the users of Condor. Users specify their jobs the same way as a Condor job. The only change is needed in the universe specification. First of all, the grid universe has to be selected as execution environment. Then, Condor-G provides support for various grid/cloud computing systems which are distinguished from each other by their corresponding middleware. Users can differentiate among those via specifying the proper `grid_type`.

Condor-G was originally developed to provide support for Globus middleware [25, 26]. Still, it is the most commonly used `grid_type` by Condor-G. In this case, the `grid_type` has to be specified as `gt2` or `gt5` depending on the specific Globus version at the

remote resource. Once the job specification is done properly, user can submit and manage the job the same way as a regular Condor job. At the same time, Condor-G provides all the same job management capabilities for the job that is available for Condor jobs.

In Fig. 2.9, we show a sample job designed to be submitted and controlled via Condor-G. The associated specification is done at where Grid_resource is given. In this case, the job is submitted to the Lonestar resource located at TACC (Texas Advanced Computational Center). From this specification, it is seen that Globus middleware version 2 is utilized to make the connection to the remote resource. It is also specified that the LSF [27] batch management system is going to be used at Lonestar resource.

```
# Condor-G is used in this example
# Job is submitted to a remote resource at TACC

Executable      = test
Universe        = grid
Grid_resource    = gt2 gatekeeper.lonestar.tacc.teragrid.org/jobmanager-lsf
Output          = test.out
Log             = test.log
Queue
```

Figure 2.9: Sample Condor-G submit description file

With Condor-G, by default, Condor transfers the executable, as well as any input files specified in the job description. For this reason, it is important to make sure that the executable is compiled for the remote platform.

Condor-G provides support for remote resource systems with grid types other than Globus. Those grid types namely are: Nordugrid [28, 29], Unicore [30, 31], batch (for PBS [32, 33], LSF [27], and SGE [34] batch systems), Amazon's Elastic Computing Cloud (EC2) [35], and cream (for gLite [36]).

2.3.5 Fault-tolerance in Condor DAGMan – Rescue DAGs

The simplest yet the most useful fault-tolerance mechanism in Condor DAGMan is the task retry mechanism. The RETRY keyword in Condor DAGMan provides a way to retry the execution of failed nodes for a specified number of times. The use of retry is optional. Task retry is associated with a DAG node. Thus, if a node fails due to some reason and task retry is specified for this node, all parts (including PRE and POST scripts) of the node have to be retried.

As mentioned earlier, Condor scheduler manages the individual job execution in Condor DAGMan. Thus, if a job fails due to the nature of the distributed system, such as loss of network connection, it will be recovered by Condor while Condor DAGMan is unaware of such failures. If the job failures cannot be handled at the Condor level, Condor DAGMan notices the job failure and eventually halts and generates a rescue DAG. In the case of a job failure, the remainder of the DAG continues until no more progress can be made. The rescue DAG indicates the uncompleted portions of the DAG with detail of failures. Users can correct the errors of failed jobs and resubmit the rescue DAG. If the DAG is resubmitted using the Rescue DAG, the successfully completed nodes (labeled as DONE) are not executed again.

2.4 TRIANA

Triana [37, 38] is a visual workflow-oriented data analysis environment developed at Cardiff University. Triana has a GUI to compose workflows and an underlying subsystem which allows the integration with multiple services and interfaces. Through this GUI, user can drag and drop the tools or services to their workflow

application. Each tool has an input and output port, and the user connects dependent tools via cables from these ports.

Underlying subsystem of Triana consists of a collection of interfaces that bind to different middleware and services. Grid Application Toolkit (GAT) [39] interface provides bindings to Globus GRAM, and GridFTP [42]. Grid Application Prototype (GAP) interface provides bindings to JXTA [40], and web services [41]. This integration makes it possible for the user to compose complex and heterogeneous workflows by utilizing different types of services and tools.

Workflow specification in Triana is simple and based on XML. Components are represented in XML by having properties such as name, input/output ports, optional parameters and proxy/reference to the actual component. Dependency among components is specified using parent/child relationships.

Another component within Triana is the gridMonSteer (GMS) which wraps legacy applications to be executed on distributed resources. This component also monitors the execution and steers the workflow based on the progress of the applications.

2.5 TAVERNA

Taverna [43] project is collaboration among several European universities, institutes and industries. The main objective of Taverna is to assist scientists with the development and execution of bioinformatics workflows on distributed computational platforms. To this end, Taverna provides data models, enactor task extensions, and graphical user interfaces. FreeFluo [44] is used as the workflow execution engine in Taverna.

Taverna provides both graphical and language-based models to represent data. Language-based model is based on XML and is called Simple Conceptual Unified Flow Language (SCUFL). SCUFL allows implicit iteration over incoming data sets. Workflow execution engine also provides a multithreading mechanism to speed up the iteration process. Users can configure the Thread property to specify how many concurrent instances are allowed to send parallel requests.

Taverna also provides an intuitive multi-window graphical interface for users to handle and oversee the complete lifecycle of the workflow. Using this interface, users can manipulate workflow specification, select resources, submit the workflow for execution, and monitor the progress of the workflow.

Taverna provides various fault management mechanisms, including task retry and alternate resource mechanisms. Users can also indicate whether a certain task is critical or not. If a certain task is specified as not critical, the workflow execution can progress even if the task fails. In that case, all those tasks dependent on the failed task will also be ignored.

2.6 ASKALON

Askalon [45, 46] is an application development and computing environment developed at the University of Innsbruck, Austria. Askalon is composed of workflow composition service, resource manager, scheduler, workflow execution engine, and performance analysis and performance prediction modules.

Askalon provides two approaches for workflow specification. First approach is a graphical tool that is based on the use of standard UML Activity diagrams. Other approach is via the proprietary language that is based on XML, namely the Abstract Grid

Workflow Language (AGWL) [47]. AGWL has the basic capabilities for composing workflows, such as branching, looping, parallel execution, and also resource specification constructs.

Resource manager in Askalon, namely GridARM, provides resource discovery, advanced reservation and authorization services. GridARM monitors the allocated resources and propagates exceptional situations to the client. It also works as coallocation manager. GridARM can be configured to work with multiple MDS services, such as Globus version 2 and Globus version 4.

Scheduler component in Askalon performs three basic operations: workflow refinement, workflow mapping, and workflow rescheduling. Workflow converter module performs the workflow refinement, to transform the compact but complex specification into a pure DAG-based specification. Scheduling engine module performs workflow mapping, which incorporates different scheduling algorithms. Event generator module uses the monitoring service to monitor the workflow execution and detect whether any execution contracts have been violated. In such a case, scheduler sends a rescheduling event to the workflow execution engine, which generates a new workflow specification based on the current status and sends it back to the scheduler.

Workflow execution engine is responsible for executing the workflow specification, data management activities and also fault management. It also performs static and runtime optimizations, such as archiving and compressing multiple files to be transferred between two sites or clustering multiple jobs to reduce the job submission overhead.

Workflow execution engine also provides fault management mechanisms in three levels; activity level, control-flow level, and workflow level. Activity-level mechanisms include task retry and replication. Control-flow level mechanisms include lightweight workflow checkpointing (in which the workflow state and URL for intermediate data are stored) and task migration. Workflow-level mechanisms include workflow-level redundancy and workflow-level checkpointing (in which the workflow state and actual intermediate data is saved).

2.7 KEPLER

Kepler [48, 49] project is derived from Ptolemy II system [50] and it is one of the popular workflow systems with advanced features for workflow composition. Besides the GUI-based interface, it also provides a composition model in which independent components (actors) communicate through well-defined interfaces. An actor encapsulates the set of parameterized operations performed on input to produce output data. Another component called director imposes the order and timing of actors. This modular composition model of Kepler makes it easy to modify and reuse workflow applications.

CHAPTER 3

DECENTRALIZATION OF WORKFLOW ORCHESTRATION

In this Chapter, we propose a generic framework to decentralize the orchestration of workflows that span across multiple administrative domains. Accordingly, we first explain the motivating scenarios and the issues that necessitate the utilization of such a framework. Later, we present our extensive simulation-based studies to provide us with a high-level feasibility assessment regarding the deployment of various workflows over multiple administrative domains. Then, we explain the design and prototype implementation of our decentralization framework. The main component within our decentralization framework is the DAG transformation process and we demonstrate the application of this process by presenting both the original and transformed DAG specifications over a case scenario. Finally, we compare the performance results obtained from centralized orchestration and decentralized orchestration of both a synthetic and a real-world workflow, through our experiments conducted on an actual multi-site computational infrastructure (i.e. XSEDE [3]).

3.1 MOTIVATION

During the concretization process of the workflow, one essential step is the mapping of workflow tasks onto physical resources. The main goal here is to achieve the minimum makespan possible for the execution of the whole workflow. As such, characteristics of tasks (e.g. estimated runtime) and data artifacts (e.g. estimated size), as well as the availability and characteristics of physical resources play a major role during this mapping process. Based on the availability of resources, the resulting concrete

workflow may span across multiple sites (domains) of resources. Such multi-site resources may be made available to the usage of a specific workflow application perhaps through research collaboration among multiple partners or through a national/international computational infrastructure platform (e.g. XSEDE [3]). The key common attributes of such multi-site resources is the heterogeneity and dynamicity of the resources in terms of size and capability, as well as the lack of a centralized control mechanism.

We propose a framework that facilitates the decentralized orchestration of workflows that are mapped on multi-site resources. Through our decentralized orchestration scheme, orchestration of the whole workflow is achieved collaboratively by local workflow managers deployed at each site. The main advantages of orchestrating a workflow through our decentralized framework are the performance improvement (i.e. makespan reduction) and the preservation of the site autonomy. We provide more details regarding the advantages of our framework as we discuss specific workflow mapping/orchestration scenarios in the next Section.

3.1.1 Multi-site Workflow Mapping Scenarios

We provide three concrete scenarios where multiple sites of resources can be utilized for the mapping of a workflow. We discuss workflow orchestration alternatives available for each scenario. Then, we discuss advantages specific to each scenario through the usage of our orchestration decentralization framework.

- Workflow mapping scenario 1: In this scenario, the user has access to a certain multi-site computational infrastructure (e.g. XSEDE [3]). In such an environment, user can make use of computational and data resources belonging to multiple administrative

domains accordingly with the proper allocations and policies made for him. The real merits of such platforms manifest themselves through the collaborative usage of hardware and software resources belonging to partner sites. To this end, they provide various middleware (e.g. Globus [25]) and software tools (e.g. GridFTP [42], Condor DAGMan [16]) to enable the communication and cooperation among partner sites.

Users can map their workflows on such an environment manually or automatically (e.g. Pegasus [17, 18]) to result in a mapping of workflow tasks that spans across multiple sites. Then, the user can utilize an existing centralized workflow orchestration tool (e.g. Condor DAGMan [16]) to control and manage the orchestration of the whole workflow. In such a scenario, the control, monitor, and management of all the individual tasks in the workflow reside with a single workflow orchestration tool instance located at one of the partner sites. This would result in each interaction between the workflow orchestration tool instance and a remote partner site, for the purpose of control and management of each task, to incur additional overheads. First overhead incurred is simply due to the communication overhead and delay between partner sites. Another type of overhead incurred is due to various middleware and software layer communication and processing delays that are encountered during the interactions.

Our proposal in such a scenario is the decentralized orchestration of the workflow rather than the centralized orchestration. Through our generic decentralization framework, we facilitate the orchestration of a workflow that span across multiple sites in decentralized manner. The main advantage of the decentralized orchestration is the improved performance, achieved via significant reductions in overheads

sustained through centralized orchestration. Another advantage is the preservation of site autonomy. By appointing a local workflow manager at each site responsible for the control and management of those tasks mapped on their own domain, each site can employ their own set of policies rather than a global set of policies enforced by the central workflow manager. These policies include, among others, task dispatch/monitoring policies, fault-recovery policies, and data management policies.

- Workflow mapping scenario 2: In this scenario, the user has access to both local computational resources and to a multi-site computational infrastructure (e.g. XSEDE [3]). The user aims to utilize both sets of resources in combination for the execution of a certain workflow. In such a scenario, there is not an established middleware layer and software tools to allow the integration of both set of resources seamlessly. Accordingly, there is not an available workflow manager tool that is able to achieve the orchestration of a workflow in centralized manner over both sets of resources.

In a scenario as explained above, our goal is to facilitate the utilization of both sets of resources without having to employ any cumbersome proprietary mechanisms. To this end, we propose the orchestration of the workflow in a decentralized manner in such an environment. Following our generic decentralization framework, which operates at the application layer, a certain workflow specification can be transformed into multiple, individually executable workflow specifications. The required communication/synchronization interactions among those peer workflow orchestrations can be provided independently from any other system components, via basic standard mechanisms.

- Workflow mapping scenario 3: This scenario is similar to the previous scenario. The user has access to multiple, independently administered sites of resources. In the previous scenario, one (or more) set of resources belong to an existing multi-site infrastructure (e.g. XSEDE [3]), thus providing the capabilities for centralized orchestration of a workflow in part; which is then combined with an additional set (or more) of resources for improved performance, or due to cost reasons. In this scenario, there are no established mechanisms/tools among any of the sites that would facilitate the centralized orchestration of a workflow.

The solution we propose for this scenario is again the utilization of our decentralized workflow orchestration framework. Following the same approach as explained in Scenario 2, multiple independent sites of resources can be made available for the orchestration of a workflow via basic standard mechanisms.

3.2 FEASIBILITY ASSESSMENT FOR MULTI-SITE WORKFLOW ORCHESTRATIONS

To be able to help us understand the conditions and circumstances where the deployment of a workflow on multiple sites of resources can be beneficial for the user, we conducted an extensive amount and range of simulation studies. Through simulating the execution of workflows with various size and characteristics, we measured the expected makespan values on single site versus multiple sites deployment of workflows. We present the setup conditions and assumptions followed with the results we obtained for a variety of workflows. Referring to these results, we can have an approximate idea regarding the performance expectations of real-world deployment of various workflows.

3.2.1 Simulation Setup

We conducted our simulation studies using the GridSim [98] discrete event-based simulator that has been widely used among Grid computing researchers. We have developed a module within GridSim environment that simulates the behavior of decentralized workflow orchestration approach. Input to this module is a workflow specification that is extended with mapping information of tasks on sites. Mapping of workflow tasks on sites of resources is performed via METIS [99]. METIS is a family of programs for multi-level partitioning of graphs and hypergraphs. Graph partitioning techniques deal with dividing the vertices of a graph into a given number of disjoint sets while trying to minimize the edge-cut value. Edge-cut value of a graph partitioning is the sum of the weights of edges that cross between different sets. The weight of each disjoint set (i.e. sum of the weights of vertices) is also as close to each other as possible according to the result of a weighted graph partitioning. Graph partitioning is widely used in parallel computing for load balancing [100, 101] and task mapping [102] purposes. It has also been adopted [103] to be used to map workflows on heterogeneous distributed environments. The performance metric we measure in this study is the makespan value for the execution of workflows as reported by GridSim.

Our simulations were conducted on both synthetic and real-world workflows. More information regarding the real-world workflow simulation setup will be presented later. For the simulation of synthetic workflows, we generated a wide range of synthetic workflows using TGFF v3.1 [104]. TGFF tool generates pseudo-random DAGs according to user-given input configurations. Using this tool, we generated DAGs of size 40, 80, 160, and 320 tasks where the maximum in-degree (number of edges coming in to

a task) for a task was specified as 4, and maximum out-degree (number of edges going out of a task) for a task was specified as 8. We chose these specific in-degree and out-degree values for two reasons. First, using these values makes it possible for the generated workflows to involve sub-workflows with a certain degree of parallelization. Second, using these values makes the workflows generated with this tool to have a similar structure to most of their real-world counterparts. As such, only a few numbers of tasks in these workflows have high in-degree and/or out-degree values whereas most tasks have an in-degree and/or out-degree value of 1.

For each synthetic workflow size, we generated 5 different DAGs. For each task within a workflow, we assigned a random task computation time between 300-900 seconds. We also need to observe the effect of overall communication-to-computation ratio (CCR) of a workflow on the performance values. For this reason, we simulated the execution of each workflow for CCR values of 0.1, 0.5, 1, and 5. To achieve this, each communication link going out from a task was given a cost value proportional to the computation cost value of the task and the target CCR value.

To keep the simulation environment controllable and to be able to isolate our assessment from myriad factors (e.g., middleware/software overhead, communication failures, hardware/software failures, etc.), following assumptions were made during the simulations of both synthetic and real-world workflows:

- Resources reported available by each domain remains available throughout the execution of the workflow.
- Every domain schedules tasks within its domain in first-come first-served (FCFS) manner.

- Communication cost between two tasks that were mapped on the same domain is ignored (assuming that network file system (NFS) is employed within each domain).
- All the resources within a domain and across different domains have identical hardware/software capabilities and background loads.

3.2.2 Synthetic Workflows Simulation Results

In this section, we present and analyze the results from various simulation runs. The execution of synthetic workflows on a single domain was simulated as the baseline. All the results given in this section are the average values obtained from the execution of 5 randomly generated workflows for each workflow size.

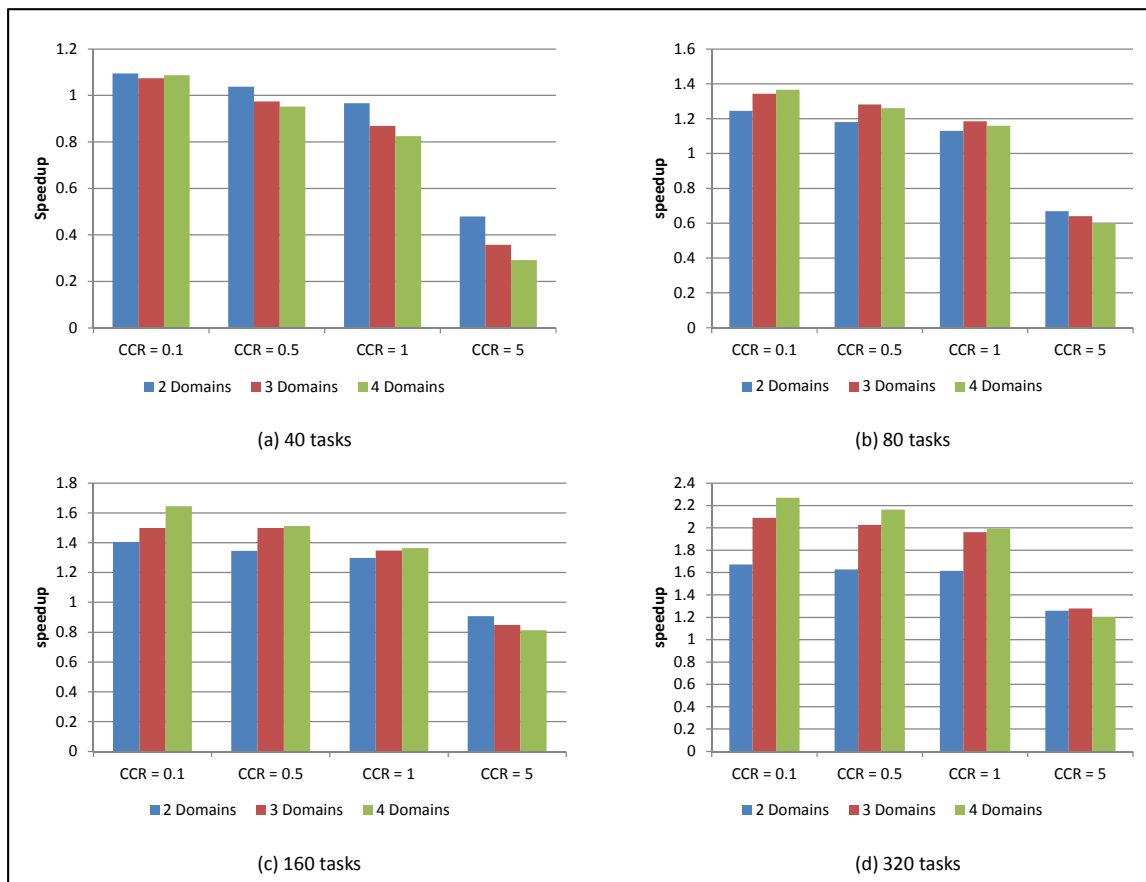


Figure 3.1: Speedup values simulating multiple domain executions of synthetic workflows

In the first set of experiments, we measure the speedup values obtained for multiple domain mapping/execution versus single domain mapping/execution of workflows. For this set of experiments, the number of CPUs available in each domain is set to be 8. According to this, in the case of 4 domains, there are a total of 32 CPUs available across domains for mapping/execution of a workflow. Fig. 3.1(a) shows the speedup values obtained for the workflow sizes of 40 tasks. A speedup value of more than 1 is achieved only when the CCR value is 0.1 for all multiple domain executions. For CCR value of 0.5, only the 2 domain execution achieves a speedup value of slightly larger than 1. The maximum speedup obtained for the set of workflows simulated at this workflow size is around 1.1 across all metrics (number of domains, CCR values). This result suggests that for workflows with size of around 40 tasks and under the assumed conditions, there isn't a significant performance improvement to be gained by mapping a workflow on multiple domains.

