

TOWARDS COLLABORATIVE SCIENTIFIC WORKFLOW
MANAGEMENT SYSTEM

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Golam Mostaeen

©Golam Mostaeen, December/2018. All rights reserved.

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Or

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9
Canada

ABSTRACT

The big data explosion phenomenon has impacted several domains, starting from research areas to divergent of business models in recent years. As this intensive amount of data opens up the possibilities of several interesting knowledge discoveries, over the past few years divergent of research domains have undergone the shift of trend towards analyzing those massive amount data. Scientific Workflow Management System (*SWfMS*) has gained much popularity in recent years in accelerating those data-intensive analyses, visualization, and discoveries of important information. Data-intensive tasks are often significantly time-consuming and complex in nature and hence SWfMSs are designed to efficiently support the specification, modification, execution, failure handling, and monitoring of the tasks in a scientific workflow. As far as the complexity, dimension, and volume of data are concerned, their effective analysis or management often become challenging for an individual and requires collaboration of multiple scientists instead. Hence, the notion of *Collaborative SWfMS* was coined — which gained significant interest among researchers in recent years as none of the existing SWfMSs directly support real-time collaboration among scientists.

In terms of collaborative SWfMSs, consistency management in the face of conflicting concurrent operations of the collaborators is a major challenge for its highly interconnected document structure among the computational modules — where any minor change in a part of the workflow can highly impact the other part of the collaborative workflow for the datalink relation among them. In addition to the consistency management, studies show several other challenges that need to be addressed towards a successful design of collaborative SWfMSs, such as sub-workflow composition and execution by different sub-groups, relationship between scientific workflows and collaboration models, sub-workflow monitoring, seamless integration and access control of the workflow components among collaborators and so on.

In this thesis, we propose a locking scheme to facilitate consistency management in collaborative SWfMSs. The proposed method works by locking workflow components at a granular attribute level in addition to supporting locks on a targeted part of the collaborative workflow. We conducted several experiments to analyze the performance of the proposed method in comparison to related existing methods. Our studies show that the proposed method can reduce the average waiting time of a collaborator by up to 36% while increasing the average workflow update rate by up to 15% in comparison to existing descendent modular level locking techniques for collaborative SWfMSs. We also propose a role-based access control technique for the management of collaborative SWfMSs. We leverage the *Collaborative Interactive Application Methodology (CIAM)* for the investigation of role-based access control in the context of collaborative SWfMSs. We present our proposed method with a use-case of Plant Phenotyping and Genotyping research domain.

Recent study shows that the collaborative SWfMSs often different sets of opportunities and challenges. From our investigations on existing research works towards collaborative SWfMSs and findings of our prior two studies, we propose an architecture of collaborative SWfMSs. We propose — *SciWorCS* — a Collaborative Scientific Workflow Management System as a proof of concept of the proposed architecture; which is the

first of its kind to the best of our knowledge. We present several real-world use-cases of scientific workflows using SciWorCS. Finally, we conduct several user studies using SciWorCS comprising different real-world scientific workflows (i.e., from *myExperiment*) to understand the user behavior and styles of work in the context of collaborative SWfMSs. In addition to evaluating SciWorCS, the user studies reveal several interesting facts which can significantly contribute in the research domain, as none of the existing methods considered such empirical studies, and rather relied only on computer generated simulated studies for evaluation.

ACKNOWLEDGEMENTS

First of all, I would like to acknowledge my indebtedness and render my warmest thanks to my respected supervisor Dr. Chanchal K. Roy for his constant guidance, advice, motivation and extraordinary patience during my thesis works. I express my heartiest gratitude to Dr. Banani Roy who made this work possible. Her valuable time, constant guidance and encouragement have been invaluable throughout all stages of my thesis. In addition to their expert guidance in conducting my research works, I got both of them beside me in all my difficult stages of my life during the entire time span. Both of them have done so much for me than I could ever give them the thanks for.

I would like to express my special appreciation and thanks to Dr. Kevin A. Schneider. He has been extremely helpful and supportive to me along with providing his invaluable advice throughout the entire period of my thesis.

I thank the anonymous reviewers for their valuable comments and suggestions in improving the papers produced from this thesis. I would like to thank Dr. Debajyoti Mondal, Dr. Khan A. Wahid and Dr. Derek Eager, for their willingness to take part in the advisement and evaluation of my thesis work.

Special thanks to all of the members of the Software Research Lab for their extreme support and important feedback on my work. In particular, I would like to thank Rayhan Ferdous, Muhammad Asaduzzaman, Kawser Wazed Nafi, Amit Kumar Mondal, Masud Rahman, Debasish Chakroborti, CM Khaled Saifullah, Shamima Yeasmin and Judith Islam.

I am grateful to the Department of Computer Science of the University of Saskatchewan for their generous financial support through scholarships, awards, and bursaries that helped me to concentrate more deeply on my thesis work. I would like to render my warmest thanks to all the staffs of the Department for their constant support. In particular, I would like to thank Gwen Lancaster, Findlay Sophie, and Heather Webb.

I would especially like to thank all my friends and seniors who have been extremely helpful and encouraging throughout the period of my thesis work. In particular, I would like to thank Monzurul Arash, Nazmul Arnob, Md. Sami Uddin and Md. Aminul Islam.

I am extremely grateful to my family—my father MD. Ismail Dewan, my mother MOST. Mosfiqa Begum, my elder brother Golam Mostofa, my sister-in-law Anika Zaman and my beloved nephew Absi—for their unconditional support, encouragement, inspiration, and love at every stage of my life. Without their endless sacrifice, I would not have come this far.

I dedicate this to my family; my father MD. Ismail Dewan, my mother MOST. Mosfiqa Begum, my elder brother Golam Mostofa, my sister-in-law Anika Zaman and my beloved nephew Absi.

CONTENTS

Permission to Use	i
Abstract	ii
Acknowledgements	iv
Contents	vi
List of Tables	ix
List of Figures	x
List of Abbreviations	xii
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Our Contribution	4
1.3.1 Fine Grained Locking Scheme for Consistency Management	4
1.3.2 Role Based Access Control for Collaborative SWfMSs	4
1.3.3 SciWorCS: Towards a Collaborative SWfMS	5
1.3.4 Usability Study	5
1.4 Publications	5
1.5 Thesis Outline	6
2 BACKGROUND	7
2.1 Scientific Workflows	7
2.1.1 Life Cycle of A Scientific Workflow	8
2.2 Scientific Workflow Management Systems	8
3 GRANULAR ATTRIBUTE LEVEL LOCKING FOR CONSISTENCY MANAGEMENT IN COLLABORATIVE SCIENTIFIC WORKFLOW COMPOSITION	10
3.1 Motivation	10
3.2 Background: Consistency Management in Collaborative SWfMSs	12
3.3 Empirical Study on Modern Scientific Workflow Collaboration: From the Perspective of Existing Locking Schemes	15
3.3.1 Study on Modern Workflow Dependency Degree, ϕ and its Impacts on Collaboration	15
3.3.2 Study on Modern Workflow Module Structure	15
3.3.3 Impacts of Module Attributes on Collaboration	18
3.4 Proposed Method	18
3.4.1 Fine-Grained Workflow Component Locking for Workflow Collaboration	18
3.4.2 Lock Management Algorithms	21
3.5 Experiments and Evaluations	25
3.5.1 Implementation Details	25
3.5.2 Experimental Setup	25
3.5.3 Study on Average Waiting Time and Throughput	27
3.5.4 Study on Workflow Composition Time and Efficiency	28
3.5.5 Performance Study on Varying Node Access Requests Topology	29
3.5.6 Analysis Study in terms of Varying Workflow Tree Structures	30
3.6 Threats to the Validity	31