As the size of workflows get larger, a substantial increase in all speedup values is observed. In Fig. 3.1(b), all the speedup values for multiple domain executions with CCR value of 0.1, 0.5, and 1 are larger than 1. 4-domain execution outperforms all others when CCR value is 0.1; however its performance degrades more than others relatively as the CCR value increases. In both Fig. 3.1 (c) and Fig. 3.1 (d), speedup values obtained for 3-domain executions are higher than the 2-domain executions (except when CCR value is 5), and similarly speedup values for 4-domain executions are higher than the 3-domain executions. These results suggest that as the sizes of workflows get larger, mapping the workflow on more domains is likely to provide better performance results under these conditions. Another observation to note here is that, the speedup values obtained for all

multiple domain executions are less than 1 when CCR value is 5, except when the workflow size is the largest. This behavior indicates that, a communication-intensive workflow should carefully be analyzed before deciding to map/execute the workflow on multiple domains. A communication-intensive workflow with high-degree of parallelism may still be suitable for multiple domain executions.

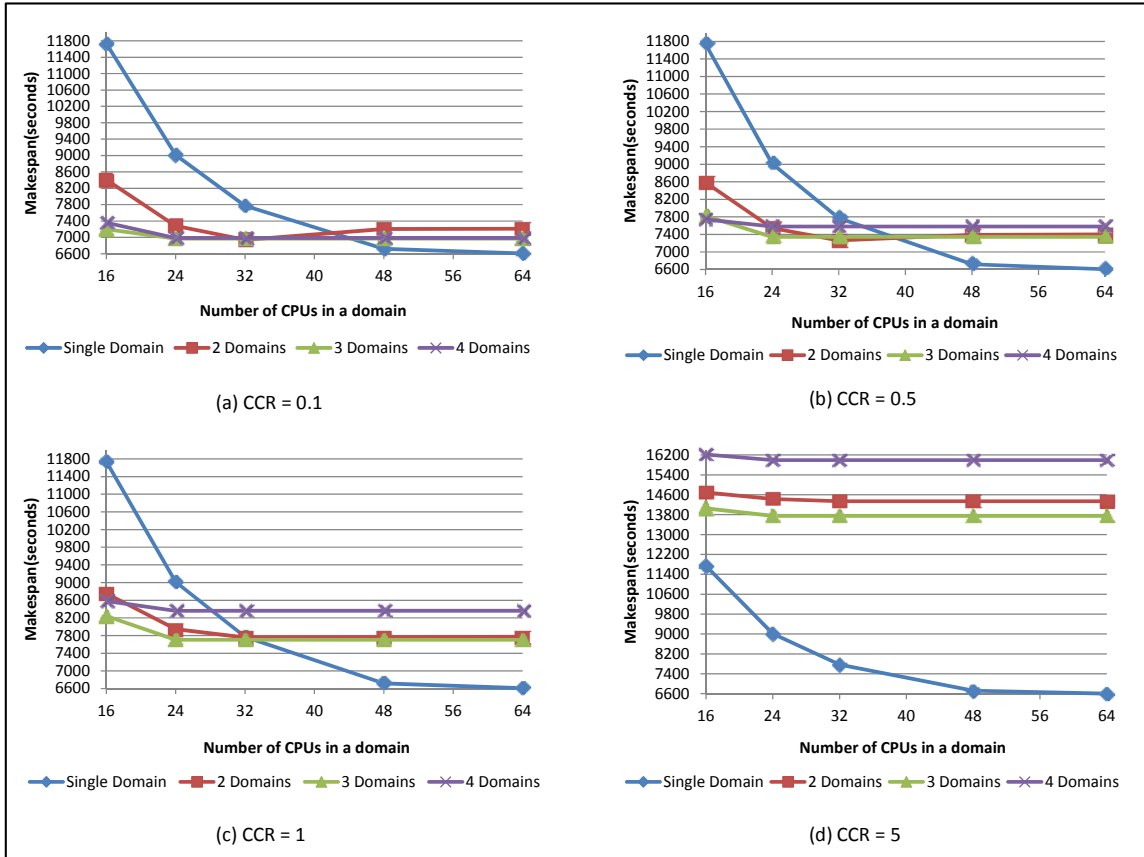


Figure 3.2: Average makespan values for the synthetic workflows with 320 tasks

In the next set of experiments, we provide the makespan trend of workflows as the number of available CPUs is increased in each domain. For this study, synthetic workflows with only the size of 320 tasks are used. Fig. 3.2 displays the average makespan values for single and multiple domain executions across different CCR values of the synthetic workflows. First thing to note here is that the single domain makespan

shows a logarithmic behavior as the number of CPUs increases and then it reaches its limit around 64 CPUs. It is also observed that the makespan values for all multiple domain executions remain flat beyond 32 CPUs in each domain. These results provide us with guidelines in terms of mapping/executing workflows on single or multiple domains of resources based on the number of resources available for the utilization of the workflow. Fig. 3.2 illustrates also the effects of the CCR characteristics of the workflow on the makespan values for multiple domain executions, which is significantly adverse when the CCR value is 5.

3.2.3 Real-world Workflows Simulation Results

In this section, we present the results we gathered from the simulation of two real-world scientific workflows. These workflows namely are the Montage [4] and LIGO Inspiral Analysis [105] workflows. We have used the execution profile results from [106] to map these workflows into our simulation environment.

Based on the execution profile results [106], the proper data dependency characteristics among tasks were included in the workflow specifications to be taken into account by the simulation environment. For the data transfer rate between different domains, we have used a fixed 1 MB/s bandwidth rate (based on the data transfer test results among different XSEDE [3] sites at the time these simulations were conducted). Similarly, for each data transfer between different domains, 1 second of communication and software overhead has been introduced as a parameter to the simulation. Each domain was configured to have a total of 2 computational nodes to execute tasks submitted to their domain.

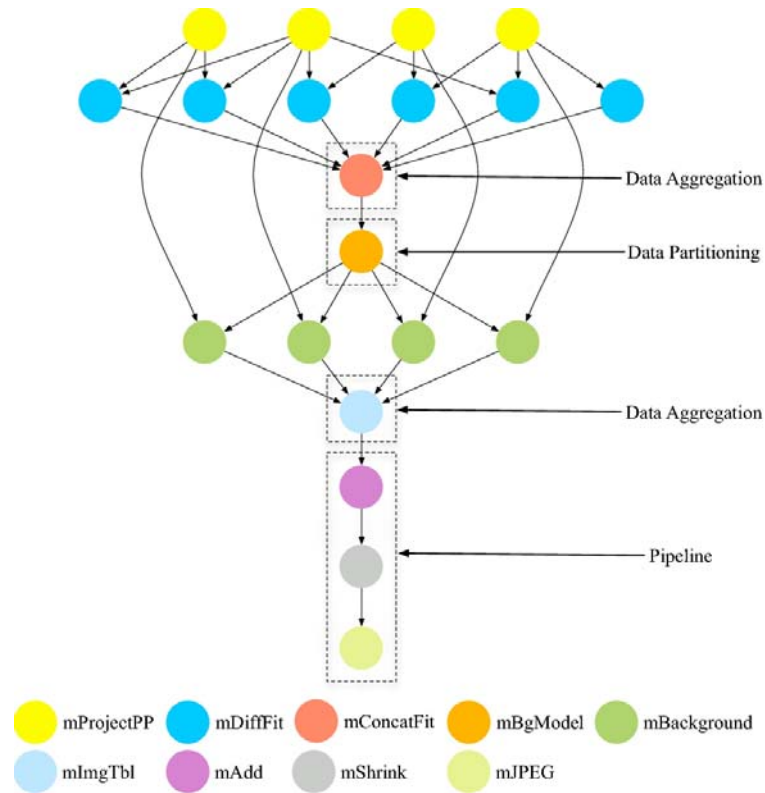


Figure 3.3: DAG structure and task types of a Montage workflow [106]

The basic DAG structure of a Montage workflow with 20 tasks is shown in Fig. 3.3. The number of inputs processed by a Montage workflow may change based on the scale of the study. Accordingly, structure of the workflow is adjusted to accommodate the processing of inputs. Thus, a specific Montage workflow instance may encompass varying number of computational tasks. In our simulation studies, we utilize two different instances of the Montage workflow. First Montage workflow instance is comprised of a total of 50 computational tasks, and the second Montage workflow instance is comprised of a total of 100 computational tasks.

Fig. 3.4 shows the speedup results we gathered from simulating the execution of Montage workflow on multiple domains. We observe that the multiple domain mapping/execution of the Montage workflow instance with 50 tasks yields around a

speedup value of around 1.2 across all three scenarios. On the other hand, the Montage workflow instance with 100 tasks yields significant and steady speedup results as more domains are utilized for the mapping/execution of the workflow.

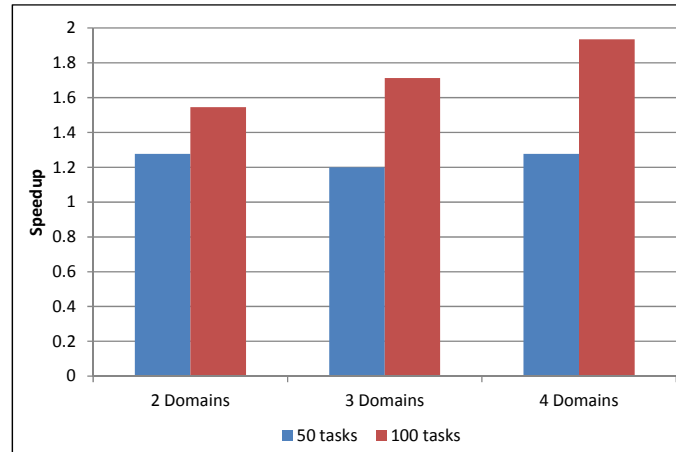


Figure 3.4: Speedup values simulating the multiple domain executions of the Montage workflow

The simplified DAG structure of a LIGO Inspiral Analysis workflow is shown in Fig. 3.5. The number of inputs processed by a LIGO Inspiral Analysis workflow may change based on the amount of data retrieved to be processed from LIGO (Laser Interferometer Gravitational Wave Observatory) detectors. Accordingly, the actual number of computational tasks in a LIGO Inspiral Analysis workflow varies. In our simulation studies, we utilize two different sizes of the LIGO Inspiral Analysis workflow. First workflow instance is comprised of a total of 50 computational tasks, and the second workflow instance is comprised of a total of 100 computational tasks. As can be seen from their DAG structures, LIGO Inspiral Analysis workflow has a much higher degree of parallelism than Montage workflow. Also, referring to their execution profiles, LIGO Inspiral Analysis workflow is found to be more computationally intensive than the Montage workflow.

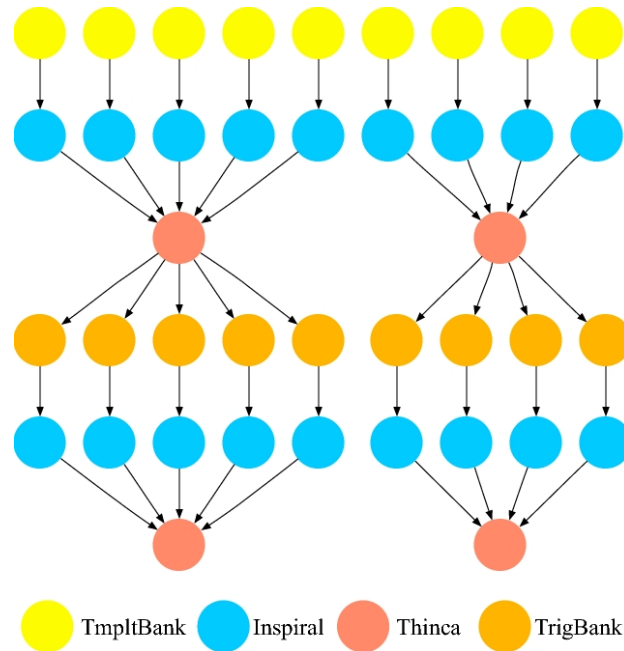


Figure 3.5: DAG structure and task types of a LIGO Inspiral Analysis workflow [106]

Fig. 3.6 shows the speedup results we gathered from simulating the execution of LIGO Inspiral Analysis workflow on multiple domains. We observe that the speedup values gained with the simulations of LIGO Inspiral Analysis workflow instances is noticeably better than the ones we gained with simulating the Montage workflow instances, except for the workflow instance with 50 tasks executed on 2 domains. Beyond the mapping on 2 domains, significant and steady speedup results are gained for multiple domain executions of the workflow instance with 50 tasks. On the other hand, the LIGO Inspiral Analysis workflow instance with 100 tasks yields near linear speedup values when it is mapped on 2 domains or 3 domains. However, when the same workflow instance is mapped on 4 domains - even though it still yields a significant speedup value of around 2 - the speedup value measured is significantly worse than that of the 3 domains execution case and almost at the same level as that of the 2 domains execution case. We detected that the reason for such behavior is the non-optimal mapping produced

by the graph partitioning tool due to insufficient adaptation of its mechanisms to the specific nature of the application. We argue that, with a proper mapping of this workflow instance, a higher speedup value can be obtained by employing 4 domains of resources than that of the 3 domains of resources.

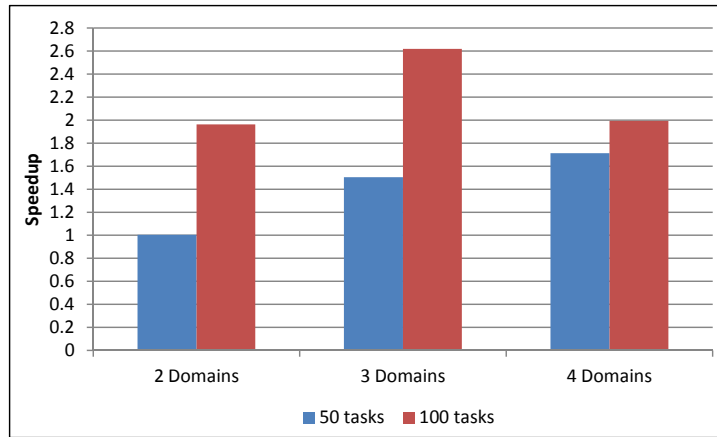


Figure 3.6: Speedup values simulating the multiple domain executions of the LIGO Inspiral Analysis workflow

3.3 DESIGN OF THE DECENTRALIZATION FRAMEWORK

Our decentralization framework [51] utilizes the common DAG patterns to transform the concrete workflow at each site. Following the transformation process, the whole workflow is ready to be orchestrated in decentralized fashion by local workflow managers at each site. Fig. 3.7 illustrates these stages that the original DAG specification goes through to result in multiple decentralized workflow processes.

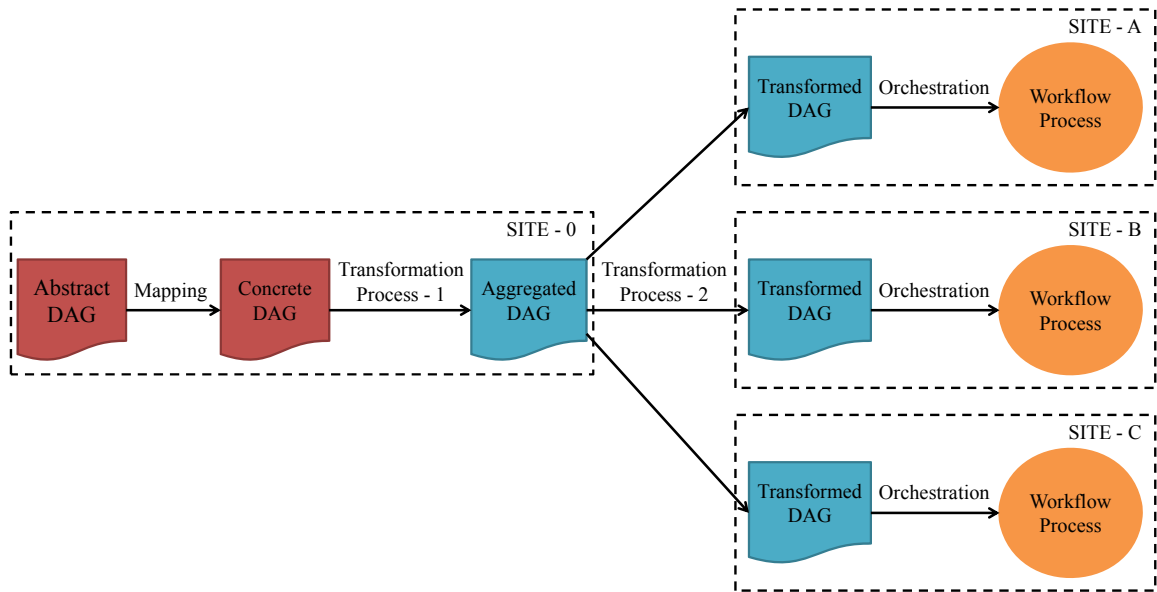


Figure 3.7: DAG specification stages according to our decentralized orchestration framework

We first introduce the DAG patterns, and then we explain the patterns necessary for the transformation process performed at each site.

3.3.1 DAG Patterns

DAG patterns form the basic building blocks for the transformation process. The key point here is that all possible scientific workflows in DAG form can be represented using a proper combination of these DAG patterns.

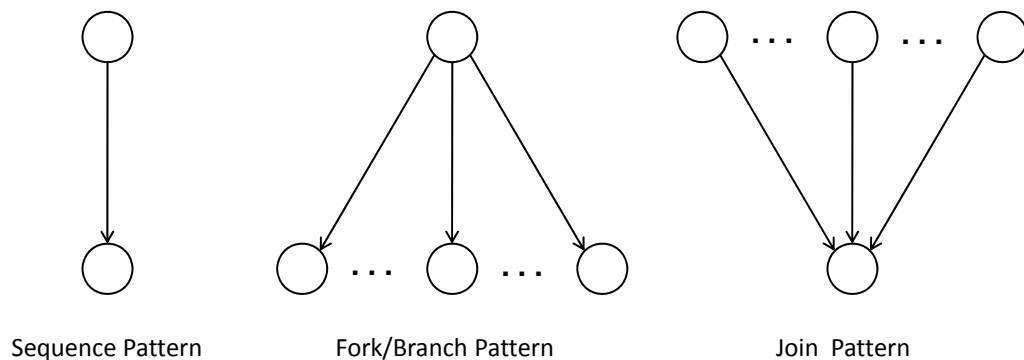


Figure 3.8: DAG patterns

Fig. 3.8 illustrates three DAG patterns, namely: Sequence pattern, Fork/Branch pattern and Join pattern. In these graphs, vertices correspond to the computational tasks, whereas directed edges correspond to the control and/or data dependencies between tasks.

3.3.2 DAG Transformation Patterns

We propose a systematic and generic framework for transforming the centralized orchestration to a collaborative orchestration via the utilization of DAG transformation patterns. These patterns are built on basic DAG patterns introduced in Section 3.3.1, and illustrate the transformations on the original DAG specifications to meet the needs of the collaborative orchestration style. Each local workflow manager individually performs these transformations at its local site, based on the mapping decisions during the concretization process.

- Transformations for the Sequence DAG pattern: Fig. 3.9 illustrates the set of DAG transformations on the Sequence DAG pattern corresponding to four possible mapping scenarios. If both tasks comprising the Sequence DAG pattern are mapped locally, as in Fig. 3.9(a), then no transformation is necessary. If both tasks comprising the Sequence DAG pattern are mapped remotely, as in Fig. 3.9(b), then these tasks are marked to indicate that they will be orchestrated by another manager. Fig. 3.9(c) illustrates the case where the parent task is mapped locally, whereas the child task is mapped remotely. In this case, the transformation process incorporates a synchronization stub between these tasks. The responsibility of a synchronization stub is to provide a peer workflow manager with the necessary means to communicate and synchronize with other peers regarding the workflow execution progress. In this case, the synchronization stub is responsible with informing the

remote workflow manager of the completion of the parent task. Fig. 3.9(d) illustrates the case where the parent task is mapped remotely, and the child task is mapped locally. In this case, the transformation process again incorporates a synchronization stub between these tasks. However, in this case the local workflow manager waits to be informed by its remote partner about the completion of the parent task.

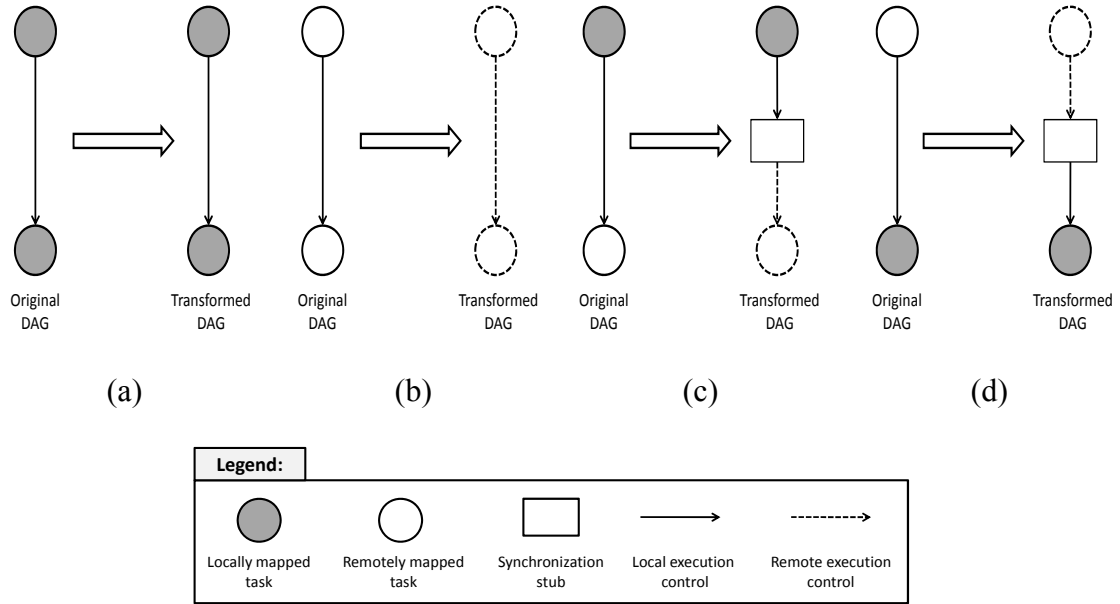


Figure 3.9: DAG transformations for the Sequence DAG pattern

- Transformations for the Fork/Branch DAG pattern: Fig. 3.10 illustrates the DAG transformations on the Fork/Branch DAG pattern corresponding to two possible mapping scenarios. We skip the two other possible mapping scenarios, in which all the set of tasks are either mapped locally or remotely, as the transformations will be very limited and done similar to the ones in Fig. 3.9(a) and Fig. 3.9(b). Fig. 3.10(a) illustrates the case where the parent task is mapped locally, whereas the children tasks are mapped remotely. In this case, the transformation incorporates a synchronization stub after the parent task. This synchronization stub is responsible for informing the

remote workflow execution manager(s) of the completion of the parent task. Fig. 3.10(b) is similar to the previous scenario, however this time parent task is mapped remotely and children tasks are mapped locally. Accordingly, the synchronization stub inserted in this scenario is responsible to wait for its remote partner to perform its corresponding synchronization stub activities.

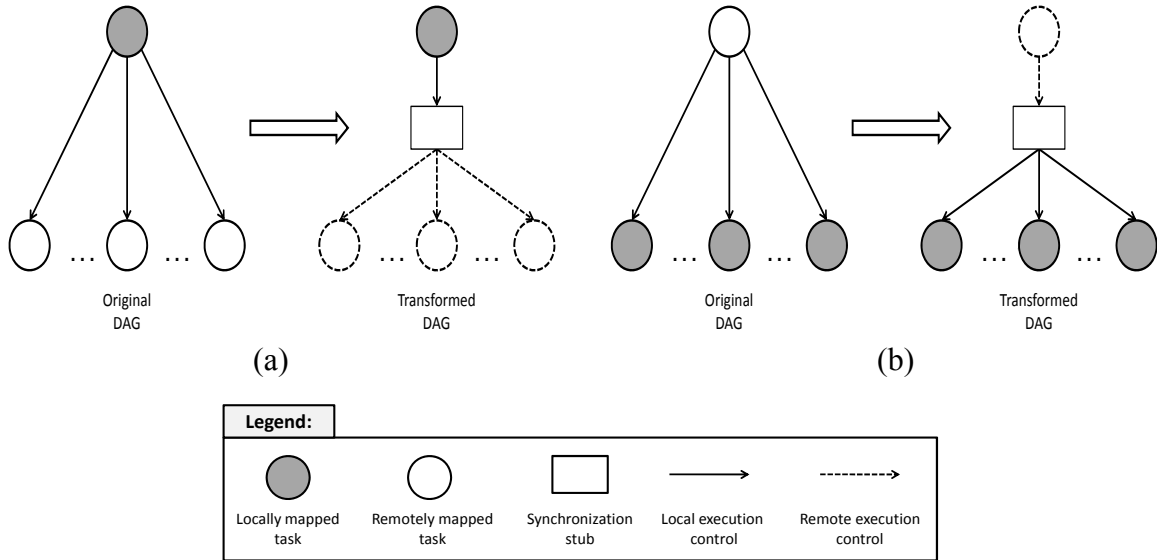


Figure 3.10: DAG transformations for the Fork/Branch DAG pattern

- Transformations for the Join DAG pattern: Fig. 3.11 illustrates the DAG transformations on the Join DAG pattern corresponding to two possible mapping scenarios. As in Fig. 3.10, we skip the two other possible mapping scenarios. Fig. 3.11(a) illustrates the case where the parent tasks are mapped locally, and the child task is mapped remotely. In this case, the transformation incorporates a single synchronization stub before the child task. This synchronization stub is responsible to inform the remote partner regarding the completion of the parent tasks. According to the scenario in Fig. 3.11(b), parent tasks are mapped remotely and the child task is mapped locally. The synchronization stub inserted in this scenario is responsible to

wait for its remote partner(s) to perform all of its (their) corresponding synchronization stub activities.

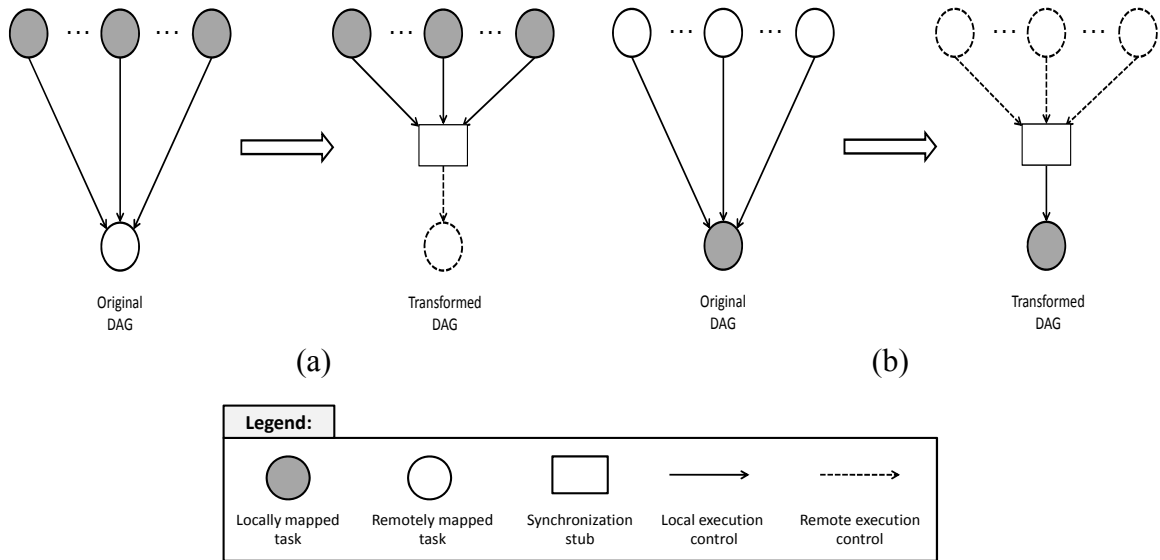


Figure 3.11: DAG transformations for the Join DAG pattern

One thing to note for the transformation of Fork/Branch and Join DAG patterns is that we incorporate a single synchronization stub among the parent and children tasks, regardless of the number of parent and children tasks in the original DAG. This design choice significantly reduces the number of synchronization activities (hence overhead) among peers, and also simplifies the transformation process.

3.4 PROTOTYPE IMPLEMENTATION

Our prototype implementation [52] is based on Condor DAGMan [11, 16] workflow execution engine. Condor DAGMan is a widely used workflow execution tool that is available in most High-Performance and High-Throughput Computing environments.

The original orchestration architecture of Condor DAGMan is centralized. Condor DAGMan specifies a DAG structure by listing the tasks and their dependencies in a

standard text file. This specification is parsed and then the execution of tasks is orchestrated by submitting them to local and/or remote resources by Condor DAGMan.

In our decentralized framework, each site has its own deployment of Condor DAGMan tool, and each engine submits computational tasks only to their local resources. Thus, the orchestration of the workflow is managed in collaboration with local Condor DAGMan workflow execution engines employed at each site.

Here, we explain the steps performed prior to the actual orchestration of the workflow. After these steps, orchestration of the workflow is accomplished in decentralized fashion as explained above.

3.4.1 Creation of the Aggregated DAG Specification

In the first stage, the original DAG specification is incorporated with some additional information so as to facilitate the transformation processes in the second stage. This is performed at the site where the DAG creation and mapping is originated. At this point, DAG specification has been concretized with actual physical resource information. However, before this original DAG specification is submitted to the local Condor DAGMan engine for orchestration, it is subjected to an aggregation process. The aggregated DAG specification then goes through the transformation process as explained in Section 3.3.2.

During this process, the original DAG specification is aggregated with mapping and site-specific contact information. Mapping information denotes the site that is responsible for each task's execution. Site-specific contact information includes the GRAM (Grid Resource Allocation and Management) end-point reference to facilitate

communication among sites, and the GridFTP [42] server and URL information to facilitate the transfer of data artifacts among sites.

This aggregated DAG specification is then distributed to all the sites that take part in the execution of the workflow. Upon receiving the aggregated DAG specification, workflow managers at each site proceed to the transformation process of the DAG specification.

3.4.2 Transformation Process for the Aggregated DAG Specification

The second stage of decentralization process is performed at each site upon receiving the aggregated DAG specification. Each workflow manager transforms its own copy of the DAG specification based on the aggregated DAG specification. There are 4 basic activities that need to be performed during the transformation process:

- Remotely mapped tasks are labeled as DONE. This way, local Condor DAGMan engine will not attempt to submit these tasks to local resources.
- DAG transformations illustrated in Fig. 3.9(c) and Fig. 3.10(a) is accomplished via the implementation of the synchronization stub as a POST script accompanying the parent task. A POST script is a standard mechanism found in Condor DAGMan to perform any kind of activity following the completion of a task. In our implementation, this POST script is populated with proper code informing the remote partner(s) about the completion of the parent task.
- DAG transformations illustrated in Fig. 3.9(d), Fig. 3.10(b), and Fig. 3.11(b) are accomplished via the implementation of the synchronization stub as a PRE script accompanying the child(ren) task(s). A PRE script is also a standard

mechanism found in Condor DAGMan to perform any kind of activity prior to execution of a task. In our implementation, this PRE script is populated with proper code that holds the execution of the main task until the completion of the parent task is conveyed by the remote partner(s).

- DAG transformation for the Join pattern illustrated in Fig. 3.11(a) is accomplished via the implementation of the synchronization stub as a single light-weight sync task followed with a POST script in the original DAG specification. The reason behind this implementation choice is due to multiple numbers of tasks being the parent of the synchronization stub. Rather than inserting a POST script after each parent task, insertion of a single light-weight DAG task (sync task) serves as the local synchronization point, which is then followed with the actual synchronization activity through the usage of the POST script.

Our POST script prototype simply generates a dummy file that is named specifically to uniquely identify the proper synchronization stub. Then, this file is transferred to the remote workflow manager site according to the information included in the aggregated DAG specification. The PRE script on the other hand simply expects to receive the proper file associated with the synchronization stub from the remote site.

After each site completes this process and creates its own copy of the transformed DAG specification, the transformed DAG specification is submitted to the local Condor DAGMan workflow execution engine. This way, as each transformed DAG specification is orchestrated by peer Condor DAGMan engines, collaboratively, the orchestration of the whole workflow is accomplished.

3.4.3 Case Scenario for the Implementation of Decentralized Orchestration

In this case scenario, we demonstrate the transformation process of our decentralization framework and provide the corresponding DAG specifications for a sample workflow, based on a sample mapping scenario. Fig. 3.12 provides a sample workflow, with a sample mapping scenario, where the mapping of workflow tasks span across two sites. Fig. 3.13 presents a basic DAG specification for this workflow that can be orchestrated via standard centralized Condor DAGMan workflow engine.

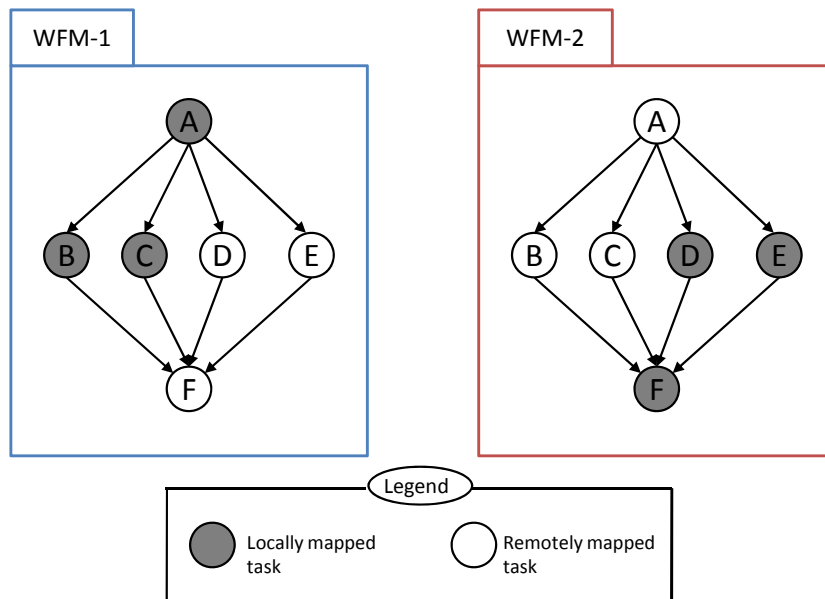


Figure 3.12: Sample workflow mapped on two sites

Job	A	A.submit		
Job	B	B.submit		
Job	C	C.submit		
Job	D	D.submit		
Job	E	E.submit		
Job	F	F.submit		
PARENT	A		CHILD	B C D E
PARENT	B C D E		CHILD	F

Figure 3.13: Basic Condor DAGMan specification for the workflow in Fig. 3.12

Based on the mapping scenario as illustrated in Fig. 3.12, the transformation process of our decentralization framework creates a corresponding transformed DAG specification at each site. Fig. 3.14 illustrates the resulting DAG structure at site WFM-1 after the transformation process. Fig. 3.15 illustrates the corresponding DAG specification for the transformed DAG, which will be orchestrated by the local Condor DAGMan engine at site WFM-1.

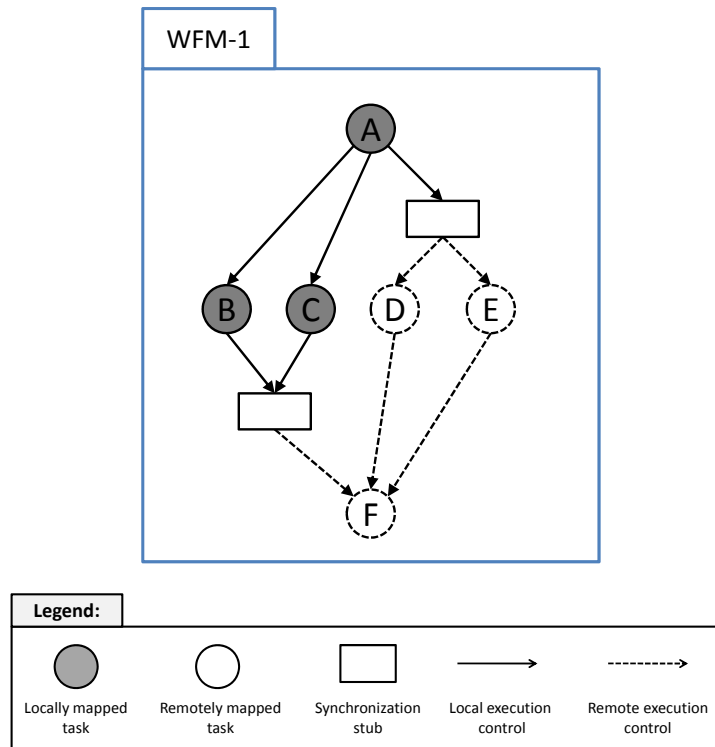


Figure 3.14: Transformed DAG structure at site WFM-1 after the transformation process

Job	A	A.submit			
Job	B	B.submit			
Job	C	C.submit			
Job	B_C	B_C.submit			
Job	D	D.submit	DONE		
Job	E	E.submit	DONE		
Job	F	F.submit	DONE		
SCRIPT	POST	A	sync.sh	A_D	A_E
SCRIPT	POST	B_C	sync.sh	B_C	
PARENT	A	CHILD	B C D E		
PARENT	B C	CHILD	B_C		
PARENT	B_C D E	CHILD	F		

Figure 3.15: Transformed DAG specification to be orchestrated at site WFM-1

Similarly, based on the mapping scenario as illustrated in Fig. 3.12, Fig. 3.16 illustrates the resulting DAG structure at site WFM-2 after the transformation process. Fig. 3.17 illustrates the corresponding DAG specification for the transformed DAG, which will be orchestrated by the local Condor DAGMan engine at site WFM-2.

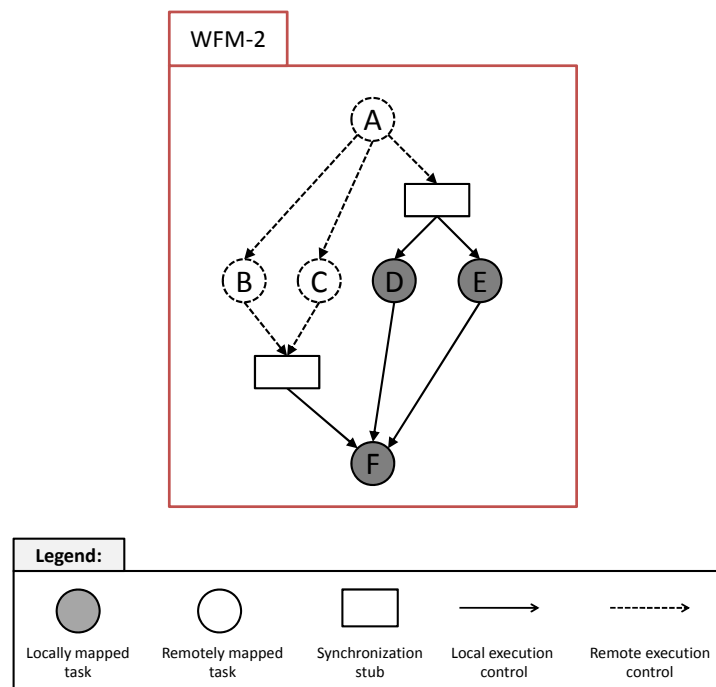


Figure 3.16: Transformed DAG structure at site WFM-2 after the transformation process

Job	A	A.submit	DONE		
Job	B	B.submit	DONE		
Job	C	C.submit	DONE		
Job	D	D.submit			
Job	E	E.submit			
Job	F	F.submit			
SCRIPT	PRE	D	sync.sh	A_D	
SCRIPT	PRE	E	sync.sh	A_E	
SCRIPT	PRE	F	sync.sh	B_C	
PARENT	A	CHILD	B C D E		
PARENT	B C D E	CHILD	F		

Figure 3.17: Transformed DAG specification to be orchestrated at site WFM-2

3.5 PERFORMANCE EVALUATION

To evaluate the performance of our prototype implementation based on our generic decentralization framework for workflow orchestration, we conducted several experiments on a real distributed computing environment. We have conducted our experiments using one synthetic workflow and one real-world workflow (i.e. Montage Astronomy workflow [4]). For each workflow type, we have used two different sizes of workflow instances. We map each abstract workflow instance on domains of resources and create the concrete workflow instance. Then, each concrete workflow instance is executed using both the centralized orchestration and the decentralized orchestration. We measure the makespan value for the execution of each workflow instance. We repeat this process 5 times and calculate the average makespan value for the centralized orchestration and decentralized orchestration of each workflow instance. Based on these

makespan values, we calculate and report the speedup values obtained through the utilization of our decentralization framework.

3.5.1 Experimental Setup

We conducted our experiments on a real multi-domain computational infrastructure, XSEDE [3], which is a collaboration of specialized high-performance and high-throughput computational and storage systems across the United States of America, which is made available through a National Science Foundation (NSF) project. These computational and storage resources within the XSEDE project are owned and administered by different organizations. The authentication of users, communication interactions, and access to resources among sites are made possible mainly through the Globus middleware [25, 26]. Various other software tools and services are used for more specialized purposes. We have utilized the computational resources of three specific systems within the XSEDE infrastructure, namely the NCSA Abe cluster, Purdue Condor pool, and LONI Queenbee cluster.

The following is the brief information regarding the properties of resources at the time our experiments were conducted. Abe cluster is located at National Center for Supercomputing Applications (NCSA), Urbana, Illinois, and provides a total of 9600 CPU cores. Purdue Condor pool is located at Purdue University, West Lafayette, Indiana, and provides over 14000 CPU cores (the actual number varies since idle resources are scavenged into the system over time). Queenbee cluster is located at Louisiana Optical Network Initiative (LONI), Baton Rouge, Louisiana, and provides a total of 5344 CPU cores. All these resources are shared among many users and the scheduling of computational jobs is managed through individual batch management software at each

domain. Each system has Network File System (NFS) support to provide uniform access to data files across all local compute nodes. The main tool that is used to transfer data across these domains is GridFTP [42].

Each system, by default, provides a local deployment of Condor [23], Condor DAGMan [16], Condor-G [24], and Pegasus [18] software tools that are used for the various phases of our experiments. In our experiments, for the centralized orchestration of a workflow, the Condor DAGMan workflow engine that is deployed at the Purdue Condor pool is used. There has been made no changes to the existing coding of these tools throughout the implementation and conducting of our experiments.

Pegasus [18] is used to map abstract workflow instances and generate the corresponding concrete workflow instances. In our experiments, we have performed the mapping of workflow instances on both 2 domains and 3 domains, and run them separately to measure the performance variation. For the mapping of a workflow instance on 2 domains, we used the Purdue Condor pool domain and the NCSA Abe domain. The DAG transformations on a concrete workflow instance required for our decentralization framework is performed manually.

To prevent the effects of variance in the availability of computational resources on the performance results, we monitored and controlled the amount of resources utilized throughout the experiments. First, in each domain, we made changes to the local job scheduler configuration, so that maximum 4 computational jobs (translates to workflow tasks in our context) are allowed to execute at any given time. We also monitored the time each workflow task spends waiting in the local job scheduler queue. If a workflow task spends more than 1 minute waiting in the local queue, the result obtained from the

execution of that workflow instance is discarded.