3.7	Related Works	32
3.8	Conclusion	34
4	MODELING A COLLABORATIVE SCIENTIFIC WORKFLOW MANAGEMENT SYSTEM USING CIAM: A CASE-STUDY WITH PLANT PHENOTYPING AND GENOTYPING	35
4.1	Motivation	35
4.2	Related Works	37
4.2.1	Frameworks or Tools Supporting Pipeline Compositions	38
4.2.2	Tools Supporting API Testing Environment	38
4.3	A Motivating Example Scenario	39
4.4	Proposed Approach	40
4.4.1	User Roles and Sociogram	40
4.4.2	Responsibilities and Process Modeling	42
4.4.3	Tool Design	45
4.4.4	Pipeline Composition	48
4.5	Evaluation	50
4.5.1	Case Study: Collaborative Development and Management of a Pipeline	50
4.5.2	User Study	54
4.6	Conclusion	55
5	SCIWORCS- TOWARDS A COLLABORATIVE SCIENTIFIC WORKFLOW MANAGEMENT SYSTEM	56
5.1	Motivation	56
5.2	Related Works	57
5.3	SciWorCS Architecture	59
5.3.1	Toolbox: Set of Reusable Computational Steps	60
5.3.2	DAG Formulation for a Data Analysis Workflow	61
5.3.3	Collaborative Composition	62
5.3.4	Job Management and Execution	63
5.4	Implementation Details	63
5.5	SciWorCS Task Specific User Interfaces	64
5.5.1	Plugging in New Tools to SciWorCS Toolbox	64
5.5.2	Collaborative Workflow Composition	68
5.5.3	Tools for Aiding Collaborative Workflow Composition	68
5.5.4	Data Visualization	69
5.6	SciWorCS Usage Examples	70
5.6.1	QC Report of FastQ File with FastQC (Bioinformatics)	70
5.6.2	Machine Learning Based Clone Validation (Software Repository Analysis)	71
5.6.3	Machine Learning Based Clone Validation Approach	74
5.6.4	Code Clone Detection (Software Repository Analysis)	86
5.7	Conclusion	87
6	UNDERSTANDING THE USER BEHAVIOR FOR COLLABORATIVE DATA ANALYSIS	89
6.1	Motivation	89
6.2	Related Works	92
6.2.1	CSCW in Aiding Scientific Experiments	92
6.2.2	Towards Collaborative Data Analysis	92
6.3	Implementation Details	93
6.4	Experimental Studies and Results	95
6.4.1	Experimental Setups	95
6.4.2	Study 1: Collaborative Composition Patterns	95
6.4.3	Study 2: Collaborative Dataflow Problem Solving and Convergence on a Plan	100
6.5	Result Discussion	104

6.6	Conclusion	105
7	CONCLUSION	107
7.1	Summary	107
7.1.1	Fine Grained Attribute Level Locking Scheme	108
7.1.2	Role Based Access Control for Collaborative SWfMSs	108
7.1.3	SciWorCS: Proposed Architecture Towards a Collaborative SWfMS	108
7.1.4	Usability Study	109
7.2	Future Work	109
7.2.1	Collaborative Provenance Models	109
7.2.2	Studying More CSCW Techniques	110
7.2.3	Deadlock Awareness among Collaborators	110
7.2.4	Studying Collaborative Task Scheduling	110
7.2.5	Adaptation of SciWorCS in Source Code Repository Analysis	111
	References	112
	Appendix A CONSISTENCY MANAGEMENT SIMULATION STUDY	124
A.1	Code Snippets	124
	Appendix B USER STUDY: USER MANUAL AND STUDY DESIGN	128
B.1	SciWorCS	128
B.2	Introduction to SciWorCS Editor	128
B.3	SciWorCS Workflow Composition	128
B.3.1	Prerequisites	128
B.3.2	Task Description	128
B.3.3	Toolbox	129
B.4	Collaborative Problem Solving	130
B.4.1	Prerequisites	130
B.4.2	Task Description	130
B.4.3	Toolbox	131
B.5	Collaborative Data Analysis using Machine Learning Classifiers and Statistical Tools	132
B.5.1	Task Description	132
B.5.2	Dataset	132
B.5.3	Toolbox	132
B.6	NASA-TLX Questionnaires	133
B.7	Log Snippet	134

LIST OF TABLES

3.1	Information of some arbitrary scientific workflows from <i>myExperiment</i> [44] ¹	17
3.2	Some of the Galaxy Tools with their Attributes Count	17
3.3	Considered Primitive Workflow Operations.	26
3.4	Primitive Operations For Component Access in Collaborative Workflow Composition Environment.	27
5.1	Selected Features Based on Distribution Analysis	84
5.2	Classification Accuracy Comparison for different Machine Learning Models.	85
6.1	Dataflow Structure Updates Operations.	95
6.2	Primitive Operations For Component Access and User Interaction in Collaborative Data Analysis.	96
6.3	Considered Arbitrary Scientific Workflows from <i>myExperiments</i> [44] for the Study.	97
6.4	Publicly Available Dataset - ' <i>Titanic: Machine Learning from Disaster</i> ' as Collected from <i>Kaggle</i> [95].	102
B.1	First Math Task (1 of 2)	132
B.2	Second Math Task (2 of 2)	132
B.3	Publicly Available Dataset - ' <i>Titanic: Machine Learning from Disaster</i> ' as Collected from <i>Kaggle</i>	133

LIST OF FIGURES

3.1	Workflow Version Conflict	14
3.2	Collaborative Workflow Consistency Management via Turn Based Floor Control Technique	14
3.3	Part of a shared workflow in <i>myExperiment</i> [44] for collaboration (workflow id=4921, as noted in June, 2018)	16
3.4	Collaborative Workflow Consistency Management via attribute level granular concurrency control Technique	20
3.5	Sub-workflow Locks in Collaborative Workflow Composition	26
3.6	Average Waiting Time and Throughput Comparison of Locking Algorithms for Collaborative Workflow Composition.	28
3.7	Comparative Study on Collaborative Workflow Composition Time and Efficiency of Locking Schemes.	29
3.8	Algorithm Efficiency on ‘Best’ and ‘Worst’ Case Scenario of Collaborative Node Access Request Topology.	30
3.9	Performance Analysis in Terms of Varying Workflow Tree Structures	32
4.1	Sociogram for the proposed Collaborative Plant Phenotyping and Genotyping.	43
4.2	Participation Table for the Proposed Collaborative System.	43
4.3	Responsibilities Model for Admin.	44
4.4	Responsibilities Model for Data Specialist.	44
4.5	Responsibilities Model for Tool Developer.	45
4.6	Responsibilities Model for Pipeline Composer.	46
4.7	Responsibilities Model for Plant Scientist.	46
4.8	Process Model for collaborative plant Phenotyping and Genotyping.	47
4.9	Abstraction Layer on modular Tools for collaborative use.	49
4.10	Collaborative Pipeline Design using modularized tools.	49
4.11	Collaborative Task Modeling for Pipeline Composition.	50
4.12	Collaborative Task Modeling for Tool Development.	51
4.13	Usage example for collaborative work by the proposed method for Phenotyping.	51
4.14	Process Model for collaborative plant Phenotyping and Genotyping.	53
5.1	High-Level Architecture of SciWorCS.	59
5.2	SciWorCS DAG Formulation for Workflows.	61
5.3	User Interface Overview of SciWorCS.	64
5.4	SciWorCS tool plugin interface.	67
5.5	Locking Schemes for Consistency Management in Collaborative Composition (A subset of the workflow nodes with corresponding Lock states).	68
5.6	Example collaborative tools for aiding collaborative data analysis process.	69
5.7	Dataset Feature Distribution - An Example Visualization for the Classification.	70
5.8	FastQC quality measures on example sequence file.	72
5.9	High-level Workflow for Machine Learning based Code Clone Validation	74
5.10	Sample feature extraction workflow for machine leaning based code clone validation.	80
5.11	Histogram of Code Fragment Line Differences.	81
5.12	Histogram of Syntactical Similarity by Token (Type 1 Norm.)	82
5.13	SciWorCS workflow for clone classification with different machine learning classifiers.	85
5.14	Code Clone Detection Workflow in SciWorCS.	88
6.1	High-level Architecture of SciWorCS (e.g., as details presented in Chapter 5)	91
6.2	Prototype Implementation of Collaborative Data Analysis Framework	94
6.3	Collaboration Process for Consistency Management Handling Concurrent Conflicting Operations	94
6.4	Collaborative Composition Work Patterns	99

6.5	NASA-TLX workload for collaborative composition in terms of different locking schemes . . .	100
6.6	Collaborators' Engagement and Contribution in Collaborative Data Analysis.	104
B.1	SciWorCS Editor for Collaborative Scientific Workflow Composition	129
B.2	Sample workflow for collaborative composition practice	129
B.3	Paired-end reads assembly after FastQ groomer using a Migale modified version of Velvet tool.	130
B.4	Workflow used when applying the CPB2012 Basic Protocol 3; Peaks for ChIP-seq data using MACS14.	131
B.5	Reference Workflow for the Task	131

LIST OF ABBREVIATIONS

LOF	List of Figures
LOT	List of Tables
SWfMS	Scientific Workflow Management System
NASA-TLX	The NASA Task Load Index
SVN	Subversion
CVS	Concurrent Versions System
CSCW	Computer-Supported Cooperative Work
CAD	Computer Aided Design
SAM	Sequence Alignment Map
BAM	Binary Alignment Map
DAG	Directed Acyclic Graph
mLOCK	Descendant Module Lock
aLock	Attribute Level Lock
VRE	Virtual Research Environment
VL	Virtual Laboratory
CVW	Collaborative Virtual Whiteboard
CIAN	Collaborative Interactive Applications Notation
CIAM	Collaborative Interactive Application Methodology

1 INTRODUCTION

We provide a short introduction of the thesis in this chapter. First, in Section 1.1 we present a general motivation of the thesis. We then state problems and our contributions to the problems in Section 1.2 and Section 1.3 respectively. In Section 1.4, we list the published and prepared papers for submissions from this thesis. Finally, in Section 1.5 we provide an outline of the remaining chapters of the thesis.

1.1 Motivation

A **Scientific Workflow Management System** (*SWfMS*) (e.g., as discussed in details in Chapter 2) automates the process of life cycle phases — composition, deployment, execution and analysis of a scientific workflow [115, 111]. The generation of the sheer amount of heterogeneous data on a daily basis by different areas of modern science has influenced many of the disciplines in moving towards data and information driven analysis and discoveries [115]. *SWfMSs* have gained much popularity in the recent years in accelerating those data-intensive analysis, visualization, and discoveries of important information. Data-intensive tasks are often significantly time-consuming and complex in nature and hence *SWfMSs* are designed to efficiently support the specification, modification, execution, failure handling, and monitoring of the tasks in a scientific workflow [111]. As far as the complexity, dimension, and volume of data are concerned, their effective analysis or management often become challenging for an individual and require collaboration of multiple scientists instead [201]. For example, in a similar study Zhang et al. [201] referred the *Large Synoptic Survey Telescope (LSST)* [113] experiment that demands a collaboration of around 1800 scientists and engineers for a complete analysis process. Besides, some scientific domains essentially require collaboration as they are highly correlated among multiple research disciplines [201, 199]. For example, *Plant Genotyping and Phenotyping* is one of such emerging research areas that requires significant collaboration among scientists of multiple domains and expertise, such as *Image Processing* for Phenotyping, *Bioinformatics* for Genotyping, *Plant Science* for the investigation of any interesting correlation between Phenotyping and Genotyping and so on.

Although a number of *SWfMSs* have been developed and proposed over the last decade, such as Taverna [140], Galaxy [66], Kepler [115], Pegasus [47], VisTrails [31], Triana [181], VIEW [109], none of them directly support collaborative works among multiple users; hence for any collaboration, users need to follow several time consuming manual steps [199, 201]. For example, for a collaborative design of a scientific workflow, a user first composes a part of a workflow (e.g., a sub-workflow), exports it from the local workflow engine and

shares it with a collaborator for possible updates on the sub-workflow. This manual back and forth process for collaboration is often very time consuming, does not support real-time editing and often impractical as the collaborating group size increases in size over time [199, 164, 165, 201, 56].

Recent studies show that the scientific experiments can be significantly accelerated with the aid of virtual environments for collaborative research, also commonly referred to as *Virtual Research Environment (VRE)* or *Virtual Laboratory (VL)* [41, 91, 68, 201]. Several recent studies on scientific collaboration, reveals the increasingly global, multipolar and networked nature of modern scientific researches [194, 32]—that demand towards collaborative virtual research environment [91, 68]. For example, in this context Candela et al. [32] reported that - ‘... *this trend calls for innovative, dynamic, and ubiquitous research supporting environments where scattered scientists can seamlessly access data, software, and processing resources ...*’.

Adapting the similar concepts of collaboration in terms of SWfMS, hence have gained significant focus in recent years among the researchers [199, 167]. Several studies present the motivations and possibilities that the collaborative SWfMSs can open up for scientific data analysis [114, 151, 200, 199, 165, 164, 201, 167, 166, 56, 78].

1.2 Problem Statement

- **Problem Statement #1** (*Consistency Management in Collaborative Workflow Composition*): One of the most important challenges for any real-time collaborative system is the consistency management of the shared objects in the face of conflicting operations by the collaborators [173, 174]. Because, in a collaborative editing system, the concurrent operations on the same shared object might create several conflicting states at any given time frame. Generally, different version controlling techniques, such as SVN - Subversion, CVS - Concurrent Versions System and so on are widely used for conflict resolution of the unstructured document collaborative systems, such as, collaborative Text Document Editing [144], collaborative Computer Aided Design (CAD) [37], object-based collaborative Graphics Editing systems [173], collaborative bitmap editing system [60] and so on. Unlike these documents, the scientific workflows are more structured where one module can be highly dependent on another due to dataflow relation in between them [201]. Even any minor changes in any part of a workflow, can significantly impact the other part of the collaborative workflow in execution and data manipulation [56, 55, 201, 199, 78]- which often make the problem notably different than that of unstructured document collaborative systems, such as text or graphics editing systems [130, 201, 165]. Hence, consistency management in the face of conflicting concurrent operations in collaborative workflow composition has been one of the important research problems in this domain. The design of a ‘*locking scheme*’ for facilitating workflow consistency need to answer several questions, such as —*How does the locking scheme can ensure least redundant locks on workflow components?, How does the request topology affect the performance of the locking scheme?, What is the effect of the locking scheme on overall average waiting time?, How different*

workflow DAG affects the performance of the locking scheme?

- **Problem Statement #2** (*Collaboration Modeling with Varying User Roles*): Studies show that the successful design of collaborative or groupware systems can often be more challenging than that of the single-user oriented systems for different requirements such as, modeling cooperative procedures, roles of multidisciplinary users, spaces for sharing information and so on — which need to be addressed [122, 196]. The similar requirements for varying roles of users have also been presented in the context of collaborative SWfMSs [17, 36, 117]. A scientific workflow is comprised of and linked with different components, such as workflow module ports, executable, datalink, data products, provenance information and so on [17, 201, 56]. Managing the usability and accessibility of the workflow components among the collaborating parties are important to guarantee their security and easier access [17]. *How to manage access to the workflow components while enabling collaboration among individuals or research groups?, How data products, workflow modules or provenance information are shared among collaborating groups?, How different interfaces for such varying user roles are modeled?* — are critical questions that need to be addressed in terms of collaborative SWfMSs design.
- **Problem Statement #3** (*Addressing Collaborative Requirements in SWfMSs*): Zhang [199, 201] presented that, often there can have several sub-groups working collaboratively on different sub-workflows of an entire scientific workflows. A collaborative SWfMS needs to handle such independent sub-workflow execution and backdoor communication among the sub-group collaborators [201]. Lu et al. investigated the possible challenges towards a collaborative SWfMSs, such as maintaining a collaborative provenance model, the relationship between scientific workflows and collaboration models [114]. Hence, from these existing studies a collaborative SWfMSs need to consider: *How to schedule the sub-workflow execution by different collaborating individuals or groups?, How to facilitate the collaborative data analysis with the aid of CSCW technologies?*
- **Problem Statement #4** (*User Behavior in Collaborative SWfMSs Setups*): In the context of CSCW, understanding the user behavior, styles of work and so on is important towards designing effective and efficient system collaborative systems [142]. Several locking schemes have been proposed in recent years towards collaborative SWfMSs [56, 199, 201, 165, 164, 167]. For the evaluation, these studies, in general, conducted several computer-generated simulation experiments. While these simulation results depict the performance of the proposed methods; however, to the best of our knowledge none of them considered the usability analysis of their methods in terms of real-world setups. *What are styles of works collaborators adapt while data analysis with SWfMSs?, How the data analysis task affects the collaborative data analysis process?, What confounding factors influence the styles of works for collaborative workflow composition?* — are some interesting findings that can significantly contribute towards the research domain.