3.5.2 Synthetic Workflow Evaluation Results

The synthetic workflow in our experiments almost serves as a yardstick evaluating the performance expectations originating from the workflow orchestration process. Accordingly, our synthetic workflow has a simple diamond-shaped DAG structure, similar to that of Fig. 3.12. Such a DAG structure makes the workflow application to have a degree of parallelism that is near maximum. Also, due to the same DAG structure, peer workflow managers found in the decentralized orchestration framework can operate almost completely in parallel with minimum level of interactions required among them.

To be able to expose and emphasize the performance characteristics of workflow orchestration strategies, the computational costs associated with workflow tasks and data dependencies among them were kept at relatively minimum levels. Accordingly, a random computational time of between 50-200 seconds (implemented through basic ‘sleep’ command in Linux) were assigned to each task. Similarly, the sizes of data files required to be transferred among dependent tasks were assigned a random size between 1 Megabyte and 50 Megabytes.

We generated two synthetic workflows complying with these characteristics. First workflow has a total of 40 computational tasks whereas the second workflow has a total of 80 computational tasks. Both of these workflows were mapped and then executed within our experimental environment as explained before. Based on the makespan values generated from centralized and decentralized orchestration of the workflows, speedup values comparing both strategies are presented in Fig. 3.18. The speedup values gained

through decentralization of the orchestration of these workflows range between 1.23 and 1.27 within the confines of our experimental environment and conditions. We can observe that the performance gain through the decentralized orchestration is higher with the larger size of the synthetic workflow. We can also observe that the performance gain attained through decentralized orchestration is slightly higher when the mapping of workflow spans across 3 domains rather than just 2 domains.

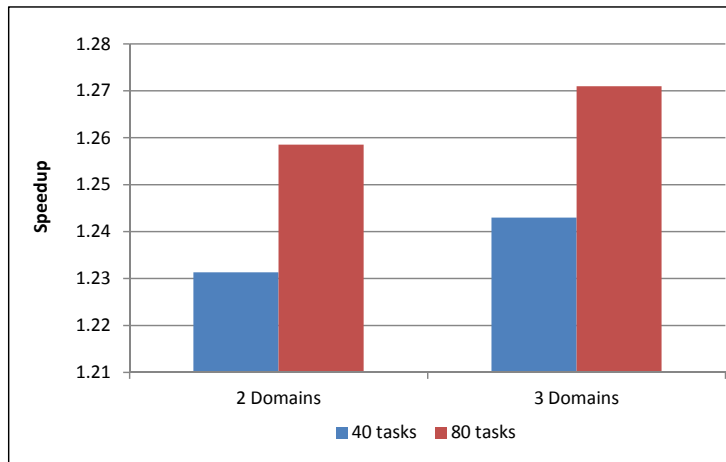


Figure 3.18: Speedup values obtained for decentralized versus centralized orchestration of the synthetic workflow

3.5.2 Montage Workflow Evaluation Results

We conducted our performance evaluation studies comparing the centralized orchestration and decentralized orchestration strategies via experimenting with the Montage [4] Astronomy workflow. Montage has a basic DAG structure as shown in Fig. 3.3 and the detailed execution profile of Montage workflow tasks and dependencies is available [106]. In our experiments, we have used a Montage workflow instance consisting of 50 tasks and another Montage workflow instance consisting of 100 tasks. Both of these workflow instances were mapped and then executed within our experimental environment as explained before.

Based on the makespan values generated from centralized and decentralized orchestration of the workflows, speedup values comparing both strategies are presented in Fig. 3.19. The speedup values gained through decentralization of the orchestration of these workflows range between 1.08 and 1.11. First thing to note with these results is that, the speedup values obtained for the orchestration of the Montage workflows are substantially lower than those obtained for the orchestration of the synthetic workflows. This is simply due to the DAG structure and the computational requirements of the Montage workflow. Montage workflow has a less degree of parallelism compared to the synthetic workflow, which results in more interactions being performed among peer workflow managers during decentralized orchestration. Also, the computational cost associated with Montage workflow tasks is much larger than that of the synthetic workflow. Due to this characteristic, workflow orchestration overhead has less impact in the overall makespan value of the Montage workflow compared with the synthetic workflow.

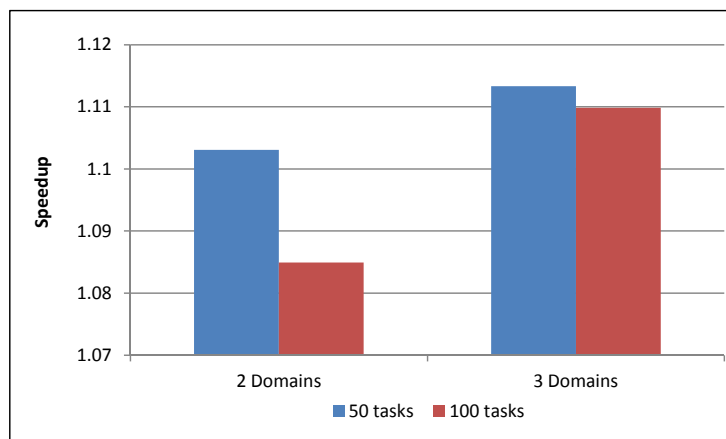


Figure 3.19: Speedup values obtained for decentralized versus centralized orchestration of the Montage workflow

Comparing various speedup values in Fig. 3.19, we observe that the performance gain attained through decentralized orchestration of the Montage workflow is slightly worse for the instance with 100 tasks compared to that of the instance with 50 tasks. We also observe that, similar to the synthetic workflows, the performance gain attained through the decentralized orchestration of the Montage workflow is slightly higher when the mapping of workflow spans across 3 domains rather than just 2 domains.

CHAPTER 4

RUN-TIME ADAPTATION FOR WORKFLOW ORCHESTRATION

In this chapter, we propose mechanisms to provide efficient adaptation capabilities during the decentralized orchestration of workflows. Thus, the designs and mechanisms provided here builds upon the decentralization framework that were explained in Chapter 3.

First, we explain the motivating issues that demand run-time adaptation capabilities during workflow orchestration. Then, we explain the extent of activities that are entailed with providing such capabilities. Later, we explain the system design and mechanisms we provide to carry out those activities within our decentralized workflow orchestration framework. Then, we provide some details regarding the prototype implementation for our adaptation mechanisms. Finally, we provide the performance evaluation results obtained via the utilization of our adaptation mechanisms during the decentralized orchestration of both a synthetic and a real-world workflow, through our experiments conducted on an actual multi-site computational infrastructure (i.e. XSEDE [3]).

4.1 MOTIVATION

The main goal of the run-time adaptation process is to keep the makespan metric of the workflow execution within user expectations. There are two main reasons that may necessitate run-time adaptation during the orchestration of a workflow.

The first main reason that may necessitate run-time adaptation during the orchestration of a workflow is the change/dynamicity of the availability of computational resources for the utilization of workflow tasks at the run-time. The availability of these

resources may change basically due to hardware failures, increased workload, and higher priority tasks being deployed in the system. Especially long-running and large-scale workflows are highly susceptible to these kinds of changes in the execution environment.

The second main reason that may necessitate run-time adaptation during the orchestration of a workflow stems from inaccurate task runtime predictions made during the workflow mapping process. During the workflow mapping process, making use of historical data or domain expertise knowledge, runtime predictions are made for each task in the workflow. However, a large number of factors may cause variations - insignificant to dramatic - in the accuracy of these predictions. These factors range from the inaccuracy/unpredictability issues caused by various hardware/software configuration metrics to code-level unpredictability issues such as the utilization of random numbers in determining the number of compute intensive iterations.

Resource availability metrics and task runtime predictions are the two main criteria while making decisions during the workflow mapping process. Thus, variations in the accuracy of these criteria at the run-time may cause inefficient execution of individual tasks followed by inefficient progress over the execution of the whole workflow. Consequently, these efficiency problems would negatively affect the makespan of the workflow.

For the following discussions we provide, we make the assumption that the workflow tasks span across multiple sites due to the decisions made during the original workflow mapping process. Also, here, we are not concerned with the adaptation/re-scheduling activities that may be performed within a single site. Our discussions and

mechanisms are based on the run-time adaptation activities that are performed across different sites.

4.2 RUN-TIME ADAPTATION PROCESS

We can categorize the activities that need to be performed for run-time adaptation process into three main phases.

- **Monitoring phase for the run-time adaptation:** This phase involves the continuous monitoring of workflow progress and the status of resources. During this, it is also checked whether a condition has arisen that necessitates an adaptation process.
- **Planning phase for the run-time adaptation:** After the detection, an appropriate corrective action has to be planned to cope with the condition. Some of these plans may only be concerned with some adjustments and/or rescheduling activities that are internal to the site. However, sometimes, the corrective actions may require re-mapping of a set of tasks among different sites. Set of tasks that needs to be re-mapped and the site(s) that will be involved in the adaptation process are identified in this phase.
- **Enactment phase for the run-time adaptation:** In this phase, adaptation plan proposed in the previous phase is enacted upon the ongoing workflow orchestration/execution process. Efficient and non-intrusive enactment of such a plan is as important as coming up with the adaptation plan itself.

4.3 SYSTEM DESIGN

Fig. 4.1 shows the main components for the system design of our decentralized run-time adaptation framework. Gray-colored components and artifacts denote those that are specifically concerned with the run-time adaptation. As our framework is based on

peer-to-peer communication and collaboration of workflow managers, each workflow manager is responsible for detecting any problems in their own domain that may prevent QS objectives from being met. In such a case, a set of tasks may then be migrated to other available domains for execution. The orchestration of the rest of the workflow is not affected by this adaptation. Only the peer workflow managers directly involved within this adaptation process need to perform necessary steps.

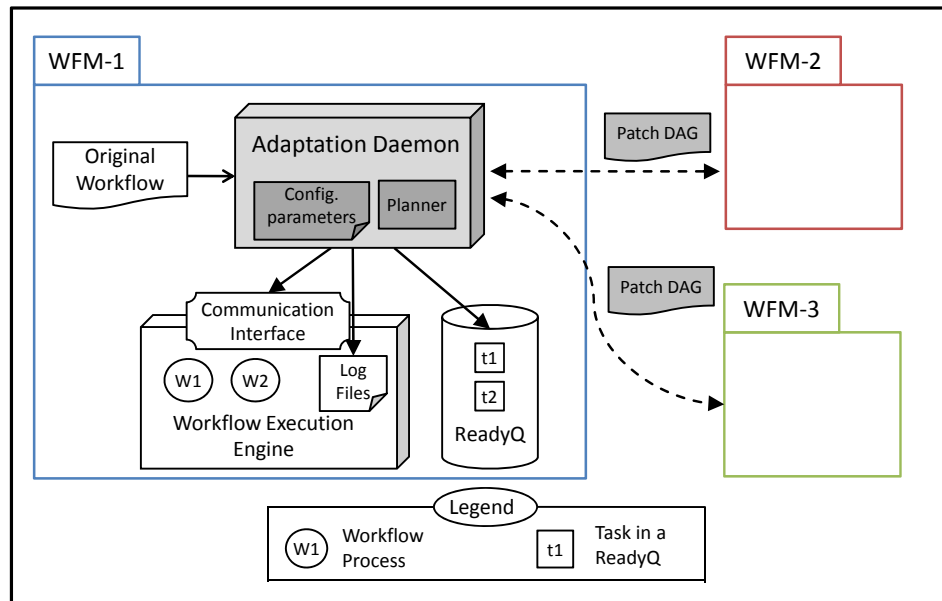


Figure 4.1: System design for the decentralized run-time adaptation framework

4.4 RUN-TIME ADAPTATION FRAMEWORK

Following the detection of a condition that necessitates an adaptation process, a corresponding adaptation plan to remedy the effects of unfavorable conditions is generated. A standard run-time adaptation plan basically involves making changes to the original mapping/execution site of a set of workflow tasks based on problems encountered during run-time. Modifications to the mapping site of tasks needs to be reflected and implemented accordingly by the workflow execution managers. Here, we

provide a run-time adaptation framework that integrates with our decentralized workflow orchestration framework discussed in Chapter 3. One key aspect of our framework is the low-level of intrusiveness to carry out the adaptation process. Our pattern-based framework has little effect on the ongoing workflow orchestration process. The peer workflow execution managers implement the re-mapping of tasks without any disruption to the orchestration of the rest of the whole workflow.

Please note that even though the contributions presented in Section 4.4.1 (Monitoring and Detection), and Section 4.4.2 (Planning) are published in (Kalayci, Dasgupta, Fong, Ezenwoye, & Sadjadi, 2010) [52], the main ideas introduced in these sections belong to Dr. Gargi Dasgupta. These sections are included in this dissertation because the mechanisms provided through these contributions complement those other mechanisms we present to make up our complete run-time adaptation framework.

4.4.1 Monitoring and Detection

Each workflow manager (WFM) independently monitors its resource queues for normal operations, as well as detection of problems and troubleshooting. A backed-up resource queue is almost always an indication of some underlying problem. We consider readyQ (Fig. 4.1) that queues all the tasks that are ready for execution at the WFM. Under normal operating conditions, tasks from the readyQ are steadily dispatched for execution to the underlying scheduler resources. A steadily building up readyQ signifies that there is some problem with the underlying infrastructure resources (e.g., resource outages, surge in background traffic, etc). We use readyQ-length as the indicator for the onset of congestion and proactively make runtime reconfigurations to deal with this.

To detect a congested queue, each WFM continually monitors its readyQ-length. For every readyQ, we assign two thresholds, low and high. The average queue length is calculated, using an exponential weighted moving average, given by:

$$\text{avg}_t = (1 - w) * \text{avg}_{t-1} + w * \text{qlen}_t,$$

where qlen_t is the length of readyQ at time t , and w takes values between $[0,1]$.

At time t , the following decisions are taken:

- If the queue length is below low, no corrective action is taken.
- If the queue length is between low and high, we probabilistically pick some tasks for remapping.
- If the queue length is above high, we pick all tasks for remapping.

4.4.2 Planning

A runtime reconfiguration in the original mapping involves the movement of yet-to-start computation or data management tasks to another domain. Our approach is based on RED [107], albeit accounting for data transfer involved in remapping the tasks. The probability, \hat{p}_j , of selecting a particular task j for moving depends on the average queue length, last time a task was moved from the queue, and also on data characteristics of the task.

Since the average length varies at a queue (belonging to site i) from low to high, the probability that a new compute task j is moved varies linearly from 0 to P (a small fraction). The final moving probability \hat{p}_j , however, also increases slowly with the number of tasks seen since last move (count), and decreases as the local data of j at i increases. We define the probability,

$$\hat{p}_j = \{p_b / (1 - \text{count} * p_b)\} * p_{ij}, \text{ where}$$

$$p_b = \{(\text{avg}_q - \text{low}_q)/(\text{high}_q - \text{low}_q)\} * P,$$

$$p_{ij} = (\Delta_{\max} - \Delta_{i,j})/(\Delta_{\max} - \Delta_{\min})$$

where Δ_{\max} , Δ_{\min} denote the maximum, minimum data requirements of j , respectively, and $\Delta_{i,j}$ is the size of local data present for j at site i .

4.4.3 Enactment

We utilize DAG adaptation patterns [51] at peer workflow managers to enact the re-mapping of tasks at run-time. Through this adaptation, the responsibility for the orchestration of the set of tasks being re-mapped (S_T) is transferred from the originating site to the destination site(s).

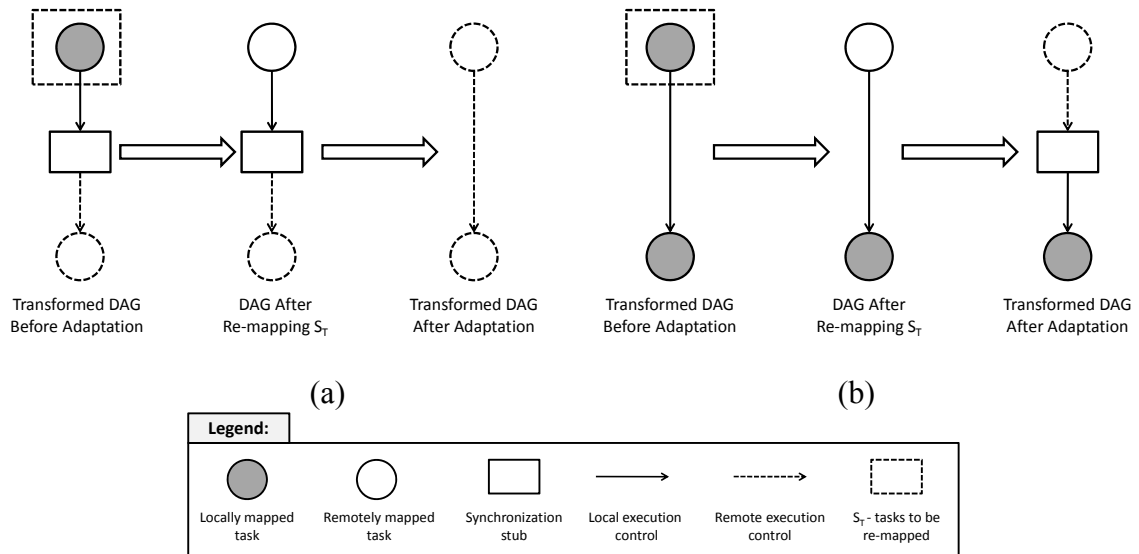


Figure 4.2: DAG adaptations for the Sequence DAG pattern

- DAG adaptation patterns at the originating site: At the originating site, DAG adaptation patterns transform S_T from being local tasks to remote tasks. Also, if there is a synchronization stub between S_T and child(ren) task(s), it is removed. Fig. 4.2(a) illustrates this scenario for the Sequence pattern, Fig. 4.3(a) illustrates the same scenario for the Fork/Branch pattern, and Fig. 4.4(a)

illustrates the same scenario for the Join pattern. However, if the child(ren) task(s) of S_T is (are) mapped locally, then a synchronization stub is incorporated between S_T and child(ren) task(s). Fig. 4.2(b) illustrates this scenario for the Sequence pattern, Fig. 4.3(b) illustrates the same scenario for the Fork/Branch pattern, and Fig. 4.4(b) illustrates the same scenario for the Join pattern.

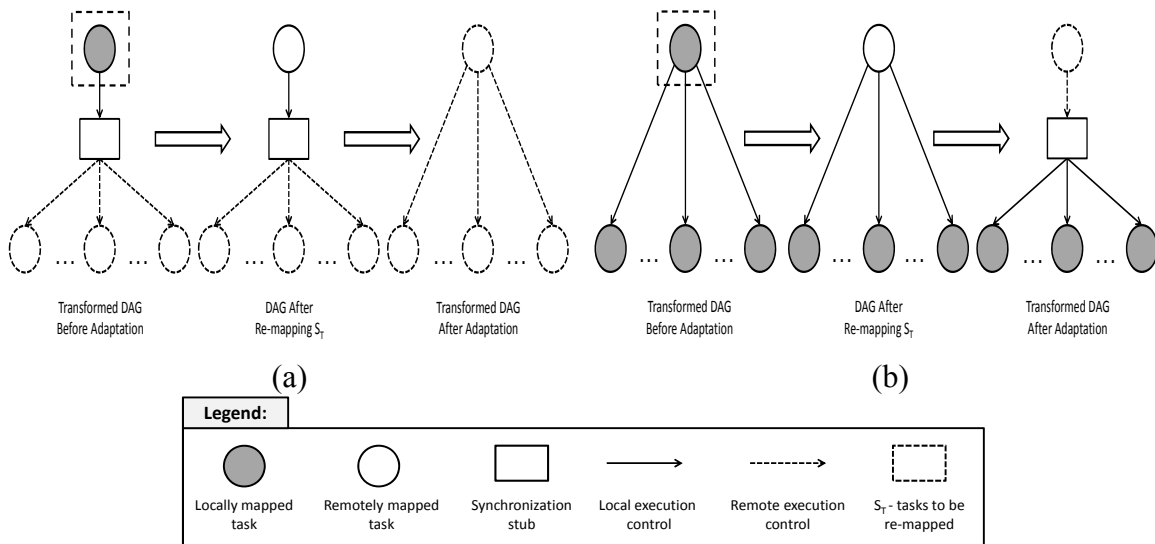


Figure 4.3: DAG adaptations for the Fork/Branch DAG pattern

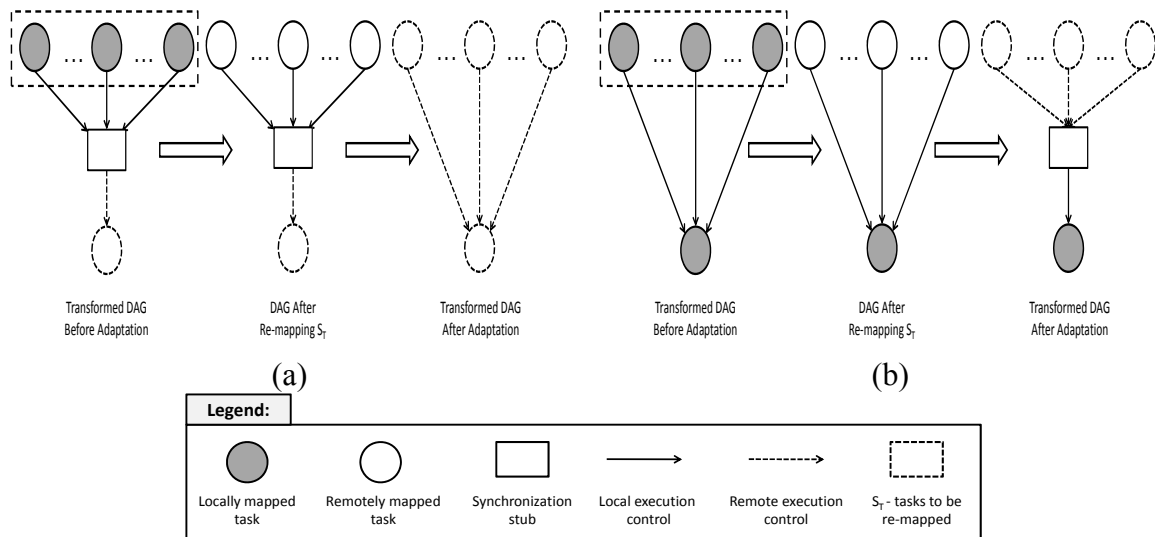


Figure 4.4: DAG adaptations for the Join DAG pattern

- DAG adaptation patterns at destination site(s): The destination site(s) basically captures the re-mapped tasks (S_T) and orchestrates them in accordance with the resulting DAG structure. However, attempting to capture and integrate this DAG structure with the transformed DAG specification at destination site(s) proves to be cumbersome to design and implement. For this reason, we propose the orchestration of these DAG structures in isolation from the transformed DAG structure(s) at destination site(s). In fact, we refer to these DAG structures that are generated through run-time adaptation processes as patch DAGs. These patch DAGs at destination site(s) are orchestrated in isolation. The combined orchestration of patch DAGs with the transformed DAGs provides the same business logic as the orchestration of the original complete DAG.