1.3 Our Contribution

Focusing on the above research problems in terms of collaborative SWfMSs, our studies make four major contributions. Here in this section we briefly present our contribution of the study.

1.3.1 Fine Grained Locking Scheme for Consistency Management

With an attempt to facilitating the consistency management, researchers have proposed different locking techniques of the workflow components. The locking techniques - where only a single collaborator gets exclusive *Write* access to a part of the workflow to facilitate the consistency management [165, 201] by preventing concurrent conflicting operations on the same workflow component. These locking schemes hence allow collaborators to concurrently work on different sub-workflows of the shared workflow. For example, some related studies are: the entire workflow object locking in turns [199], descendant modules locking [56, 201], multiple variants of module locking [165, 164] and so on (details in Chapter 3). However, as all of these existing locking methods work on workflow modular levels, the *collaboration concurrency* [173] is dropped significantly as the workflow complexity grows over time with an increased number of modules and complicated datalink relations among them [56, 201, 166, 199].

In order to improve the collaboration concurrency in terms of collaborative SWfMSs, we propose a novel approach by further extending the workflow module locking to a more fine-grained attribute level. A large amount of redundant descendent workflow module locks imposed by the existing methods can be significantly avoided or minimized by the proposed method. We conducted several experimental studies to evaluate the proposed method against existing locking schemes in terms of collaborative SWfMSs. Our studies show that the proposed method can reduce the average waiting time of a collaborator by up to 36% while increasing the average workflow update rate by up to 15% in comparison to existing descendent modular level locking techniques. We also conduct several evaluation studies on varying request topologies, workflow DAG structures and so on, to answer the research questions presented in *Problem Statement #1* of Section 1.2. Our proposed fine-grained locking scheme, in such setups, demonstrates promising results in comparison to the existing locking schemes. We present the details of our studies in Chapter 3.

1.3.2 Role Based Access Control for Collaborative SWfMSs

While the consistency management is one of the primary requirements of a collaborative system [173], such collaborative systems involving multiple disciplines often need to consider the role-based access controls to manage or orchestrate the entire process of collaboration [72, 124, 202]. As facilitating the collaboration among multiple research disciplines is one of the important motivations of collaborative SWfMSs [201, 114, 199, 78], we studied the concept of role-based access control in the context of collaborative SWfMSs. We followed the *Collaborative Interactive Application Methodology (CIAM)* [124] for the investigation of role-based access control in the context of collaborative SWfMSs. We used *Collaborative Interactive Application*

Notation (CIAN) [123] for the study of role-based access control in terms of collaborative SWfMSs. We present our study on role-based access control with a use-case of collaborative Plant Phenotyping and Genotyping research domain to answer the research questions presented in *Problem Statement #2* of Section 1.2. Chapter 4 presents the details of the studies.

1.3.3 SciWorCS: Towards a Collaborative SWfMS

From our findings and investigations of the prior two studies, we propose an architecture for collaborative SWfMSs. Studies show that Collaborative SWfMSs often have a different set of challenges and requirements in contrast to the single user based SWfMSs [114, 199]. For example, Zhang [199, 201] presented that, often there can have several sub-groups working collaboratively on different sub-workflows of entire scientific workflows. A collaborative SWfMS needs to handle such independent sub-workflow execution and *backdoor communication* among the sub-group collaborators [199]. Lu et. al investigated the possible challenges towards a collaborative SWfMSs, such as maintaining a collaborative provenance model, the relationship between scientific workflows and collaboration models, and so on [114]. For our proposed architecture we tried to address such studied challenges in the context of collaborative SWfMSs. As a proof of concept of the proposed architecture, we developed- *SciWorCS* -a Collaborative **Scientific Workflow Management System**. We present different experimental use-cases for the evaluation of the proposed architecture. We present several use-case studies to answer the research questions regarding the sub-workflow execution, usage of CSCW technologies, data analysis and so on (e.g., as presented in *Problem Statement #3* of Section 1.2) in the context of collaborative SWfMSs. We present the details of the studies in Chapter 5.

1.3.4 Usability Study

We conducted several user studies to understand the user behavior and styles of work in the context of collaborative SWfMSs. In our studies, we considered different recent existing proposed method for consistency management and collaboration in SWfMSs. We leverage SciWorCS for the empirical studies on collaborative data analysis in terms of SWfMSs. In our study, participants were asked for some data analysis tasks, such as real-world workflows from *myExperiment* [44], building machine learning classification models for a given dataset and so on collaboratively. Our studies reveal several interesting findings and answer different research questions as presented in *Problem Statement #4* of Section 1.2 — which have a significant contribution towards a better design of collaborative SWfMSs. We present the details of the our conducted studies in Chapter 6.

1.4 Publications

Following is a list of published and prepared papers for submission (e.g., with co-authors) from the thesis works:

- *Golam Mostaeen*, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. Fine-grained attribute level locking scheme for collaborative scientific workflow development. In *Services Computing (SCC), 2018 IEEE International Conference on*, pages 273-277. IEEE, 2018.
- *Golam Mostaeen*, Jeffrey Svajlenko, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. On the use of machine learning techniques towards the design of cloud based automatic code clone validation tools. In *Source Code Analysis and Manipulation, 2018. SCAM 2018. 18th IEEE International Working Conference on*, pages 155-164. IEEE, 2018.
- *Golam Mostaeen*, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. Collaborative Data Analysis With Scientific Workflow Management Systems in Theory vs Reality. *IEEE International Conference on Services Computing, 2019 (to be submitted)*.
- *Golam Mostaeen*, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. Consistency Management In Real-Time Collaborative Scientific Workflow Composition By Granular Attribute Level Locking. *IEEE Journal - Transactions on Services Computing (to be submitted)*.
- *Golam Mostaeen*, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. SciWorCS: Towards Collaborative Scientific Workflow Management Systems. *IEEE Journal - Transactions on Services Computing (to be submitted)*.
- Debasish Chakroborti, Banani Roy, Amit Mondal, *Golam Mostaeen*, Ralph Deters, Chanchal K. Roy, Kevin A. Schneider. A Data Management Scheme for Micro-Level Modular Computation-intensive Programs in Big Data Platforms. In *International Symposium on Big Data Management and Analytics, 2018 (accepted in the Conference, the extended book chapter has also been reviewed and notified as accepted)*.

1.5 Thesis Outline

In Chapter 2 we discuss some background on the concepts of *Scientific Workflows* and *Scientific Workflow Management Systems (SWfMSs)*. We present our proposed locking scheme for consistency management in Chapter 3. Chapter 4 focuses and presents discussion on the role based access control techniques in the context of collaborative SWfMSs. In Chapter 5 we present our proposed method for collaborative SWfMSs and demonstrate the developed tool *SciWorCS* in terms of several use-cases as a proof of concept. To understand the real world impact of collaborative SWfMSs in terms of data analysis, we considered different user studies. We present the details of the study in Chapter 6. Finally, in Chapter 7, we conclude with overall summary of the thesis and discussion of some future research directions.

2 BACKGROUND

In this chapter, we provide a short discussion of background and technical preliminaries of the thesis. In Section 2.1 of the chapter, we first discuss on *Scientific Workflows* with their general life cycle. We then present a general overview of *Scientific Workflow Management Systems (SWfMSs)* in Section 2.2 of the chapter.

2.1 Scientific Workflows

A workflow is a facilitation or automation of a process as a part or whole [83, 12] during which the targeted data are passed from one participant (e.g., human or computer) to another for certain actions or processing as per some set of rules [111, 83, 12]. Workflows were initially adopted by the business community to accelerate the overall of a business process [111, 83]. Human and computer both are common participants for a business workflow step which are linked together forming the whole workflow. Business workflows usually take advantage of traditional procedural programming language for the computer-based workflow steps [111]. The workflows mainly follow two types of architectural approaches: *Service Orchestration* and *Service Choreography* [12]. Several process modeling languages and techniques thus have been proposed and developed by the community [12], such as: *Business Process Execution Language (BPEL)* [6], *Yet Another Workflow Language (YAML)* [187], *WfMOpen* [110] - an open source workflow engine, *Web Service Choreography Description Language (WS-CDL)* [100] and so on.