The essential duty of a destination site involved in the run-time adaptation process is to capture S_T and compose the corresponding patch DAG. Once the patch DAG is ready, destination site orchestrates the patch DAG in isolation.

Fig. 4.5 illustrates the corresponding patch DAGs for the adaptation of the Sequence patterns in Fig. 4.2. In both Fig. 4.5(a) and Fig. 4.5(b), the corresponding patch DAG has the same structure. The difference between them is the specific implementation of the synchronization stub. In Fig. 4.5(b), the synchronization stub informs the originating site regarding the completion of the parent task. On the other hand, synchronization stub in Fig. 4.5(a) informs a third site different from the originating site regarding the completion of the parent task. In fact, the third site involved in this process is

the one that is originally responsible for the execution of the child task, and it is unaware of these changes made at run-time.

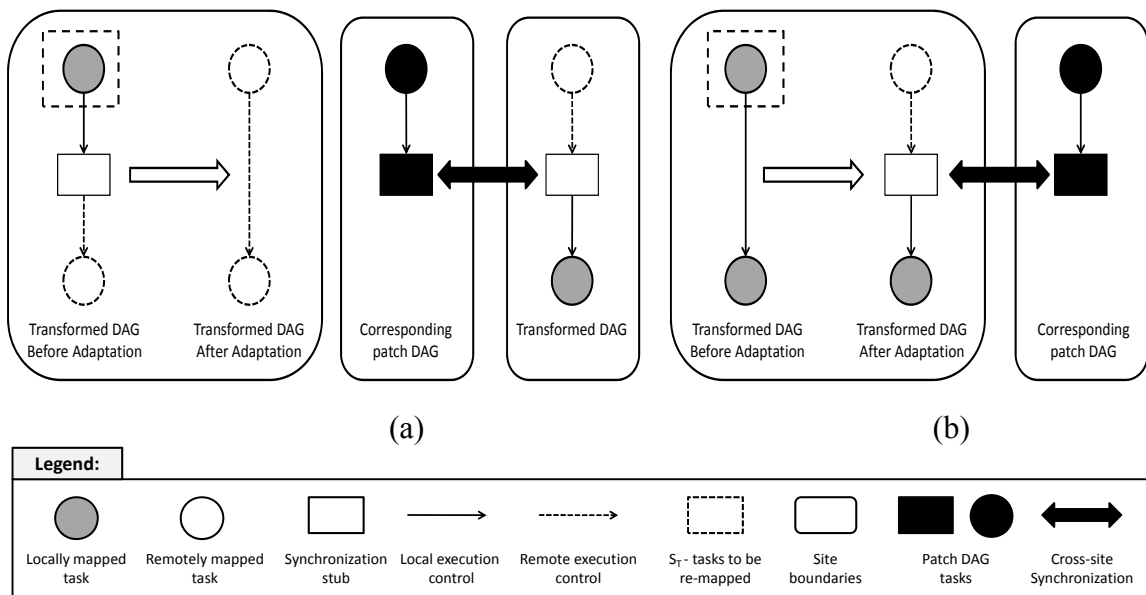


Figure 4.5: Patch DAG patterns corresponding to the adapted Sequence patterns

Patch DAGs corresponding to the adaptation of the Fork/Branch patterns look and operate the same way as explained for the adapted Sequence patterns. The only difference between these two patterns is the number of children tasks the patch DAG synchronization stub enables for progression.

The corresponding patch DAG structures to the adaptation of the Join pattern in Fig. 4.4(a) and Fig. 4.4(b) are also identical. As in the Sequence pattern, the difference between them is the specific implementation of the synchronization stub. However, in the Join pattern, it is possible for more than one site to be involved in the re-mapping of S_T , due to the multiplicity of the number of tasks in S_T . To illustrate this point, we provide two alternative scenarios for the patch DAG composition and operation corresponding to the

case in Fig. 4.4(b). Fig. 4.6 illustrates the corresponding patch DAG structure in a scenario where only one site ends up re-mapping all the tasks in S_T . For this scenario, the composition and the operation of the patch DAG is quite similar to the patterns explained earlier.

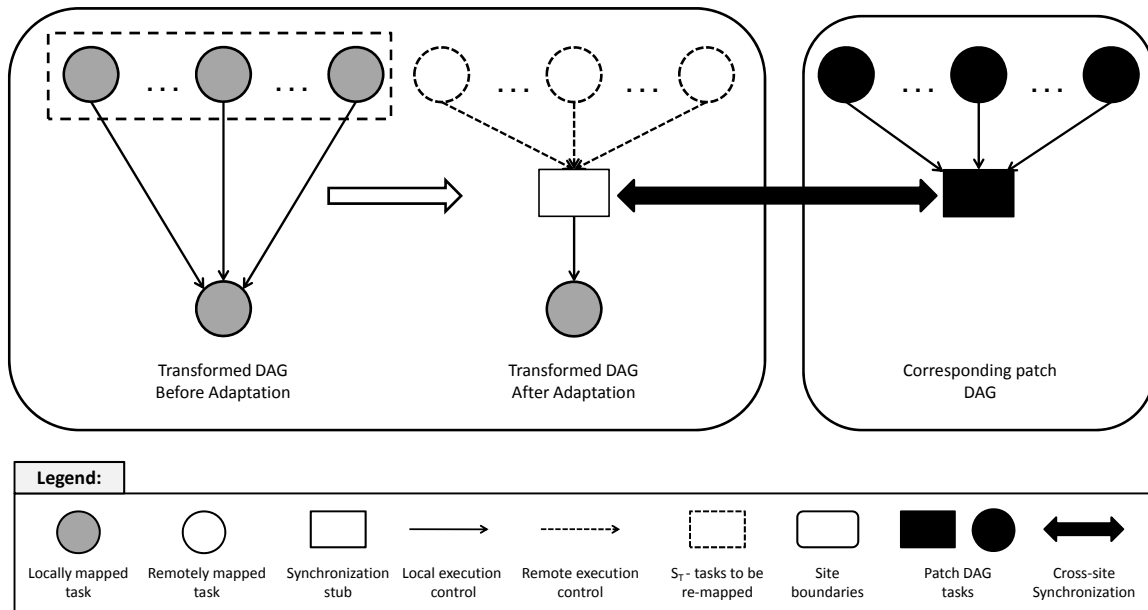


Figure 4.6: Patch DAG pattern corresponding to the adapted Join pattern in a scenario where a single site re-maps S_T

Fig. 4.7 illustrates the corresponding patch DAG structure in a scenario where two destination sites are involved in re-mapping of the tasks in S_T . In this scenario, two patch DAGs need to be composed, and one of those DAGs is designated as the primary patch DAG. Primary patch DAG is responsible for handling the synchronization with the originating site. In addition, an additional layer of synchronization is needed between the destination sites in this scenario.

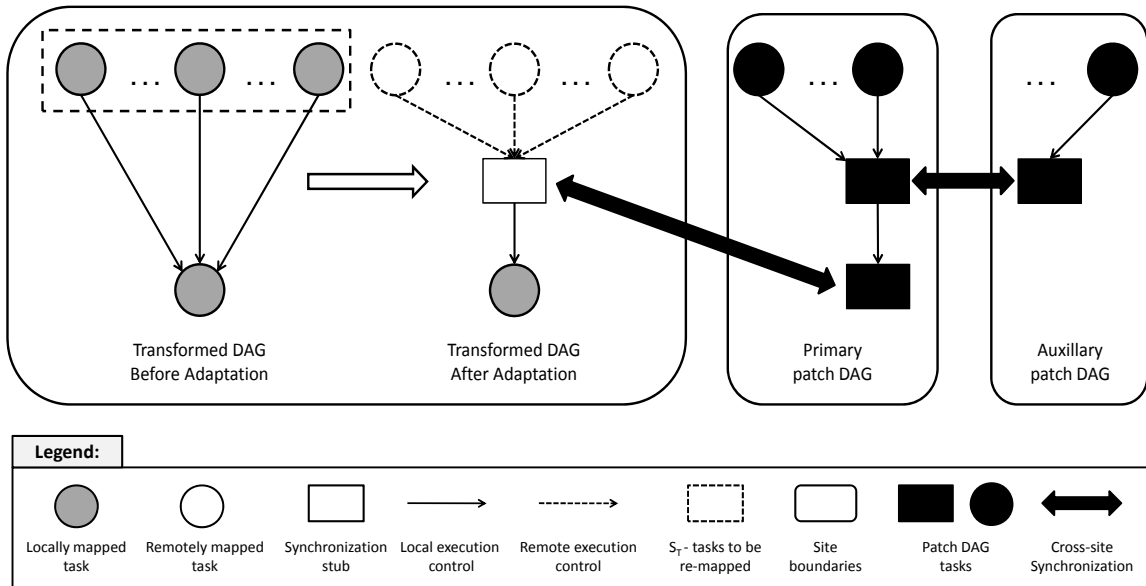


Figure 4.7: Patch DAG pattern corresponding to the adapted Join pattern in a scenario where two sites re-map the tasks in S_T

4.5 PROTOTYPE IMPLEMENTATION

We provide run-time adaptation through an adaptation daemon running at each site as illustrated in Fig. 4.1. The adaptation daemon constantly monitors the execution progress of the workflow and the length of the ready queue. It also has a thread that listens for requests that might come from other workflow managers.

Workflow execution engine of our prototype implementation [52] is based on Condor DAGMan [11, 16] workflow execution engine.

The specific functionality of an adaptation daemon in case of an adaptation process varies depending on its role.

- Adaptation process at the originating site: As a fault-tolerance feature, Condor DAGMan provides a checkpoint-style recovery mechanism via the creation of rescue DAGs. The rescue DAG represents the specification of a DAG that was not run to completion due to various reasons. In a rescue DAG

specification, those tasks that have successfully finished execution are labeled as DONE, which prevents them to be executed again. At the originating site, we utilize the rescue DAG mechanism to halt the DAG specification prior to the adaptation and then to re-enact the corresponding DAG specification after adaptation. To generate the proper corresponding DAG specification, decisions made in the adaptation plan should be reflected on to the standard rescue DAG specification. This is achieved via labeling re-mapped tasks as DONE and also performing similar DAG transformation activities as was done for decentralization purposes. Fig. 4.8 lists the steps performed at the originating site during a basic adaptation process.

1. Detecting the need for adaptation
2. Selection of task(s) to re-map
3. Pick a site among candidates
 - a. if negotiation is successful, move to step 4
 - b. if negotiation is unsuccessful, pick another site, repeat step 3
 - c. if tried all the sites, abort the adaptation
4. Send the list of task(s) to be re-mapped to the receiver site
5. Remove re-mapped tasks from the local queue
6. Wait for a rescue DAG to be created by Condor DAGMan
7. Modify the rescue DAG
 - a. label the migrated tasks as DONE
 - b. incorporate Pre script(s) for those local tasks that are children of the migrated task(s)
8. Submit the modified rescue DAG to Condor DAGMan.

Figure 4.8: Adaptation process at the originating site

- Adaptation process at the destination site(s): At the destination site(s), corresponding patch DAG specification(s) needs to be generated. DAG pattern of the task(s) received from the originating site provides enough

information for this purpose. The destination site can then perform the same kind of DAG transformation activities to generate the corresponding patch DAG specification. Following this, destination site submits and orchestrates the patch DAG at its local site in isolation. Fig. 4.9 lists the steps performed at the destination site during a basic adaptation process.

1. Receive the request to re-map task(s)
2. Check the status of the local queue and respond to the request
 - a. if response is negative, go back to step 1
 - b. if response is positive, go to step 3
3. Receive the list of task(s)
4. Generate the corresponding patch DAG specification
5. Submit the patch DAG to the local Condor DAGMan

Figure 4.9: Adaptation process at the destination site

4.6 PERFORMANCE EVALUATION

To evaluate the performance of our prototype implementation based on our adaptation framework which is built upon the decentralized workflow orchestration framework, we conducted several experiments on a real distributed computing environment. We have conducted our experiments using one synthetic workflow and one real-world workflow (i.e. Montage astronomy workflow [4]).

The DAG structure of the synthetic workflow is illustrated in Fig. 4.10. Each task in this workflow is assigned a random runtime between 100 and 300 seconds. We also associated each data dependency between tasks with a random size data transfer activity (assuming an average rate of 1 Megabyte/second for each data transfer) to result in an average communication-to-computation ratio (CCR) of 0.1, 0.5, and 1, to generate three different workflow instances.

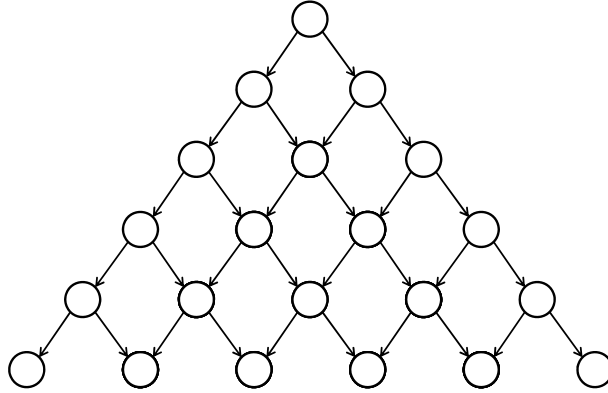


Figure 4.10: DAG structure of the synthetic workflow

The Montage astronomy workflow [4] has a DAG structure as illustrated in Fig. 3.3 and the detailed execution profile of Montage workflow tasks and dependencies is available [106]. In our experiments, we have used three different Montage workflow instances differing from each other by the number of total tasks in them. Accordingly, these workflow instances include 20, 40, and 80 total tasks respectively.

We conduct our experiments via the decentralized orchestration of each workflow instance under various load conditions. To this end, each time, we introduce no overload, partial overload, or constant overload on the computational resources, and measure the makespan metric for the execution of the workflow instance. We repeat this process 5 times for each workflow instance with and without the utilization of our run-time adaptation framework and calculate the average makespan value.

4.6.1 Experimental Setup

We conducted our experiments on a real multi-domain computational infrastructure, XSEDE [3], which is a collaboration of specialized high-performance and high-throughput computational and storage systems across the United States of America, which is made available through a National Science Foundation (NSF) project. These

computational and storage resources within the XSEDE project are owned and administered by different organizations. The authentication of users, communication interactions, and access to resources among sites are made possible mainly through the Globus middleware [25, 26]. Various other software tools and services are used for more specialized purposes. We have utilized the computational resources of two specific systems within the XSEDE infrastructure, namely the NCSA Abe cluster, and Purdue Condor pool.

The following is the brief information regarding the properties of resources at the time our experiments were conducted. Abe cluster is located at National Center for Supercomputing Applications (NCSA), Urbana, Illinois, and provides a total of 9600 CPU cores. Purdue Condor pool is located at Purdue University, West Lafayette, Indiana, and provides over 14000 CPU cores (the actual number varies since idle resources are scavenged into the system over time). Each system has Network File System (NFS) support to provide uniform access to data files across all local compute nodes. The main tool that is used to transfer data across these domains is GridFTP [42].

Each system, by default, provides a local deployment of Condor [23], Condor DAGMan [16], Condor-G [24], and Pegasus [18] software tools that are used for the various phases of our experiments. Pegasus [18] is used to map abstract workflow instances and generate the corresponding concrete workflow instances. There has been made no changes to the existing coding of these tools throughout the implementation and conducting of our experiments.

To prevent the effects of variance in the availability of computational resources on the performance results, we monitored and controlled the amount of resources utilized

throughout the experiments. First, in each domain, we made changes to the local job scheduler configuration, so that maximum 4 computational jobs (translates to workflow tasks in our context) are allowed to execute at any given time. We also monitored the time each workflow task spends waiting in the local job scheduler queue. If a workflow task spends more than 1 minute waiting in the local queue (besides the artificial queue wait time introduced by us to emulate various resource overload conditions), the result obtained from the execution of that workflow instance is discarded.

Our adaptation daemon has been configured to check the status of the tasks in the system every 30 seconds. If more than 80% of the tasks in submission are in the queue, then the adaptation daemon performs the migration of all the tasks in the queue. If between 50% and 80% of the tasks in submission is in the queue, then the adaptation daemon probabilistically selects a number of tasks from the queue for migration to reduce the queue length below 50%.

4.6.2 Evaluation Results under Constant Overload

In the first set of experiments, we introduced extra overload to one of the sites (Purdue Condor pool) throughout the execution of a workflow instance. We simulate the presence of an overload in the system by putting all the jobs submitted at Purdue Condor pool into a Hold state for a random time between 100 and 250 seconds. If no adaptation is applied, a task in a Hold state waits in the local queue with no progress until it is released.

Fig. 4.11 shows the makespan results from the execution of the synthetic workflow. Makespan value under no overload is also measured for reference. Makespan value obtained when the adaptation mechanism is in place shows better results than with no adaptation for the workflows with average CCR values of 0.1 and 0.5. However, in the

case of the workflow with average CCR value of 1, the adaptive execution performs worse than the non-adaptive execution. This occurs due to costly data transfers between sites to facilitate the migration of those tasks chosen during adaptation.

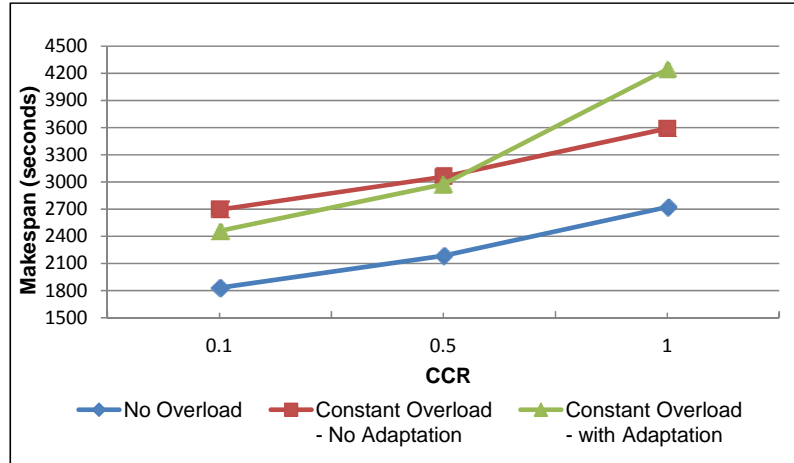


Figure 4.11: Makespan values for the synthetic workflow under constant overload

Fig. 4.12 shows the makespan results from the execution of the Montage workflow. Here, the makespan values for varying sizes of the Montage workflow are displayed. According to these results, our adaptation mechanism is able to improve the performance of the Montage workflow under constant load due to its specific data and computation characteristics. Also, we observe that our mechanism is able to provide this improvement for varying sizes of the Montage workflow.

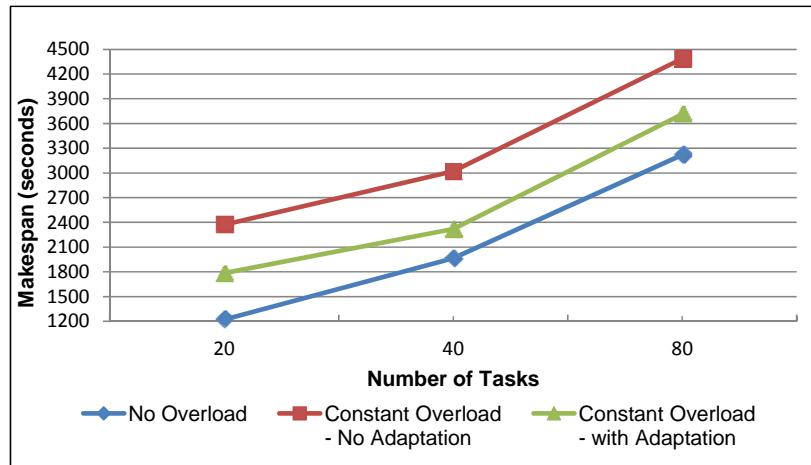


Figure 4.12: Makespan values for the Montage workflow under constant overload

In addition, Fig. 4.13 represents the performance improvement achieved via our adaptation framework for the Montage workflow under constant overload by comparing average queue wait times. With the deployment of our adaptation framework, the extended queue wait times resulting under constant overload are reduced significantly.

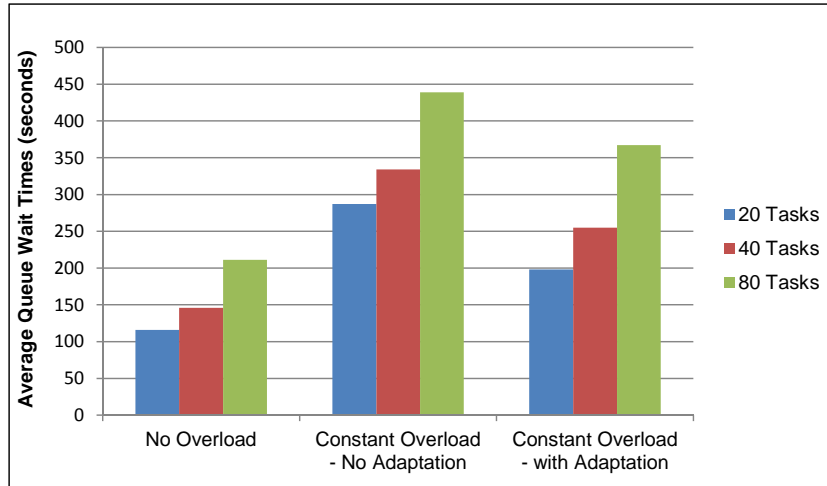


Figure 4.13: Average queue wait times for the Montage workflow under constant overload

4.6.3 Evaluation Results under Partial Overload

In this set of experiments, each task submitted at Purdue Condor pool is put in to Hold state with a 0.5 probability. The time to keep a task in the Hold state is between 100 and 250 seconds.

Fig. 4.14 shows the makespan results from the execution of the synthetic workflow under the partial load. For the average CCR values of 0.1, and 0.5, the adaptive workflow execution performs slightly better than the non-adaptive execution. For the average CCR value of 1, adaptive workflow execution generates worse results than the non-adaptive execution; however the performance difference is much less in this case compared with the constant overload scenario.

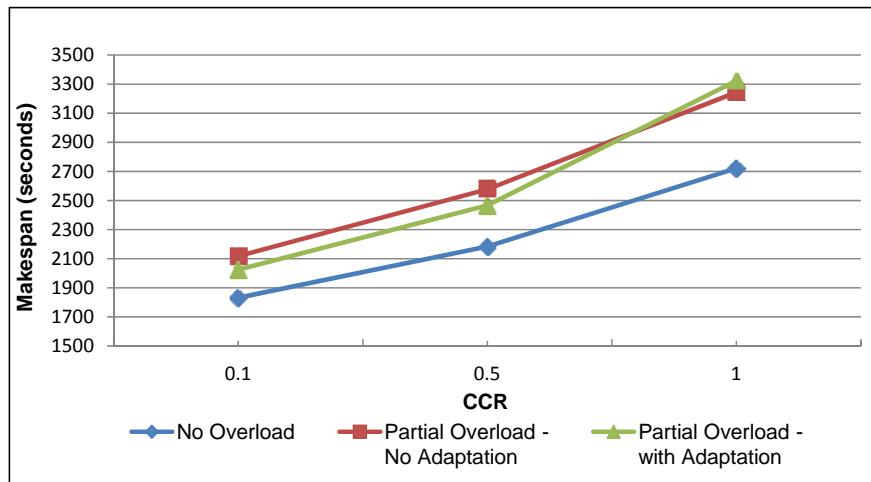


Figure 4.14: Makespan values for the synthetic workflow under partial overload

Fig. 4.15 shows the makespan results from the execution of the Montage workflow. In this case, the improvement achieved through run-time adaptation is almost negligible. The main reason behind this behavior is the selection of only a portion of tasks to be migrated from the partially overloaded site. Due to the strong existence of data dependencies among those tasks that are migrated with the ones that are not migrated, the

benefits gained through the reduction in queue wait times are almost cancelled out by the additional data transfer costs resulting from the adaptation process.

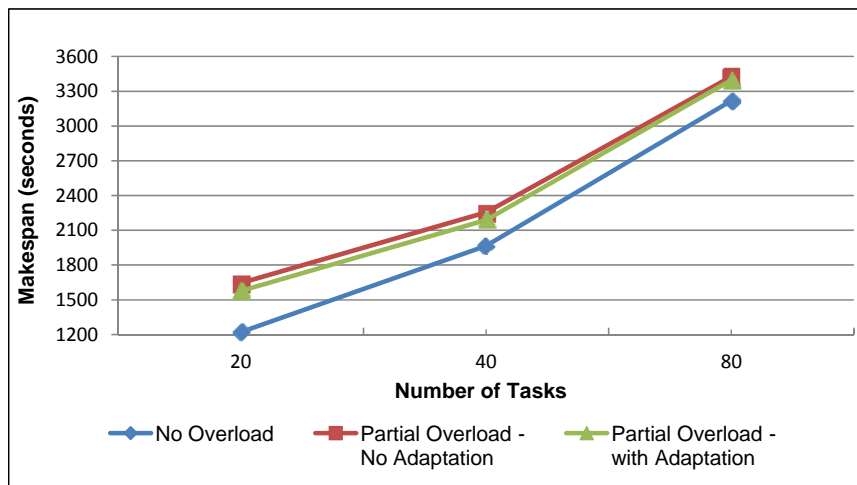


Figure 4.15: Makespan values for the Montage workflow under partial overload

CHAPTER 5

OPTIMIZATION TECHNIQUES FOR PARAMETER STUDIES

In this chapter, we provide some optimization techniques for efficient orchestration of workflows that are especially targeted for parameter-sweep applications. Optimization techniques we provide are once again meant for large-scale workflows that span across multiple sites of resources. Thus, we base our optimization efforts on the decentralized workflow orchestration framework that we introduced in Chapter 3.

First, we give some background information regarding parameter-sweep workflows and their characteristics. Then, we introduce the limited behavior resulting from standard decentralized orchestration of such workflows. Based on those limitations, we explain the design and benefits of a few generic optimization patterns. Then, we discuss some issues that are relevant to the implementation of these patterns. Finally, we present a couple of real-world parameter studies and discuss how our optimization patterns benefit such studies.

5.1 BACKGROUND

A large number of scientific fields make use of computational approaches, at varying degrees, to conduct research. Due to the increased availability of data and computational resources, utilization of such computational research tools has been following a consistently upward trend. This does not however necessarily point to a linear increase in the variety of scientific exploration. Most of the increased usage of computational tools arises from the repeated and extensive usage of the same type of tools on different sets of data. These types of computational studies are commonly and broadly referred as parameter studies.

Scientific exploration demands repeatable experimentation, and sound and reliable analysis. Computational resources have provided great means to achieve these goals at much deeper level and much wider scale in almost all fields of research. Using computational tools, scientists are able to conduct larger studies that produce much detailed and high-precision results. Most of the scientific exploration necessitates the same or similar experimentation and analysis tools to be run over a wide-ranging spectrum of parameters/conditions. The multitude of observations helps the scientist to make better decisions concerning her study. Although, scientists from almost all disciplines, to a certain extent, conduct such parameter studies, here we especially focus on those that are compute- and data-intensive in terms of today's existing technological capabilities.

Due to the repetitive nature of large parameter studies, scientists ideally utilize automation tools in this otherwise labor-intensive process. DAG-based workflows can easily and successfully capture the business logic of most parameter studies. After the scientist or a computational expert captures the business logic of the parameter study in a DAG-based workflow specification, it can be handed over to a workflow management system to automate the rest of the lifecycle of the workflow.

At this point, we give more detailed information regarding the structure and characteristics of a couple of basic types of parameter-sweep workflow DAGs. Namely, these are: single-level parameter-sweep workflows and multi-level parameter-sweep workflows. These DAGs differ from each other by the number of levels that comprise the computation intensive portion of the workflow.

5.1.1 Single-level Parameter-sweep Workflows

Fig. 5.1 illustrates the DAG structure of a typical single-level parameter-sweep workflow. The top-level task acts as to generate and distribute the input data/parameters that would be consumed by the Processing Tasks. Each Processing Task receives a different set of input data/parameters and independently executes the same or similar software/service using those data/parameters. The results generated by these Processing Tasks are then passed on to the Data Aggregation/Post-Processing Task. This task, depending on the specifics of the study, performs some sort of data aggregation and/or post-processing activities utilizing the results acquired from Processing Tasks. The set of Processing Tasks in this given DAG structure comprises the parameter-sweep aspect of the workflow application.

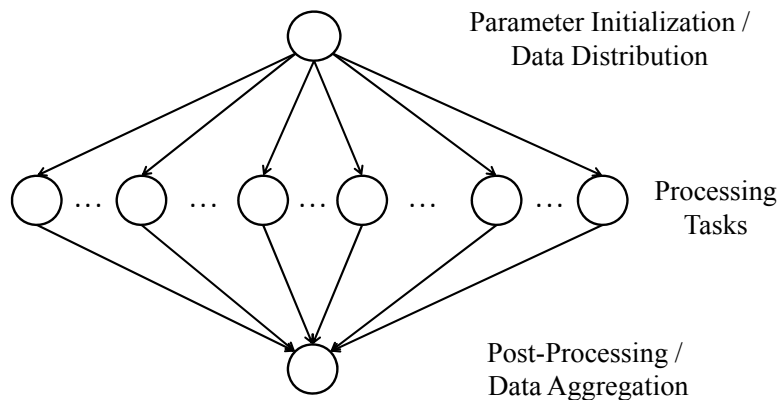


Figure 5.1: DAG structure of a single-level parameter-sweep workflow

In a parameter-sweep workflow, typically, the Processing Tasks are the most computationally intensive components of the workflow. At the same time, these tasks are typically embarrassingly parallel, making them eligible to be executed concurrently on various computational resources. Also, the top-level task (i.e. Parameter Initialization)

and the bottom-level task (i.e. Post-processing Task) are typically more data-intensive in nature than Processing Tasks.

5.1.2 Multi-level Parameter-sweep Workflows

Fig. 5.2 illustrates the DAG structure of a typical two-level parameter-sweep workflow. These types of workflows are used to perform more intense exploration/analysis activities iteratively at each level based on the results acquired from the previous level. A more coarse-grain computation and analysis is performed over a higher-number of Processing Tasks at the first level. Then, based on the results acquired at the first level, more fine-grain computation and analysis is performed using the same or perhaps a different software/service over a less number of Processing Tasks at the second level. This process can be repeated in the same manner as many times as deemed necessary by the user.

As also can be seen from Fig. 5.2, in a two-level parameter-sweep workflow DAG, each of level-1 and level-2 tasks are comprised of same kind of DAG structure as it is found in a single-level parameter-sweep workflow DAG. Typically, purpose of employing such multiple levels in a parameter-sweep study is to conduct a coarse-grain analysis at level-1 for a large number of potential parameters. After these potential parameters are narrowed down to a more manageable number, a finer-grain analysis can be conducted at level-2.

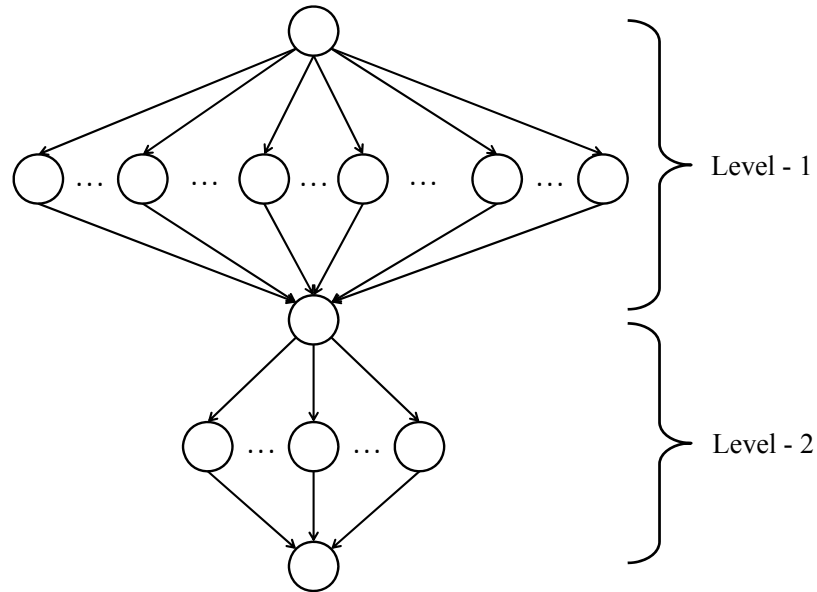


Figure 5.2: DAG structure of a two-level parameter-sweep workflow

5.2 OPTIMIZATION PATTERNS

Here, we introduce the optimization patterns [63] that we employ on our existing decentralized orchestration framework (Chapter 3) [51, 52]. Even though they may be applicable to more general instances of workflows as well, these patterns are mainly designed to exploit the general characteristics of parameter-sweep workflows.

Optimization patterns that we introduce here suggest minor changes in the DAG structure of the workflow. These changes are proposed due to the specific nature of a parameter-sweep workflow where the dependencies among tasks are much more loosely coupled than a more generic workflow instance. Also, the relationships among different levels of tasks at a parameter-sweep workflow are more uniform and flexible than a more generic workflow instance. By exploiting these characteristics of parameter-sweep workflows, we provide optimizations to the decentralized orchestration of such workflows.

5.2.1 Parameter Initialization / Data Distribution

This pattern applies to the Fork/Branch pattern that manifests itself between the top-level Parameter Initialization Task and the lower-level Processing Tasks. At a multi-site deployment of a large-scale parameter-sweep workflow, all these embarrassingly parallel Processing Tasks have control and data dependency on the single Parameter Initialization Task. Fig. 5.3 illustrates a typical scenario for such behavior. Each different color notates deployment of a task on a different site. Fig. 5.3 also illustrates the data/control logistics of this pattern following a centralized orchestration approach. Accordingly, the centralized workflow manager handles all the data and control logistics even though the tasks span across multiple different sites. This behavior incurs significant control and data overhead to the total wall-clock execution time of the workflow (makespan).

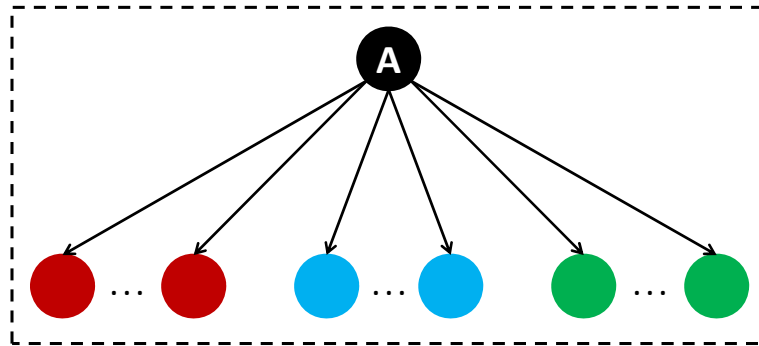


Figure 5.3: DAG-mapping and centralized orchestration of the Fork/Branch pattern

Fig. 5.4 illustrates the overall view for the orchestration of the same pattern following our standard decentralized approach. In these Figures, dashed lines indicate the boundaries for the responsibility of each site. Each site employs its own local workflow manager as it is discussed in Chapter 3. Also, the rectangle boxes that cross between two site boundaries illustrate the synchronization activities which are inserted to the DAG

specification during the second phase of DAG transformation process (Fig. 3.7). Thus, Fig. 5.4 illustrates the collaborative DAG structures and the interactions among them to achieve synchronization.

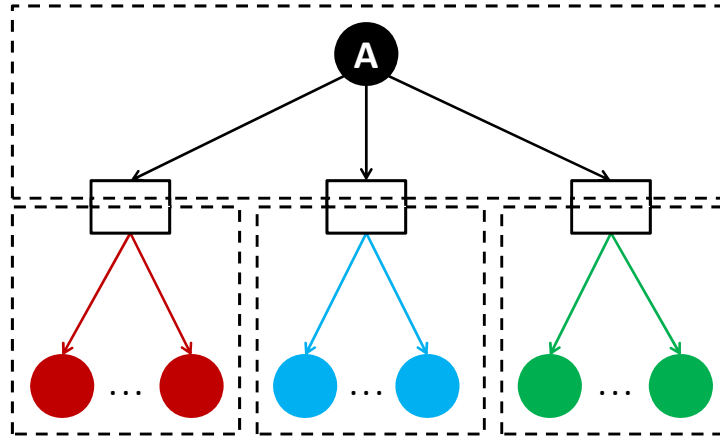


Figure 5.4: Overall view for the decentralized orchestration of the Fork/Branch pattern

Due to flexible characteristics of the Parameter Initialization Task at the top-level of this DAG pattern, we propose an optimization pattern as illustrated in Fig. 5.5. In Fig. 5.5, it can be seen that, task A (Parameter Initialization Task) has been replicated such that each site employs its own copy. As a result, each peer workflow manager can orchestrate the local DAG structure completely independent from others. Thus, the orchestration of the presented DAG structure incurs no additional overhead to the workflow execution time. Notice that, the replicated task A is referred as A' as it may be necessary to make minor adjustments in the business logic of task A. We will discuss this issue in Section 5.3.

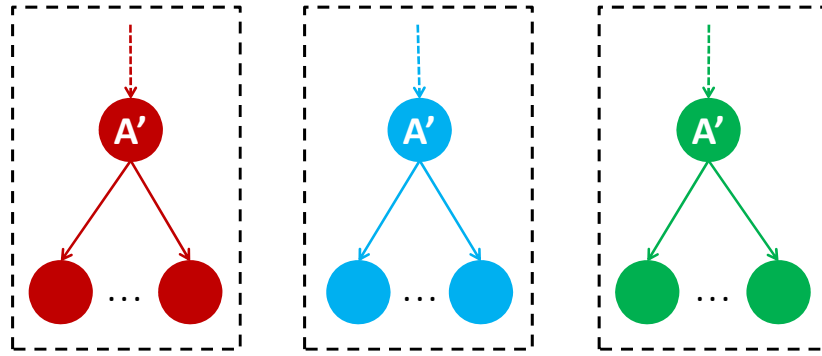


Figure 5.5: Overall view for the optimized Fork/Branch pattern DAG orchestrated in decentralized manner

5.2.2 Post-Processing / Data Aggregation

This pattern applies to the Join pattern that manifests itself between the Processing Tasks and the lower-level Post-Processing Task. At a multi-site deployment of a large-scale parameter-sweep workflow, all these embarrassingly parallel Processing Tasks have control and data dependency with the single Post-processing Task. Fig. 5.6 illustrates the data/control logistics of this pattern following a centralized orchestration approach. Accordingly, all the data and control logistics is handled through the centralized workflow manager even though the tasks span across multiple different sites. This behavior incurs significant control and data overhead to the total workflow execution time.

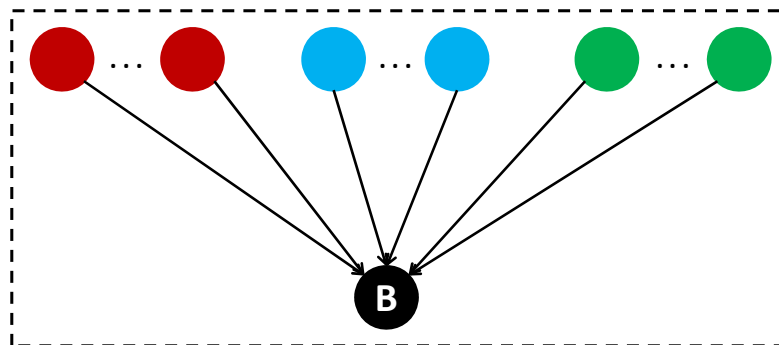


Figure 5.6: DAG-mapping and centralized orchestration of the Join pattern

Fig. 5.7 illustrates the overall view for the orchestration of the same pattern following our standard decentralized approach. The collaborative DAG structures and the interactions among them to achieve synchronization are illustrated.

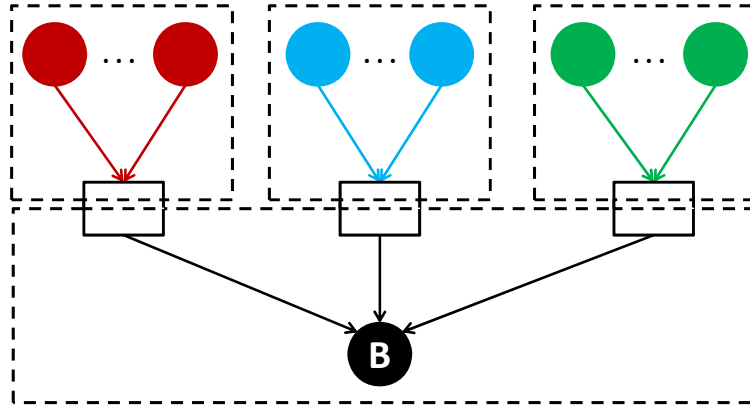


Figure 5.7: Overall view for the decentralized orchestration of the Join pattern

We propose a corresponding optimization pattern for this DAG pattern as illustrated in Fig. 5.8. In Fig. 5.8, it can be seen that, task B (Post-Processing Task) has been replicated such that each site employs its own copy. As a result, the collaborative peers can perform the Post-processing Task at their local site. Notice that, the replicated task B is now referred as B' as it may be necessary to make minor adjustments in the business logic of task B. Also, notice that, in this case we need a second-level Post-Processing Task (task B*) to further perform post-processing on local results. However, even with the inclusion of task B*, the optimized orchestration can help reduce the overheads incurred due to data transfers between Processing Tasks and (original) task B. The reasoning behind this behavior is as follows. Due to their nature, most scientific applications generate large sizes of output files. During the non-optimized orchestration (centralized or decentralized) of such a pattern, all of the individual output files generated by Processing Tasks are required to be transferred to a remote site, so that a single task B

can perform post-processing activities on all this data. According to our optimized orchestration of this pattern, individual output files are not required to be transferred to a remote site. Each site needs to only transfer the output files generated by the local task B' (that are generally much more modest in size), which are then further post-processed by task B*.