Scientific workflows, on the other hand, operate on a more abstract level, and used for modeling and performing scientific experiments on a dataset [111, 12]. Unlike the business workflows, the workflow steps for different data processing are entirely carried out by machines [111]. In terms of scientific workflows, the workflow steps are more commonly referred to *Computational Modules*. A computational module is responsible for some independent tasks of data manipulation and processing. The modules define the associated input data format, the data processing methods and the corresponding output data format [111]. The execution behavior of the modules can also be configured as per the given data analysis task. Computational modules are linked with one another forming *Directed Acyclic Graph (DAG)* of the modules representing the dataflow relations among the modules [164, 56, 111, 12]. Based on the functionalists of a set of computational modules, the whole DAG can often be categorized into multiple sub-graphs, -which are also commonly referred as *sub-workflows*. Scientific workflows generally follow data flow oriented architecture for its execution [111, 12]. That is, a workflow module starts its execution only when its corresponding required input datasets are

available.

2.1.1 Life Cycle of A Scientific Workflow

From composition to the analysis the life cycle of a scientific workflow can be broadly categorized into four phases [46, 111, 69]:

1. *Composition Phase* [111, 46, 120]: In this phase, the creation of the workflow comprising the modular abstraction is done. The workflow modules are configured as per the requirement of the data analysis task, which is then linked with one another forming the complete workflow. Such workflow DAG composition can be done both from textual or Graphical User Interface (GUI) [111]. However, the GUI is more common for the task, as they provide more intuitive interfaces with a visual representation of the workflow steps [3]. The composition is facilitated by one or more toolbox(es) [3]. A toolbox, T is a set of different computational modules, m , i.e., $m_1, m_2, \dots, m_n \in T$. For a given data analysis problem, the workflow composition reuses the modules from the toolboxes. For example, $w = m_2 \rightarrow m_1 \rightarrow m_4$, is a simple linear workflow definition of abstract level- comprising of modules from the toolbox.
2. *Deployment Phase* [111, 46]: While the composition phase defines the abstract level of the workflow, in the deployment phase the composed workflow is prepared for the execution with required setups. The preparation includes the association of corresponding source codes with the abstract modules, setting up a data source, configuring the execution environment (e.g., cluster or local instance setups) and so on [111, 3].
3. *Execution Phase*: The deployed workflow is executed with associated input data in this phase to produce the corresponding output of the workflow [111, 46, 120].
4. *Analysis Phase*: Finally, the output data is analyzed as per some research hypotheses in this phase. Workflow data provenance, output data visualization and so on are some examples of the Analysis phase [111].

2.2 Scientific Workflow Management Systems

A Scientific Workflow Management System (SWfMS) automates the process of life cycle phases- composition, deployment, execution and analysis (e.g., as discussed in Section 2.1.1) of a scientific workflow [115, 111]. That is, a SWfMS handles and manages the overall execution of scientific workflows. They provide GUI or command line based interfaces for the composition of the workflow in abstract level [186, 3]. SWfMSs provides some automatic facilities for associating the corresponding codes against those high-level representation of workflow [111] for the deployment phase. SWfMSs also provide supports for several other configurations in the deployment phase, such as input data source selection, execution environment setup (e.g., cluster or

local instance) and so on. SWfMSs also provides the workflow runner for the execution phase. Typically, for running the tasks or jobs of the workflows, SWfMSs contain a job manager that is responsible for workflow task scheduling, running and monitoring [3, 66, 111]. For the analysis phase, SWfMSs provides visualization functionality and different meta data information, such as workflow provenance data [111, 59]. The provenance data captures the different histories of the workflow execution, such as the origin of data, completion status of the workflow modules, intermediate states of the dataset and so on [59, 111]. The examples of some popular modern scientific workflow management systems are: *Taverna* [141], *Galaxy* [66], *Kepler* [115], *Pegasus* [47], *VisTrails* [31], *Triana* [181], *VIEW* [109], *DiscoveryNet* [152], *Chiron* [138], *GridNexus* [24]. We present discussion on the major functionalities and focuses of these SWfMSs in Chapter 5.

3 GRANULAR ATTRIBUTE LEVEL LOCKING FOR CONSISTENCY MANAGEMENT IN COLLABORATIVE SCIENTIFIC WORKFLOW COMPOSITION

Scientific Workflow Management Systems are being widely used in recent years for data-intensive analysis tasks or domain-specific discoveries. It often becomes challenging for an individual to effectively analyze the large-scale scientific data of relatively higher complexity and dimensions and requires a collaboration of multiple members of different disciplines. Hence, researchers have focused on designing collaborative workflow management systems. However, consistency management in the face of conflicting concurrent operations of the collaborators is a major challenge in such systems. In this chapter, we propose a locking scheme (e.g., collaborator gets write access to non-conflicting components of the workflow at a given time) to facilitate consistency management in collaborative scientific workflow management systems. The proposed method allows locking workflow components at a granular level in addition to supporting locks on a targeted part of the collaborative workflow. We conducted several experiments to analyze the performance of the proposed method in comparison to related existing methods. Our studies show that the proposed method can reduce the average waiting time of a collaborator by up to 36% while increasing the average workflow update rate by up to 15% in comparison to existing descendent modular level locking techniques for collaborative scientific workflow management systems.

In Section 3.1 of this chapter, we first discuss the motivation and importance of consistency management in terms of collaborative SWfMSs. Section 3.2 outlines technical preliminaries and challenges for consistency management in a collaborative workflow development environment. We present our empirical study on modern scientific workflows from the perspective of collaboration and existing methods in Section 3.3. We then present our proposed method in Section 3.4. Section 3.5 contains several experimental evaluations of the proposed method. We discuss possible threats to the validity of our work in Section 3.6. Section 3.7 presents the related research works. Finally, Section 3.8 reports our future works and draws the conclusion.

3.1 Motivation

Although a number of SWfMSs have been developed and proposed over the last decade (i.e., Taverna [140], Galaxy [66], Kepler [115], Pegasus [47], VisTrails [31], Triana [181], VIEW [109] and so on), none of them directly support collaborative works among multiple users; hence for any collaboration, users need to follow

several time consuming manual steps [199, 201]. For example, for a collaborative design of a scientific workflow, a user first builds a part of a workflow (e.g., a sub-workflow), exports it from the local workflow engine and shares it with a collaborator for possible updates on the sub-workflow. Around 3910 such scientific workflows has been shared among 10665 members (as last noted in June 2018) for collaboration in *myExperiment*[44] - a shared social space for scientific artifacts. The manual collaboration process is repeated a number of times to complete building the whole workflow comprising of several sub-workflows [201, 199]. This manual back and forth process for collaboration is often very time consuming, does not support real-time editing and often impractical as the collaborating group size increases in size over time [199, 164, 165, 201, 56].

Realizing the necessity of collaboration in workflow composition, several methods have been proposed and developed in recent years [199, 165, 164, 201, 167, 166, 56]. One of the most important challenges for any real-time collaborative system is the consistency management of the shared objects in the face of conflicting operations by the collaborators [173, 174]. Because, in a collaborative editing system, the concurrent operations on the same shared object might create several conflicting states at any given time frame. Generally, different version controlling techniques (e.g. SVN - Subversion, CVS - Concurrent Versions System and so on) are widely used for conflict resolution of the unstructured document collaborative systems, such as, collaborative Text Document Editing [144], collaborative Computer Aided Design (CAD) [37], object-based collaborative Graphics Editing systems [173], collaborative bitmap editing system [60] and so on. Unlike these documents, the scientific workflows are more structured where one module can be highly dependent on another due to dataflow relation in between them.

Modern scientific workflows are usually complex and data-intensive in nature such as, in the research fields of Astronomy [47], High Energy Physics [115], Bioinformatics [66, 141] and so on, scientists need to analyze terabytes of data requiring a significant amount of execution time [12]. As the workflow modular tasks and their corresponding configurations vary significantly from domain to domain, defining a unified rule for conflict resolution can often be very challenging in case of collaborative SWfMSs [164, 165, 56]. The existing research works for collaborative SWfMS has hence adapted several *locking* techniques of the workflow modules and datalinks to facilitate the consistency management. The locking techniques allow only a single user to get exclusive *Write* access to a component of the collaborative object to facilitate consistency management via preventing conflicting concurrent operations by the collaborators [173, 174]. Some of the recent proposed locking schemes for collaborative SWfMSs are: entire workflow object locking in turns [199], descendant modules locking [56, 201], multiple variants of module locking [165, 164] and so on (as discussed in details in Section 3.7). While those existing modular level locking techniques facilitate consistency, their usability reduces significantly as the number of collaborators or the complexity of the workflow grows over time (i.e., increased number of modules with complex datalink relation among them) [199, 201]. We have presented an empirical study on the limitations based on several recent real-world workflows from *myExperiment* in Section 3.3.

In an attempt to address those limitations of the existing methods, in this chapter of the thesis we propose

a novel approach for consistency management in collaborative SWfMSs. Our method works by extending the modular locks to a more granular attribute level. As the proposed method imposes locks on granular attributes level, it can minimize the significant amount of redundant locks in comparison to existing methods that generally operate on modular levels. We developed a prototype system as a proof of concept of the proposed method. We got promising results from our several simulated experimental studies. The experimental results show that the proposed locking scheme can reduce the average waiting time of a collaborator by up to 36%.

Our work makes three main contributions to the research domain. First, we present several insights from our investigation on modern scientific workflow systems from the perspective of collaborative workflow development system (Section 3.3). Our study includes recent real-world scientific workflows from collaborative shared space (e.g., *myExperiment* [44]). Second, we propose attribute level locking scheme based on our investigation study, which is the first of its kind to the best of our knowledge that works on a finer level beyond module locking for collaborative workflow development environment (Section 3.4). Third, we introduce two different aspects in the simulated experimental study of collaborative workflow development systems which have not been considered by any of the previous related studies: i) impact on performance for varying complex workflow tree structures (in addition to simple *VLinear*, *HLinear* and *HBinary* workflow trees used in an experimental study by Fei *et al.* [56]) and ii) performance analysis for varying topology of node access requests (Section 3.5).

3.2 Background: Consistency Management in Collaborative SWfMSs

With the increase of heterogeneous data and the complexity of problems, researchers have focused on *Computer Supported Cooperative Work (CSCW)* fields to exploit the advantages of a collaborative team works [137]. Real-time collaborative systems allow geographically distributed users to work concurrently and synchronously [173] on a shared text, image, graphics or multimedia document such as, collaborative Computer Aided Design (CAD) [37, 38], collaborative Text Document Editing [144, 20, 174], collaborative Graphics Editing systems [173], collaborative SWfMSs [199, 201, 56] and so on. Sun and Chen [173] mentioned the following two primary important requirements for any collaborative system to be successful:

1. *High Responsiveness*: This property demands that a collaborative system should be light-weight and user’s action must be quick— provided the condition of non-deterministic communication latency. Non-responsiveness and a significant delay between an action intention and its corresponding effect creates confusion and reduces the overall effectiveness of a collaborative system.
2. *High Concurrency*: The system should be able to handle concurrent edit operations on the shared object from multiple collaborators. As the number of users increases in a collaborative system, the probability of user’s action conflict on a shared object also increases. The collaborative system must

be able to handle the stream of user actions while maintaining the consistency of the shared objects among the users.

While existing research works adopted simple *replicated architecture* for maintaining *High Responsiveness* (i.e., keeping a local copy of the system to every client ends and using light-weight message passing for required collaboration information), ensuring *High Concurrency* is comparatively more challenging in the face of current conflicting operations.

If two concurrent workflow operations O_1 and O_2 (for example, changing module settings, updating datalink and so on) by the two users targets completely independent workflow components, then any execution order of the operations can ensure the consistency of the workflow objects. However, two concurrent operations $O_1 = changeValue(c, X)$ and $O_2 = changeValue(c, Y)$ raise conflict in case they target the same workflow component c but with different values i.e., where $X \neq Y$. For example, Fig. 3.1 demonstrates a use-case for workflow collaboration between two geographically separated collaborators. The targeted workflow for collaboration has been replicated to both the local workflow engines for ensuring *High Responsiveness*. We assume a consistent existence of the shared workflow object as version 0 (i.e., illustrated as 1.0 and 2.0 to represent the corresponding users) at any given time. At this state, we assume two concurrent operations - $O_1 = uDLink(m_m, \mathbf{m}_i)$ and $O_2 = uDLink(m_m, \mathbf{m}_k)$ are executed independently at user ends 1 and 2 respectively. However, the two operations raise conflict (i.e., on message passing its information to the other end), as they target the same workflow module m_m but with different values - m_k and m_i , (i.e., where $m_k \neq m_i$).

To facilitate the consistency management, the related works in workflow collaboration adopted several locking schemes, where collaborators are given exclusive *Write* access to different workflow components in turns to prevent concurrent conflicting operations. For example, Fig. 3.2 illustrates a locking schemes that imposes lock on the entire workflow object for a collaborator [199]. A central server maintains the lock requests and grants *Write* in turns. As at best only a single collaborator gets the *Write* at any given time, the above discussed conflicting cases can be avoided. Other recent locking schemes for workflow collaboration includes: descendant modules locking [56, 201], locking for dangling datalink prevention [165], multiple variants of module locking [165, 164] and so on (details on the related works in Section 3.7).

While the existing methods can facilitate consistency management, their usage for collaboration can often be impractical as the collaborating group sizes or workflow complexity grows over time. We have presented the findings on the limitation of the existing locking schemes from our empirical study on several modern real-world workflows of varying workflow management systems in Section 3.3.