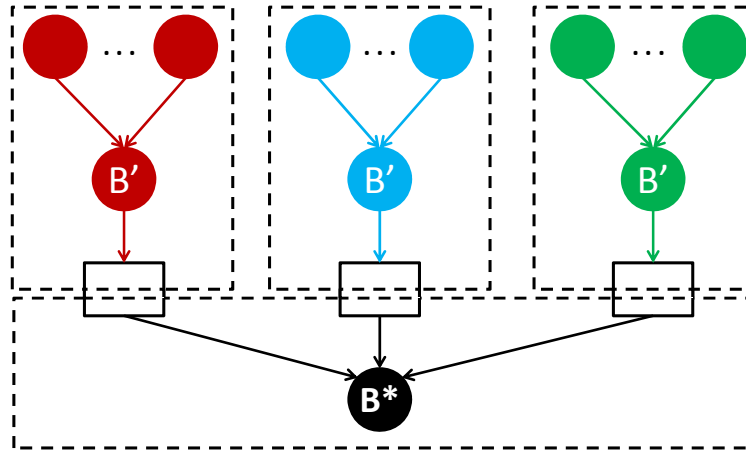


Figure 5.8: Overall view for the optimized Join pattern DAG orchestrated in decentralized manner

5.3 PROTOTYPE IMPLEMENTATION

Workflow execution engine of our prototype implementation is based on Condor DAGMan [11, 16] workflow execution engine. However, as mentioned before, the original orchestration approach for Condor DAGMan is centralized. Our optimization patterns rely on the decentralized orchestration framework that we discussed in Chapter 3. Thus, all the DAG specifications and the workflow orchestration activities mentioned here correspond to those discussed in Chapter 3.

Here, we only address specific implementation issues pertaining to the optimization efforts (patterns) that we introduced in Section 5.2.

As it was explained in Section 5.2, we introduce optimization patterns to exploit certain characteristics of large-scale parameter-sweep workflows. However, these patterns necessitate minor changes to the original DAG structure, which in turn may affect the business logic of the workflow application. Thus, a domain expert should verify these structural changes to be appropriate. After the domain expert approves the DAG structure change, he/she should also be consulted regarding the adjustments intended for Parameter Initialization Task and Post-processing Task. Once the appropriate task adjustment methods are agreed upon, these methods can be reused multiple times to run similar instances of large-scale parameter-sweep workflows.

As suggested above, the domain expert should contribute to the proper method for replicating the Parameter Initialization Task (see Fig. 5.5). In some cases, the Parameter Initialization Task involves generating input data/parameters in a randomized way. In some other cases, this task involves splitting a wide-range of input data/parameters uniformly among the Processing Tasks. For these and similar cases, the proper task adjustment methods are expected to be much more trivial than more sophisticated cases.

The business logic of the Post-processing/Data Aggregation Tasks also usually consists of standard and well-defined behavior, which may be adjusted accordingly through consultation with a domain expert. For example, in some cases this task involves generating statistical results from the data generated by Processing Tasks. In some other cases, this task mainly involves choosing the best results (or eliminating worst results) among all the generated results according to certain criteria. For these and similar cases, the proper task adjustment methods are expected to be much more trivial than more sophisticated cases.

After the consultation and proper task adjustment phases are completed successfully, these behaviors are required to be specified in the Aggregated DAG specification. At this point, we provide this information (i.e. tasks to be restructured, task adjustment methods) manually. In the next stage of the DAG transformation process, each local site comes up with its own local transformed DAG specification. Finally, since we only make use of standard Condor DAGMan functionalities throughout the transformation process, no other changes are necessary in the system.

5.4 PERFORMANCE ANALYSIS

5.4.1 Molecular Modeling and Dynamics

Molecular modeling [64] is concerned with theoretical and computational techniques used for mimicking/simulating the behavior of molecules. As the size of molecules increase, the computational requirements of modeling the molecule also increases. Molecular modeling and simulation techniques are used for various purposes in the fields of computational chemistry [67], computational biology [68], materials science [69], drug design [70, 71], protein folding [72-74], and others.

Another related line of research is the study of molecular dynamics. Molecular dynamics [65] mainly is the study of simulating the physical movements of atoms and molecules abiding by the rules of N-body simulation [66] concept. Because of the vast number of particles and vast parameter space, computational simulation software is essential for conducting this type of study. Several advanced software suites [55 - 59] have been developed and utilized by the research community to serve the needs of various molecular modeling and dynamics purposes.

The specific parameter-sweep workflow we provide as a sample scenario in the area of molecular dynamics is concerned with the study of advanced materials [53, 54]. The field of nanoscience and nanotechnology is the initiative to assemble and manipulate matter at the molecular or even atomic level. Physics, chemistry, biology and materials engineering jointly contribute to the effort of tailoring novel materials by arranging small constituents in a controlled manner. Nanoscience research is largely guided by the promise of novel devices at the nanoscale that will be vastly more compact, fast, flexible and energy efficient than the tools used in current technology. To realize this goal, it is necessary to understand the principles that govern the nanosphere, characterized by spatial dimensions in the order of 10^{-9} m. Similarly, in-depth knowledge of the physical and chemical properties of nanomaterials is required. Studies on these and related problems involve various computational methods based on quantum physics and chemistry, such as density functional theory (DFT) as well as methods of ab initio theory.

The specific software suite used in this parameter space exploration for advanced material studies, due to our domain expert's specific needs at East Tennessee State University (ETSU), is the Vienna Ab initio Simulation Package (VASP). VASP [108, 109] is a program for atomic scale materials modeling, and is used to perform electronic structure calculations and quantum-mechanical molecular dynamics. This is accomplished by solving the Schrödinger equation - a partial differential equation that describes how the quantum state of some physical system changes with time - using appropriate approximations for energies and forces. VASP allows researchers to perform a wide variety of atomic/molecular interaction simulations, and is widely used by research groups in academia and industry worldwide.

VASP is a computation intensive application that provides both shared-memory (OpenMP) and distributed-memory (MPI) parallelization. Due to its tightly-coupled communication characteristics, it scales nicely over multiple nodes only if those nodes communicate over a low-latency network infrastructure (e.g. Infiniband). Thus, a single VASP application is not suitable to be executed across different resource sites.

VASP as an input receives multiple input files (POSCAR, INCAR, POTCAR, KPOINTS, etc.) that define the conditions and boundaries for the specific simulation study. An unlimited number of simulations can be studied over varying the setup configuration files of VASP. In its simplest case, the POSCAR file defines the initial positions of the atoms used within the simulation. In our benchmark parameter space exploration conducted with VASP at ETSU cluster computing resources, we utilized many variations of the POSCAR file settings. To automate this process, we created a script that would generate each POSCAR file that is used for each VASP simulation run. After the pre-set number of VASP simulations are run independently from each other, the results from each run is post-processed simply comparing the specific parameter of interest (i.e. energy value) and choosing the most appropriate ones among them.

The parameter space exploration study explained above complies with all aspects and characteristics discussed in our optimization patterns effort. The script that is used to initialize a subset of parameter setup serves as the Parameter Initialization Task. Each individual VASP run (which typically takes up multiple physical nodes at once through MPI parallelization) serves as a Processing Task. Finally, the task of selecting the simulations with best output results serves as the Post-Processing Task.

Here, we give more details regarding each component of the molecular dynamics parameter-sweep simulation study. The script that serves as the Parameter Initialization Task is a very simple executable that takes a few seconds to generate multiple input files with pseudorandom values. Thus, replication of this script does not incur a significant overhead on resources or the makespan of the parameter-sweep workflow. Depending on the availability of computational resources, we run between 10 and 100 VASP simulations as part of each parameter-sweep workflow. Each simulation run, in general, takes between 10 minutes and 2 hours to complete its execution and produces output data of size typically between 50 and 300 Megabytes. Thus, the size of data artifacts resulting from most of our parameter-sweep studies range between 0.5 and 30 Gigabytes. This means, when we utilize both clusters (namely, Knightrider and Blackpearl) at our disposal locally, by performing one local Post-Processing Task at each cluster, we eliminate the unnecessary costs associated with the transfer of around 0.25 and 15 Gigabytes of data.

The behavior and results explained above applies only to a single instance of VASP parameter-sweep simulation study. Due to its nature and vast parameter space, the same process is repeated many times in accordance with the computational availability and researcher needs.

5.4.2 Ensemble Forecasting – Weather Forecasting

Ensemble forecasting [75] is a computational prediction method that is used for simulating the state and behavior of a dynamical system. The fundamental aspect of ensemble forecasting studies is the conduct of multiple computational simulations with slightly different initial conditions/parameters. Those initial conditions/parameters are

typically gathered from a set of observations and measurements on the system being forecasted. The main reason for conducting multiple simulations, forecasting the same dynamical system, stems from the uncertainties inherent in forecasting models. Two major sources of uncertainty in forecasting models are:

1. Errors introduced by the use of imperfect initial conditions/parameters.
2. Errors introduced by the imperfections found in the forecast model itself. For example, the imperfection of mathematical methods (i.e. approximate methods) used in the forecast model.

Perhaps the most common and popular ensemble forecasting study is the one that is used for weather forecasting [62] purposes. In this scenario, the dynamic system is the atmosphere, which is a very complex system with a vast number of parameters. Computational forecasting techniques have been used for a while for weather forecasting. However, these forecasting models suffer from those imperfections mentioned above. Thus, to obtain more accurate results from a weather forecast simulation, an ensemble weather forecasting is used preferably.

Weather Research and Forecasting (WRF) model [60, 61] is one of the widely used forecasting models used for both research and operational forecasting purposes. The current version of WRF has been designed to run either on a single machine or on a cluster of homogeneous nodes. The distributed memory parallelization is established through MPI and it also has a tightly-coupled communication characteristics. The high resource requirements of WRF for fine resolution and ensemble forecasting demand a larger number of computing nodes with substantial memory and disk storage connected through a high speed network.

A typical ensemble WRF study can be conducted simply by simulating multiple forecasts with slightly different initial conditions. Another typical way to conduct an ensemble WRF study is done by simulating multiple forecasts through different models found in the WRF software suite (multi-model ensemble forecasting). In either ensemble WRF study, individual forecast results are combined to come up with a forecasting report. Here, the results are combined mostly following one of these two simple techniques:

1. Average of the individual forecasting results are calculated
2. Degree of agreement between individual forecasting results are measured, and then the results are represented by their overall spread

Fig. 5.9 displays the DAG structure and components in our WRF ensemble forecasting study [5] consisting of 5 ensemble members. Please note that, in our experimental studies, all ensemble members belong to the FIU administrative domain. Real is a standard module (real.exe) found in WRF. The main purpose of the Real module is to retrieve data from the WRF Preprocessing System (WPS) to initialize input conditions for the WRF simulation. Perturb module is a custom script that takes the input conditions generated by the Real module and performs random variations on those conditions. The outputs generated by the Perturb module is then consumed by each WRF instance (wrf.exe) executed at various ensemble member locations. Finally, the Aggregate module is another custom script that performs post-processing activities (e.g. calculating the average) on the output data generated by each ensemble member.

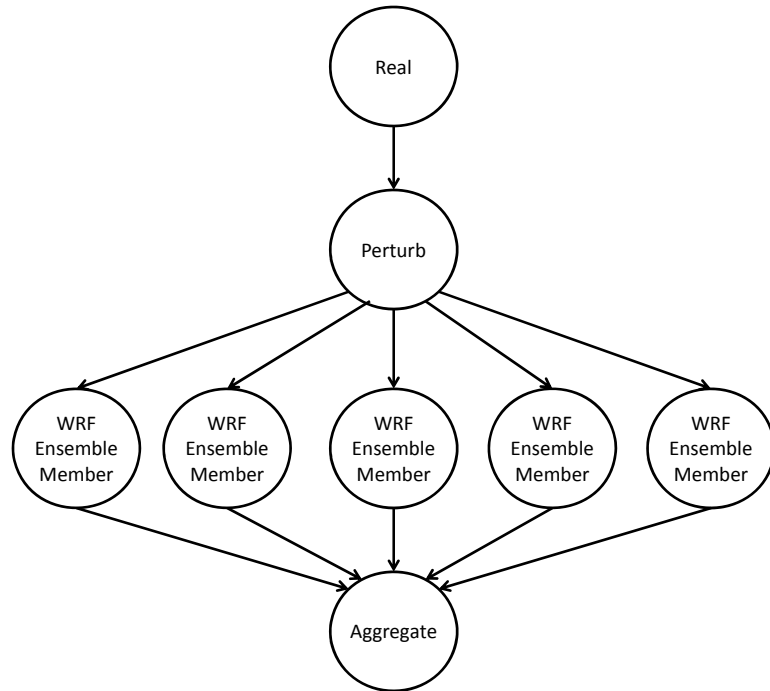


Figure 5.9: WRF Ensemble Forecast

The computational requirements for each of the following modules can be summarized as follows. Real and Perturb modules (Parameter Initialization Tasks) together, in general takes less than a minute to execute and does not change much based on the specific setup conditions of the studies. However, being the computationally heavy module of the studies, each WRF instance (Processing Task) show a great variation based on the geographical size (e.g. state of Florida vs. the continental U.S.A) and the degree of resolution (1 kilometer vs. 15 kilometers) sought in the specific study. At our local benchmark studies conducted on FIU cluster resources which covered a square area of 75 x 75 km with a 4 km resolution, the WRF instance execution time typically varied between 5 and 20 minutes. Nevertheless, except for more trivial scenarios, the computational requirements of other modules in the ensemble study are negligible compared to those of the WRF instances. Similarly, the output data artifacts associated

with the execution of a WRF instance varies greatly based on the geographical size and the degree of resolution. However, in our benchmark studies conducted at local FIU cluster resources, the size of output data generated by each WRF instance execution typically varied between 120 and 380 Megabytes.

The operational and research oriented forecasting studies targeted in the future cover much larger geographical area with much higher resolution. This greatly increases the demand for computational and storage capabilities. For those multi-site deployment and execution of such ensemble forecast studies, without careful investigation, unnecessary costs (especially data transfer costs) would be incurred through the automated orchestration. Due to the specific characteristics of such an ensemble forecast study, the optimization patterns introduced here can be utilized to avoid those additional costs to a large extent.

CHAPTER 6

CONCLUSIONS

Scientific applications are becoming more complex, resulting in the need for more computational resources than may be available to scientists locally. As a result, for many scientists, utilization of remote and heterogeneous computational resources has become a standard practice. However, effective utilization of these resources, especially for large-scale workflow applications, necessitates the employment of software tools that are efficient and adaptive.

In this dissertation, we propose a generic framework for the decentralization and run-time adaptation for the execution of large-scale workflow applications that span across diverse and heterogeneous resource domains. Our framework is applicable to any scientific workflow application that is specified in a DAG form. By investigating recurring DAG patterns, we devise corresponding transformation and adaptation patterns to incorporate decentralization and run-time adaptation capabilities to standard workflow execution managers.

Our decentralization framework adopts the separation of concerns and consequently does not alter the business logic of the application. Our prototype implementation on a standard workflow orchestration tool (i.e. Condor DAGMan) shows the feasibility and transparency of our approaches. Also, our framework is generic enough and can be easily incorporated by other orchestration tools.

Another related and major contribution in this dissertation focuses on large-scale parameter studies. A large and diverse group of computational scientific research efforts deals with parameterized studies, in which same or similar computational tools are

applied on different sets of data. Such uniform and well-defined analysis efforts can be encapsulated as parameter-sweep workflows. Due to the computation and data-intensive nature, resources that span across multiple domains may be needed for timely and efficient execution of this type of workflow as well.

Building on our existing decentralization framework, we provide further improvements to the orchestration of such workflows. Basically, we propose some additional optimization patterns specific to the characteristics and requirements of parameter-sweep workflows. By exploiting the general characteristics of parameter-sweep workflows, we provide ways to reduce control and data overheads associated with the decentralized orchestration.

We also discuss some implementation issues that arise from the adoption of these optimization patterns. The utilization of these optimization patterns may require minor changes to the structure of the original DAG specification and business logic of certain tasks. We discuss the potential drawbacks of making such changes and argue that in most cases they can be easily addressed by incorporating the feedback of a domain expert in the process. Even though these patterns were designed and aimed primarily for parameter-sweep workflows, they may be applicable, to a certain extent, for more general-purpose large-scale workflows as well.

By no means do we claim that the frameworks and mechanisms provided here address all problems in the field and represent final solutions to those problems. First of all, there are many other aspects of workflow management that we do not address in our studies. We primarily focus on the orchestration efficiency of certain workflows. Workflow orchestration occurs following or in cooperation with other components in a

workflow management system. Thus, decisions made and/or efficiency of those other components in the workflow management system may significantly affect or alter the usage and actual efficiency of our frameworks.

Also, the frameworks and mechanisms explained here are designed specifically for workflows and resource environments that meet certain criteria. Our proposals target large-scale workflows that span across multiple administrative resource domains. Here, a workflow being large-scale - among other things - depends on the specific size, capability, and availability of the resources that will be utilized. Thus, we do not attempt to provide any specific details pertaining to this aspect of a workflow.

Another feature of our proposals is their generic applicability to the application and resource environments. Due to the nature of the field, both application and resource environments happen to vary and scale in a very wide range of size and feature-list. Also, both application and resource environments are dynamic entities with future size and feature aspects in flux. Thus, our proposals aimed for no specific current or future application/resource platform.

The solutions provided in our study can be extended in various ways. First of all, the prototype implementation of our frameworks/mechanisms can be improved and adopted by other workflow execution managers as well. It is quite plausible that, during these extension efforts, further and/or better solution designs to the same issues may emerge.

Second, our studies focus only on DAG-based workflows. The artifacts of our studies may be extended to address non-DAG-based workflows as well. The major

differences between non-DAG-based and DAG-based workflows are the availability of selection and iteration patterns in the workflow structure of the former.

Third, it is also an interesting avenue of research to investigate various business and technical aspects of cloud bursting on the orchestration and adaptation of workflow applications. Cloud bursting is basically the act of shifting some or all computational requirements of an application to public and/or private cloud resources at runtime, due to unavailability or insufficiency of local resources. Due to specific concerns and requirements of them, cloud bursting of large-scale workflows would require careful decision-making and sophisticated system designs. Decentralization and adaptation frameworks introduced here may provide a good starting point for such studies.

Last but not least, further optimization patterns and mechanisms can be incorporated to the existing framework. It may be possible to identify certain DAG (even non-DAG) patterns, which can then be exploited accordingly to provide further optimization benefits.

REFERENCES

- [1] "FutureGrid.". Available: <http://futuregrid.org/>
- [2] "Open Science Grid". Available: <http://www.opensciencegrid.org>.
- [3] "XSEDE". Available: <http://www.xsede.org/>
- [4] Berriman, G. B., et al., "Montage: A Grid enabled image mosaic service for the national virtual observatory." San Francisco : ASP Conference Proceedings, 2003. Vol. 314.
- [5] Sadjadi, S., et al. "Transparent grid enablement of weather research and forecasting." In Proceedings of the Mardi Gras Conference 2008 - Workshop on Grid-Enabling Applications, Baton Rouge, Louisiana, USA, January 2008.
- [6] Oinn, T., Addis, M. J., Ferris, J., Marvin, D. J., Senger, M., Carver, T., Greenwood, M., Glover, K., Pocock, M. R., Wipat, A. and Li, P., "Taverna: a tool for the composition and enactment of bioinformatics workflows." s.l. : Bioinformatics, 2004, Issue 17, Vol. 20, pp. 3045-3054.
- [7] Bray, Tim, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. "Extensible markup language (XML)." *World Wide Web Consortium Recommendation REC-xml-19980210*. <http://www.w3.org/TR/1998/REC-xml-19980210> (1998).
- [8] Leymann, Frank. "Web services flow language (WSFL 1.0)." (2001).
- [9] Andrews, Tony, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu et al. "Business process execution language for web services." (2003).
- [10] Yu, Jia, and Rajkumar Buyya. "A novel architecture for realizing grid workflow using tuple spaces." In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pp. 119-128. IEEE, 2004.
- [11] Frey, James. "Condor DAGMan: Handling inter-job dependencies." *University of Wisconsin, Dept. of Computer Science, Tech. Rep* (2002).
- [12] Murata, Tadao. "Temporal uncertainty and fuzzy-timing high-level Petri nets." In *Application and Theory of Petri Nets 1996*, pp. 11-28. Springer Berlin Heidelberg, 1996.
- [13] Murata, Tadao. "Petri nets: Properties, analysis and applications." *Proceedings of the IEEE* 77, no. 4 (1989): 541-580.