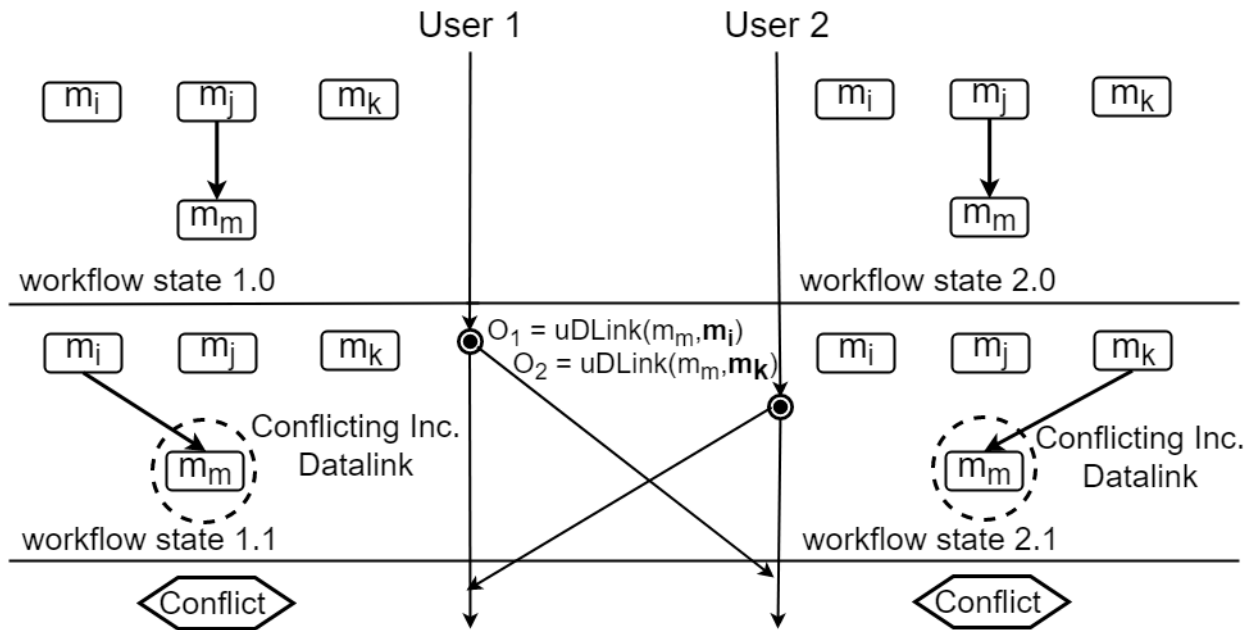


Figure 3.1: Workflow Version Conflict

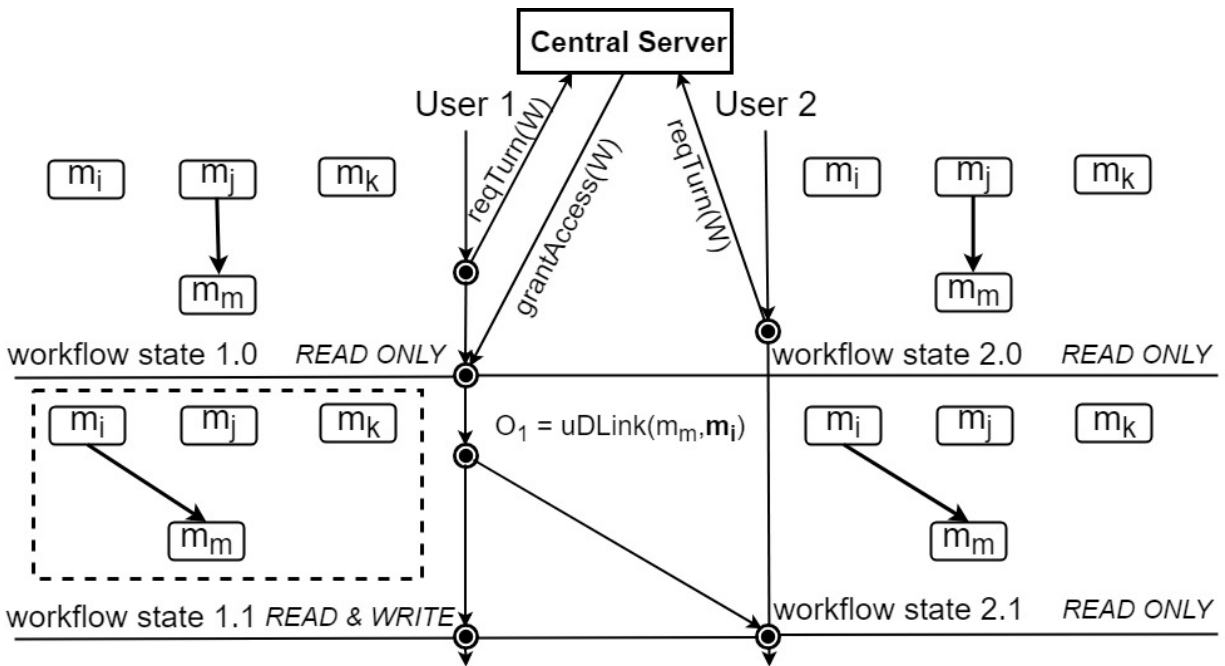


Figure 3.2: Collaborative Workflow Consistency Management via Turn Based Floor Control Technique

3.3 Empirical Study on Modern Scientific Workflow Collaboration: From the Perspective of Existing Locking Schemes

The existing locking schemes [199, 201, 56, 165, 164] might reduce collaboration concurrency in comparison to the proposed method significantly as the scientific workflow gets complicated with several dependency relations over time [56]. In this context, Fei *et al.* [56] defined the dependency degree of workflow modules in order to evaluate the concurrency level for collaborative workflow composition. The dependency degree of a workflow module m_d , denoted as $\phi(m_d)$, is defined as the cardinality of the set: $\{m_x | m_x \in M \wedge \mathcal{D}(m_d, m_x)\}$, e.g., the total number of distinct descendant modules. In this section, we present a comparative analysis of the proposed method in terms of existing methods from our empirical study on modern scientific workflow collaboration.

3.3.1 Study on Modern Workflow Dependency Degree, ϕ and its Impacts on Collaboration

The modern data-intensive scientific workflows that require collaboration the most, often contain a significant number of modules with a relatively higher number of dependency relation among them [56, 201, 199, 165]. For example, Table 3.1 shows the information of some publicly shared arbitrary workflows in *myExperiment*[44]. The number of modules comprising the workflows for different workflow engines indicates their growth in size and complexity over time. In the context of collaboration, allowing only strict sub-workflow locks by existing methods [56, 201] on a module with higher dependency degree ϕ , might result on a higher probability of reduction in the collaboration concurrency which is not expected for any collaborative system [173]. The average number of dependency degree (e.g., Avg. ϕ , in Table 3.1) for different workflows illustrates the average pruning of modules per hierarchical locking on a module from unlocked sub-workflow graph w^U (e.g., as noticeable from average lock % in the table). This average strict locks can be significantly reduced by the proposed locking scheme, as it operates on a more granular level avoiding possible redundant sub-workflow lock conflicts. Usually, the workflow graph modules contain an arbitrary number of dependency degree based on the particular task it solves. For example, Table 3.1, shows the top three modules in terms of dependency degree (e.g., Avg. ϕ) for the corresponding workflow from *myExperiment* [44]. It is noticeable from the numbers that the strict locking on such modules, in turn, locks the majority of the workflow modules (e.g., Lock %), resulting in a significantly lower collaboration concurrency.

3.3.2 Study on Modern Workflow Module Structure

Modern scientific workflow modules also contain a large set of compulsory and optional attributes or parameter settings (i.e., $c_i \in C_l, (1 \leq i \leq \mathcal{P}_{C_l})$ of a module m_l) for configuring as per the requirement of varieties

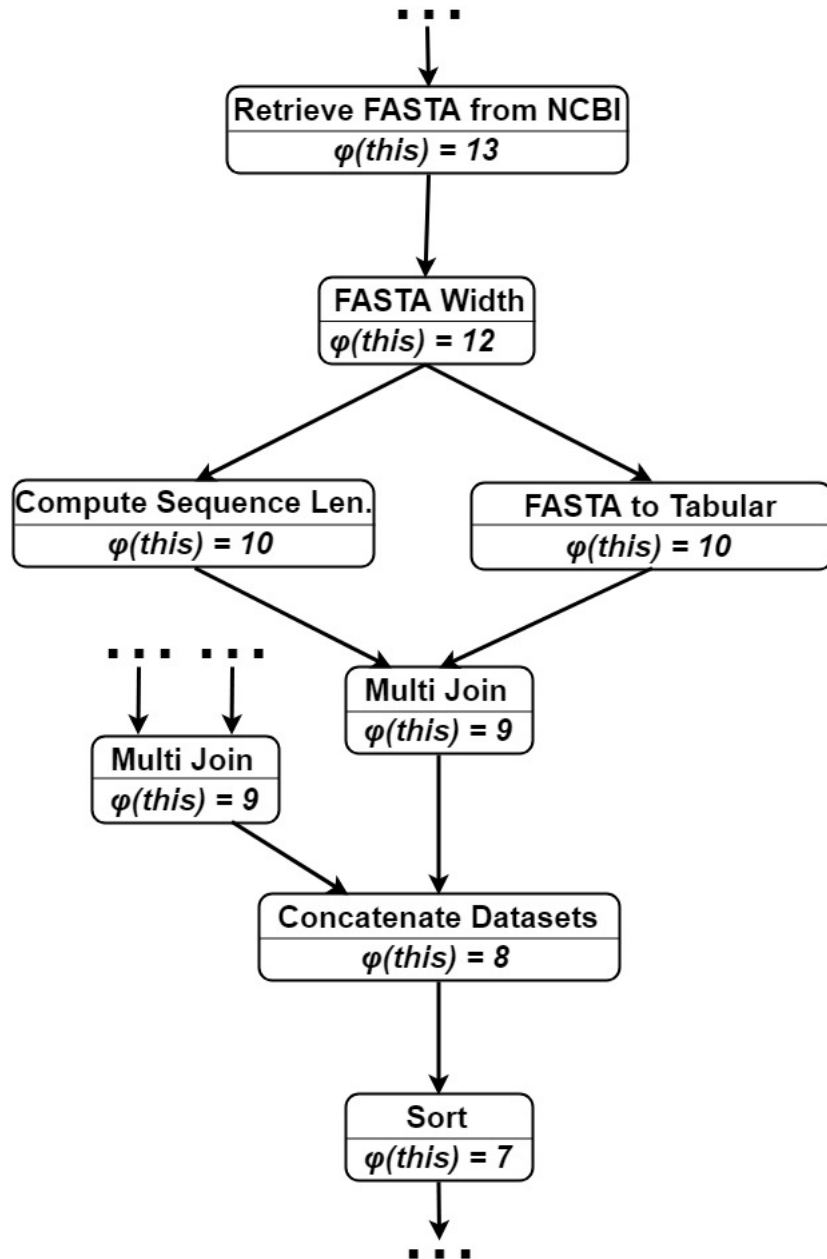


Figure 3.3: Part of a shared workflow in *myExperiment* [44] for collaboration (workflow id=4921, as noted in June, 2018)