- [14] Rumbaugh, James, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.
- [15] Casavant, Thomas L., and Jon G. Kuhl. "A taxonomy of scheduling in general-purpose distributed computing systems." *Software Engineering, IEEE Transactions on* 14, no. 2 (1988): 141-154.
- [16] Team, Condor. "DAGMan: A Directed Acyclic Graph Manager." *See website at <http://www.cs.wisc.edu/condor/dagman>* (2005).
- [17] Deelman, Ewa, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, Kent Blackburn et al. "Mapping abstract complex workflows onto grid environments." *Journal of Grid Computing* 1, no. 1 (2003): 25-39.
- [18] Deelman, Ewa, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, and Miron Livny. "Pegasus: Mapping scientific workflows onto the grid." In *Grid Computing*, pp. 11-20. Springer Berlin Heidelberg, 2004.
- [19] Chervenak, Ann, Ewa Deelman, Ian Foster, Leanne Guy, Wolfgang Hoschek, Adriana Iamnitchi, Carl Kesselman et al. "Giggle: a framework for constructing scalable replica location services." In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pp. 1-17. IEEE Computer Society Press, 2002.
- [20] Deelman, Ewa, Carl Kesselman, and Gaurang Mehta. *Transformation catalog design for GriPhyN*. Technical report griphyn-2001-17, 2001.
- [21] Czajkowski, Karl, Steven Fitzgerald, Ian Foster, and Carl Kesselman. "Grid information services for distributed resource sharing." In *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*, pp. 181-194. IEEE, 2001.
- [22] Team, Condor. "Condor® Version 7.7. 6 Manual." (2012).
- [23] Litzkow, Michael J., Miron Livny, and Matt W. Mutka. "Condor-a hunter of idle workstations." In *Distributed Computing Systems, 1988., 8th International Conference on*, pp. 104-111. IEEE, 1988.
- [24] Frey, James, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. "Condor-G: A computation management agent for multi-institutional grids." *Cluster Computing* 5, no. 3 (2002): 237-246.
- [25] Foster, Ian, and Carl Kesselman. "Globus: A metacomputing infrastructure toolkit." *International Journal of High Performance Computing Applications* 11, no. 2 (1997): 115-128.

- [26] Foster, Ian, and Carl Kesselman. "The globus toolkit." *The grid: blueprint for a new computing infrastructure* (1999): 259-278.
- [27] Zhou, Songnian. "LSF: Load Sharing in Large Heterogeneous Distributed Systems." In *I Workshop on Cluster Computing*. 1992.
- [28] Ellert, Mattias, Aleksandr Konstantinov, Balázs Kónya, Oxana Smirnova, and Anders Wäänänen. "The NorduGrid project: Using Globus toolkit for building Grid infrastructure." *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 502, no. 2 (2003): 407-410.
- [29] Smirnova, Oxana, Paula Eerola, Tord Ekelöf, Mattias Ellert, John Renner Hansen, Aleksandr Konstantinov, Balázs Kónya, Jakob Langgaard Nielsen, Farid Ould-Saada, and Anders Wäänänen. "The NorduGrid architecture and middleware for scientific applications." In *Computational Science—ICCS 2003*, pp. 264-273. Springer Berlin Heidelberg, 2003.
- [30] Erwin, Dietmar W., and David F. Snelling. "UNICORE: A Grid computing environment." In *Euro-Par 2001 Parallel Processing*, pp. 825-834. Springer Berlin Heidelberg, 2001.
- [31] Romberg, Mathilde. "The UNICORE grid infrastructure." *Scientific Programming* 10, no. 2 (2002): 149-157.
- [32] Bayucan, Albeaus, Robert L. Henderson, Casimir Lesiak, Bhroam Mann, Tom Proett, and Dave Tweten. *Portable batch system: External reference specification*. Vol. 5. Technical report, MRJ Technology Solutions, 1999.
- [33] Henderson, Robert L. "Job scheduling under the portable batch system." In *Job scheduling strategies for parallel processing*, pp. 279-294. Springer Berlin Heidelberg, 1995.
- [34] Sun, O. N. E. "Grid Engine." *Administration and User's Guide, Sun Microsystems* (2002).
- [35] Amazon, E. C. "Amazon elastic compute cloud (Amazon EC2)." *Amazon Elastic Compute Cloud (Amazon EC2)* (2010).
- [36] Andreetto, Paolo, Sergio Androozzi, Giuseppe Avellino, Stefano Beco, Andrea Cavallini, Marco Cecchi, Vincenzo Ciaschini et al. "The gLite workload management system." In *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062007. IOP Publishing, 2008.
- [37] Taylor, Ian, Matthew Shields, Ian Wang, and Andrew Harrison. "The triana workflow environment: Architecture and applications." In *Workflows for e-Science*, pp. 320-339. Springer London, 2007.

- [38] Taylor, Ian, Matthew Shields, Ian Wang, and Andrew Harrison. "Visual grid workflow in Triana." *Journal of Grid Computing* 3, no. 3-4 (2005): 153-169.
- [39] Allen, Gabrielle, Tom Goodale, Thomas Radke, Michael Russell, Ed Seidel, Kelly Davis, Konstantinos N. Dolkas et al. "Enabling applications on the grid: A Gridlab overview." *International Journal of High Performance Computing Applications* 17, no. 4 (2003): 449-466.
- [40] Gong, Li. "JXTA: A network programming environment." *Internet Computing, IEEE* 5, no. 3 (2001): 88-95.
- [41] Alonso, Gustavo, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services*. Springer Berlin Heidelberg, 2004.
- [42] Allcock, William, Joe Bester, John Bresnahan, Ann Chervenak, Lee Liming, and Steve Tuecke. "GridFTP: Protocol extensions to FTP for the Grid." *Global Grid ForumGFD-RP 20* (2003).
- [43] Oinn, Tom, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver et al. "Taverna: a tool for the composition and enactment of bioinformatics workflows." *Bioinformatics* 20, no. 17 (2004): 3045-3054.
- [44] "Freefluo". Available: <http://freefluo.sourceforge.net/>
- [45] Fahringer, Thomas, Radu Prodan, Rubing Duan, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, Alex Villazon, and Marek Wiczorek. "ASKALON: A grid application development and computing environment." In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pp. 122-131. IEEE Computer Society, 2005.
- [46] Fahringer, Thomas, Radu Prodan, Rubing Duan, Jürgen Hofer, Farrukh Nadeem, Francesco Nerieri, Stefan Podlipnig et al. "Askalon: A development and grid computing environment for scientific workflows." In *Workflows for e-Science*, pp. 450-471. Springer London, 2007.
- [47] Fahringer, Thomas, Jun Qin, and Stefan Hainzer. "Specification of grid workflow applications with AGWL: an Abstract Grid Workflow Language." In *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 2, pp. 676-685. IEEE, 2005.
- [48] Altintas, Ilkay, Adam Birnbaum, Kim K. Baldridge, Wibke Sudholt, Mark Miller, Celine Amoreira, Yohann Potier, and Bertram Ludaescher. "A framework for the design and reuse of grid workflows." In *Scientific Applications of Grid Computing*, pp. 120-133. Springer Berlin Heidelberg, 2005.
- [49] Ludäscher, Bertram, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. "Scientific workflow

- management and the Kepler system." *Concurrency and Computation: Practice and Experience* 18, no. 10 (2006): 1039-1065.
- [50] Liu, Xiaojun, Jie Liu, Johan Eker, and Edward A. Lee. "Heterogeneous modeling and design of control systems." *Software-Enabled Control: Information Technology for Dynamical Systems* (2002): 105-122.
- [51] S. Kalayci and S. M. Sadjadi, "Pattern-based Decentralization and Run-time Adaptation Framework for Multi-site Workflow Orchestrations", In *Proceedings of The 2013 International Conference on Software Engineering and Knowledge Engineering*, Boston, USA, July 2013.
- [52] Kalayci, Selim, Gargi Dasgupta, Liana Fong, Onyeka Ezenwoye, and Seyed Masoud Sadjadi. "Distributed and Adaptive Execution of Condor DAGMan Workflows." In *SEKE*, pp. 587-590. 2010.
- [53] Kaiser, Alexander, Michael Probst, Holly A. Stretz, and Frank Hagelberg. "Aggregates of PCBM molecules: A computational study." *International Journal of Mass Spectrometry* 365 (2014): 225-231.
- [54] Wu, Jianhua, Anahita Ayasoufi, Jerzy Leszczynski, and Frank Hagelberg. "Geometric, Magnetic, and Adsorption Properties of Cross-Linking Carbon Nanotubes: A Computational Study." *The Journal of Physical Chemistry C* 117, no. 7 (2013): 3646-3652.
- [55] Pearlman, David A., David A. Case, James W. Caldwell, Wilson S. Ross, Thomas E. Cheatham III, Steve DeBolt, David Ferguson, George Seibel, and Peter Kollman. "AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules." *Computer Physics Communications* 91, no. 1 (1995): 1-41.
- [56] Berendsen, Herman JC, David van der Spoel, and Rudi van Drunen. "GROMACS: A message-passing parallel molecular dynamics implementation." *Computer Physics Communications* 91, no. 1 (1995): 43-56.
- [57] Plimpton, S., P. Crozier, and A. Thompson. "LAMMPS-large-scale atomic/molecular massively parallel simulator." *Sandia National Laboratories* (2007).
- [58] Phillips, James C., Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. "Scalable molecular dynamics with NAMD." *Journal of computational chemistry* 26, no. 16 (2005): 1781-1802.
- [59] Valiev, Marat, Eric J. Bylaska, Niranjan Govind, Karol Kowalski, Tjerk P. Straatsma, Hubertus JJ Van Dam, Dunyou Wang et al. "NWChem: a comprehensive

- and scalable open-source solution for large scale molecular simulations." *Computer Physics Communications* 181, no. 9 (2010): 1477-1489.
- [60] Skamarock, W. C., J. B. Klemp, and J. Dudhia. "Prototypes for the WRF (Weather Research and Forecasting) model." In *Preprints, Ninth Conf. Mesoscale Processes, J11–J15, Amer. Meteorol. Soc., Fort Lauderdale, FL. 2001.*
- [61] Michalakes, J., J. Dudhia, D. Gill, T. Henderson, J. Klemp, W. Skamarock, and W. Wang. "The weather research and forecast model: software architecture and performance." In *Proceedings of the 11th ECMWF Workshop on the Use of High Performance Computing In Meteorology*, vol. 25, p. 29. World Scientific, 2004.
- [62] Gneiting, Tilmann, and Adrian E. Raftery. "Weather forecasting with ensemble methods." *Science* 310, no. 5746 (2005): 248-249.
- [63] Selim Kalayci and S. Masoud Sadjadi. "Optimization Patterns for the Decentralized Orchestration of Parameter-Sweep Workflows". The IEEE International Conference on Cloud and Autonomic Computing (CAC 2014), Imperial College, London, September 2014.
- [64] Schlick, Tamar. *Molecular Modeling and Simulation: An Interdisciplinary Guide: An Interdisciplinary Guide*. Vol. 21. Springer, 2010.
- [65] van Gunsteren, Wilfred F., and Herman JC Berendsen. "Computer simulation of molecular dynamics: Methodology, applications, and perspectives in chemistry." *Angewandte Chemie International Edition in English* 29, no. 9 (1990): 992-1023.
- [66] Efstathiou, George, M. Davis, S. D. M. White, and C. S. Frenk. "Numerical techniques for large cosmological N-body simulations." *The Astrophysical Journal Supplement Series* 57 (1985): 241-260.
- [67] Cramer, Christopher J. *Essentials of computational chemistry: theories and models*. John Wiley & Sons, 2013.
- [68] Gentleman, Robert C., Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis et al. "Bioconductor: open software development for computational biology and bioinformatics." *Genome biology* 5, no. 10 (2004): R80.
- [69] Frankland, S. J. V., A. Caglar, D. W. Brenner, and M. Griebel. "Molecular simulation of the influence of chemical cross-links on the shear strength of carbon nanotube-polymer interfaces." *The Journal of Physical Chemistry B* 106, no. 12 (2002): 3046-3048.
- [70] Åqvist, Johan, Carmen Medina, and Jan-Erik Samuelsson. "A new method for predicting binding affinity in computer-aided drug design." *Protein engineering* 7, no. 3 (1994): 385-391.

- [71] Alonso, Hernan, Andrey A. Bliznyuk, and Jill E. Gready. "Combining docking and molecular dynamic simulations in drug design." *Medicinal research reviews* 26, no. 5 (2006): 531-568.
- [72] Scheraga, Harold A., Mey Khalili, and Adam Liwo. "Protein-folding dynamics: overview of molecular simulation techniques." *Annu. Rev. Phys. Chem.* 58 (2007): 57-83.
- [73] Hess, Berk, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. "GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation." *Journal of chemical theory and computation* 4, no. 3 (2008): 435-447.
- [74] Larson, Stefan M., Christopher D. Snow, and Michael Shirts. "Folding@ Home and Genome@ Home: Using distributed computing to tackle previously intractable problems in computational biology." (2002).
- [75] Leutbecher, Martin, and Tim N. Palmer. "Ensemble forecasting." *Journal of Computational Physics* 227, no. 7 (2008): 3515-3539.
- [76] Sakellariou, Rizos, and Henan Zhao. "A low-cost rescheduling policy for efficient mapping of workflows on grid systems." *Scientific Programming* 12, no. 4 (2004): 253-262.
- [77] Yu, Zhifeng, and Weisong Shi. "An adaptive rescheduling strategy for grid workflow applications." In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1-8. IEEE, 2007.
- [78] Lee, Kevin, Norman W. Paton, Rizos Sakellariou, Ewa Deelman, Alvaro AA Fernandes, and Gaurang Mehta. "Adaptive workflow processing and execution in pegasus." *Concurrency and Computation: Practice and Experience* 21, no. 16 (2009): 1965-1981.
- [79] Abramson, David, Blair Bethwaite, Colin Enticott, Slavisa Garic, and Tom Peachey. "Parameter space exploration using scientific workflows." In *Computational Science-ICCS 2009*, pp. 104-113. Springer Berlin Heidelberg, 2009.
- [80] Kacsuk, Péter, Krisztian Karoczkai, Gabor Hermann, Gergely Sipos, and József Kovács. "WS-PGRADE: Supporting parameter sweep applications in workflows." In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pp. 1-10. IEEE, 2008.
- [81] Deelman, Ewa, Dennis Gannon, and Matthew Shields. *Workflows for e-Science*. Springer-Verlag London Limited, 2007.
- [82] Yu, Jia, and Rajkumar Buyya. "A taxonomy of workflow management systems for grid computing." *Journal of Grid Computing* 3, no. 3-4 (2005): 171-200.

- [83] Van Der Aalst, Wil, and Kees Max Van Hee. *Workflow management: models, methods, and systems*. MIT press, 2004.
- [84] Yu, Jia, Rajkumar Buyya, and Kotagiri Ramamohanarao. "Workflow scheduling algorithms for grid computing." In *Metaheuristics for scheduling in distributed computing environments*, pp. 173-214. Springer Berlin Heidelberg, 2008.
- [85] Dumas, Marlon, and Arthur HM Ter Hofstede. "UML activity diagrams as a workflow specification language." In *<< UML >> 2001—The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pp. 76-90. Springer Berlin Heidelberg, 2001.
- [86] Li, Dingchao, and Naohiro Ishii. "Scheduling task graphs onto heterogeneous multiprocessors." In *TENCON'94. IEEE Region 10's Ninth Annual International Conference. Theme: Frontiers of Computer Technology. Proceedings of 1994*, pp. 556-563. IEEE, 1994.
- [87] Maheswaran, Muthucumaru, Shoukat Ali, H. J. Siegal, Debra Hensgen, and Richard F. Freund. "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems." In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pp. 30-44. IEEE, 1999.
- [88] Moreno, Rafael, and Ana B. Alonso-Conde. "Job scheduling and resource management techniques in economic grid environments." In *Grid Computing*, pp. 25-32. Springer Berlin Heidelberg, 2004.
- [89] Rotithor, Hemant G. "Taxonomy of dynamic task scheduling schemes in distributed computing systems." *IEE Proceedings-Computers and Digital Techniques* 141, no. 1 (1994): 1-10.
- [90] Yu, Jia, Rajkumar Buyya, and Kotagiri Ramamohanarao. "Workflow scheduling algorithms for grid computing." In *Metaheuristics for scheduling in distributed computing environments*, pp. 173-214. Springer Berlin Heidelberg, 2008.
- [91] Sakellariou, Rizos, and Henan Zhao. "A hybrid heuristic for DAG scheduling on heterogeneous systems." In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 111. IEEE, 2004.
- [92] Yu, Jia, and Rajkumar Buyya. "A budget constrained scheduling of workflow applications on utility grids using genetic algorithms." In *Workflows in Support of Large-Scale Science, 2006. WORKS'06. Workshop on*, pp. 1-10. IEEE, 2006.
- [93] Ibarra, Oscar H., and Chul E. Kim. "Heuristic algorithms for scheduling independent tasks on nonidentical processors." *Journal of the ACM (JACM)* 24, no. 2 (1977): 280-289.

- [94] Maheswaran, Muthucumaru, Shoukat Ali, H. J. Siegal, Debra Hensgen, and Richard F. Freund. "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems." In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pp. 30-44. IEEE, 1999.
- [95] Deelman, Ewa, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta et al. "Pegasus: A framework for mapping complex scientific workflows onto distributed systems." *Scientific Programming* 13, no. 3 (2005): 219-237.
- [96] Hwang, Soonwook, and Carl Kesselman. "Grid workflow: a flexible failure handling framework for the grid." In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*, pp. 126-137. IEEE, 2003.
- [97] Crichton, Daniel J., J. Steven Hughes, and Sean Kelly. "A Science Data System Architecture for Information Retrieval." In *Clustering and Information Retrieval*, pp. 261-298. Springer US, 2004.
- [98] Buyya, R and Murshed, M., "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing." *Concurrency and Computation: Practice and Experience (CCPE)*, s.l. : Wiley Press, November - December 2002, Issue 13-15, Vol. 14.
- [99] Karypis, G and Kumar, V., "Multilevel k-way Partitioning Scheme for Irregular Graphs." *J. Parallel Distrib. Comput.* , 1998, Issue 1, Vol. 48, pp. 96-129.
- [100] Hendrickson, B and Kolda, T.G., "Graph partitioning models for parallel computing." *Parallel Computing*, s.l. : Elsevier Science Publishers B. V., 2000, Issue 12, Vol. 26.
- [101] Simon, Horst D., "Partitioning of unstructured problems for parallel processing." *Computing Systems in Engineering*, 1991, Vol. 2.
- [102] Hendrickson, B and Leland, R. "An improved spectral graph partitioning algorithm for mapping parallel computations." *SIAM J. Sci. Comput.*, Vol 16, 1995.
- [103] Kumar, S., Das, S. K., and Biswas, R. 2002. Graph Partitioning for Parallel Applications in Heterogeneous Grid Environments. In *Proceedings of the 16th international Parallel and Distributed Processing Symposium (April 15 - 19, 2002)*. IEEE Computer Society, Washington, DC, 167.
- [104] Dick, R.P, Rhodes, D.L and Wolf, W., "TGFF: task graphs for free." Washington : IEEE Computer Society, 1998. *International Conference on Hardware Software Codesign*.

- [105] Brown, Duncan A., Patrick R. Brady, Alexander Dietz, Junwei Cao, Ben Johnson, and John McNabb. *A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis*. Springer, 2006.
- [106] Bharathi, Shishir, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. "Characterization of scientific workflows." In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pp. 1-10. IEEE, 2008.
- [107] Floyd, Sally, and Van Jacobson. "Random early detection gateways for congestion avoidance." *Networking, IEEE/ACM Transactions on* 1, no. 4 (1993): 397-413.
- [108] G. Kresse and J. Hafner. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B*, 47:558, 1993.
- [109] G. Kresse and J. Furthmüller. Efficient iterative schemes for ab initio total-energy calculations using a plane-wave basis set. *Phys. Rev. B*, 54:11169, 1996.

VITA

SELIM KALAYCI

- 1998-2002 B.S., Computer Engineering
Fatih University
Istanbul, Turkey
- 2004-2006 M.S., Computer Science
Florida International University
Miami, Florida
- 2007 -2014 Doctoral Candidate
Florida International University
Miami, Florida

PUBLICATIONS AND PRESENTATIONS

Bobroff, Norman, Liana Fong, Selim Kalayci, Yanbin Liu, Juan Carlos Martinez, Ivan Rodero, Seyed Masoud Sadjadi, and David Villegas. "Enabling interoperability among meta-schedulers." In *Cluster Computing and the Grid, 2008. CCGRID'08. 8th IEEE International Symposium on*, pp. 306-315. IEEE, 2008.

Kalayci, Selim, Onyeka Ezenwoye, Balaji Viswanathan, Gargi Dasgupta, S. Masoud Sadjadi, and Liana Fong. "Design and implementation of a fault tolerant job flow manager using job flow patterns and recovery policies." In *Service-Oriented Computing-ICSOC 2008*, pp. 54-69. Springer Berlin Heidelberg, 2008.

Kalayci, Selim, Gargi Dasgupta, Liana Fong, Onyeka Ezenwoye, and Seyed Masoud Sadjadi. "Distributed and Adaptive Execution of Condor DAGMan Workflows." In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering - SEKE 2010*, pp. 587-590, 2010.

Kalayci, Selim, and S. Masoud Sadjadi. "Pattern-based Decentralization and Run-time Adaptation Framework for Multi-site Workflow Orchestrations." In *Proceedings of the 25th International Conference on Software Engineering and Knowledge Engineering - SEKE 2013*, 2013.

Kalayci, Selim, and S. Masoud Sadjadi. "Optimization Patterns for the Decentralized Orchestration of Parameter-Sweep Workflows." In *Proceedings of the IEEE International Conference on Cloud and Autonomic Computing - CAC 2014*, 2014.