Table 3.1: Information of some arbitrary scientific workflows from *myExperiment* [44] ¹

W. ID	Workflow Summary	W. Engine	#Mod	Avg. ϕ	Lock%	#Descendants	
						#1	#2
1198	Construction of skeleton SBML model using subsystem term	Taverna 2	133	16.8	12.6%	104	103
3599	Detrprok Workflow - detects candidates of 3 kinds of non coding RNA: 5'UTRs, antisense RNAs, and small RNAs.	Galaxy	43	8.6	20%	24	23
10	Human Microarray CEL file to candidate pathways	Taverna 1	80	17.8	22.1%	75	73

¹ As per the data collected in June, 2018

of heterogeneous data-intensive tasks [111]. For example, Table 3.2 shows the attribute counts for some arbitrary modules of Galaxy [66] workflow engine. The higher number of attribute counts per module in the table indicates a higher amount of time requirement for configuring it accordingly. The concurrency scope gets even reduced if that particular module contains a higher dependency degree, ϕ [56, 201, 199]. Because, in that case the hierarchical strict module lock in turns will be applied to its descendant modules for a longer period of time as well [165]. So, configuring any such module with higher attribute counts, sets lock to it and all its descendants for relatively longer amount of time. The concurrency scope gets even reduced if that particular module contains a higher dependency degree and causes strict locks in turn to all its descendants for a longer amount of time. In such cases, managing or waiting for the corresponding request lock grants and concurrent work on the workflow by the collaborator can be often difficult and impractical.

Table 3.2: Some of the Galaxy Tools with their Attributes Count

Tool	Tool Group	Version	# Attributes
HISAT2	NGS: RNA Analysis	2.0.5.2	19
FastqToSam	NGS: Picard	2.7.1.0	18
ValidateSamFile	NGS: Picard	2.7.1.0	65
Prokka	NGS: Assembly	1.12.0	33
Chimera.uchime	NGS: Mothur	1.36.1.0	19

From our investigation on the modules of popular scientific workflow management systems, such as Kepler [115], Taverna [141], Galaxy [66] and so on, we found that out of a number of attributes per module, a few of the attributes value changes significantly affects the execution behavior of the descendant modules, like: incoming data link change, output format change, threshold value of some filtering modules and so on. Any changes to such attribute values must be done via strict locking of the descendant modules, in order to preserve valid workflow execution path by inter-compatible attribute settings among the workflow modules.

3.3.3 Impacts of Module Attributes on Collaboration

Some of the attribute settings of the modules are often for further tuning its own execution behaviour without strictly affecting the execution path or behaviour of other modules in that workflow [186, 66]. For example, *FastQC* [54] - a quality control tool for high-throughput sequence data, accepts a *Fastq*, *Fastq.gz*, *Sequence Alignment/Map (SAM)* or *Binary Alignment/Map (BAM)* file as input to do a quality check analysis. As output, it produces a quality summary text file which can be used by descendant modules. However, the module can be configured to produce additional outputs, such as *HTML* based permanent report or compressed file with different quality graphs and so on. These additional outputs are not directly used by descendant modules, but rather later used for manual analysis [54]. *ValidateSamFile* - is a Galaxy [66] modular tool, that reads a *SAM/BAM* dataset and report on its validity. Out of its 65 attribute configurations (as shown in Table 3.2), 57 of them are about setting optional validation type to ignore. Setting strict lock to all the descendant modules in such cases can often be a much strong restriction for a practical and successful concurrent collaboration of workflow.

For example, Figure 3.3 shows a sub-workflow structure of a publicly shared workflow in *myExperiment* [44] for Galaxy [66] Workflow Management System (e.g., as noted in June 2018, workflow id=*4921*, title=*Retrieve from NCBI and reduce redundancy in the viral database*). The complete workflow consists of total 19 modules. As a use-case of the collaborative development of the workflow, we assume a collaborator intends for an optional configuration [66, 186] update of the workflow module - “*FASTA to Tabular*”. With dependency degree, $\phi = 10$, a strict lock on the module, in turn, locks around half of the workflow modules (e.g., $11 * 100/19 = 57.89\%$) - which is often impractical in the context of even medium-sized group collaboration and can be mitigated significantly by our proposed locking scheme.

3.4 Proposed Method

3.4.1 Fine-Grained Workflow Component Locking for Workflow Collaboration

A valid scientific workflow is composed of n finite number of workflow tasks or modules, m_i , ($1 \leq i \leq n$) which are responsible for performing some specific data oriented tasks [62]. The used set of workflow modules varies depending on the analysis or manipulation tasks on a given dataset. Besides, SWfMSs allow the configuration

of the individual workflow module via the corresponding parameter or setting changes to tune the execution behaviour for different tasks. While the configured modules are responsible for the given data oriented tasks, the datalink relation among them defines their execution order in the designed scientific workflow [55].

Based on these dataflow dependency relation among the modules, the scientific workflows are often represented as Directed Acyclic Graph (DAG), $W = (M, E)$ [56, 62, 61, 183], where M is a set of n different workflow modules $m_i \in M, (1 \leq i \leq n)$ and E is a set of directed edges $e_{ij} = (m_i, m_j), (1 \leq i \leq n, 1 \leq j \leq n, i \neq j)$ representing dataflow link from module m_i to module m_j [62]. The dataflow links create an execution dependency relation among the workflow modules. A module m_j can have zero to multiple predecessor module m_i , where m_j is dependent on m_i and the module, m_j can start its execution if and only if, all of its such predecessor modules finish their executions. We define the modular dependency relation as follows:

Definition 3.4.1 (Module Dependency Relation). *For a workflow $W = (M, E)$, a workflow module, $m_t \in M$ is dependent on the workflow module, $m_s \in M$, if there exists a sequence of workflow modules $m_0 = m_s, m_1, m_2, \dots, m_k = m_t$, such that datalink $(m_{i-1}, m_i) \in E$, for all, $1 \leq i \leq k$. Here, the workflow module, m_t is a descendant of workflow module m_s , represented as relation $\mathcal{D}(m_s, m_t)$, and cannot begin its execution until all of its such ancestor modules finish their executions.*

The workflow module in turns can be generalized as a tuple, $m = \langle id, C, S \rangle$, where id is a unique identifier (which is used for monitoring task execution, modular task scheduling, provenance management and so on) of the module in a designed workflow, C is a set of \mathcal{P} different configurations or parameter settings, $c_i \in C, (1 \leq i \leq \mathcal{P})$ to tune the module execution and S is the executable modular source code.

Consistency Management via Sub-workflow Locks for Collaborative Workflow Composition

Definition 3.4.2 (Hierarchical Descendant Module Lock). *For a workflow $W = (M, E)$, a hierarchical descendant module lock, $mLOCK(m_l)$ on any module $m_l \in M$, grants Write access to m_l and any other module $m_x \in M$, where the relation $\mathcal{D}(m_l, m_x)$ holds. The lock recursively applies on any datalink $(m_{i-1}, m_i) \in E$, in the module sequence $m_0 = m_l, m_1, m_2, \dots, m_k = m_x, (1 \leq i \leq k)$ for the dependency relation $\mathcal{D}(m_l, m_x)$.*

A hierarchical descendant module lock on any module $m_l = \langle id_l, C_l, S_l \rangle$, $mLOCK(m_l)$, thus allows Write access to any parameter settings or configuration, $c_i \in C_l, (1 \leq i \leq \mathcal{P}_{C_l})$, where \mathcal{P}_{C_l} is the number of parameter settings or configuration available in workflow module, m_l .

Definition 3.4.2 for descendant module locking is applicable for any simpler linear workflows to any hierarchical scientific workflows where a workflow is generally composed of several branched (e.g., dataflow dependency) smaller sub-workflows recursively. Based on the definition, a central locking algorithm can be designed for managing the concurrent sub-workflow lock/unlock requests by different collaborators (as presented in Section 3.4.2). For a collaborative workflow W , the central locking algorithm keeps track of currently locked and unlocked sub-workflows, such that, $W = w^L + w^U$, where $w^L = w_1 + w_2 + w_3 + \dots + w_n$,

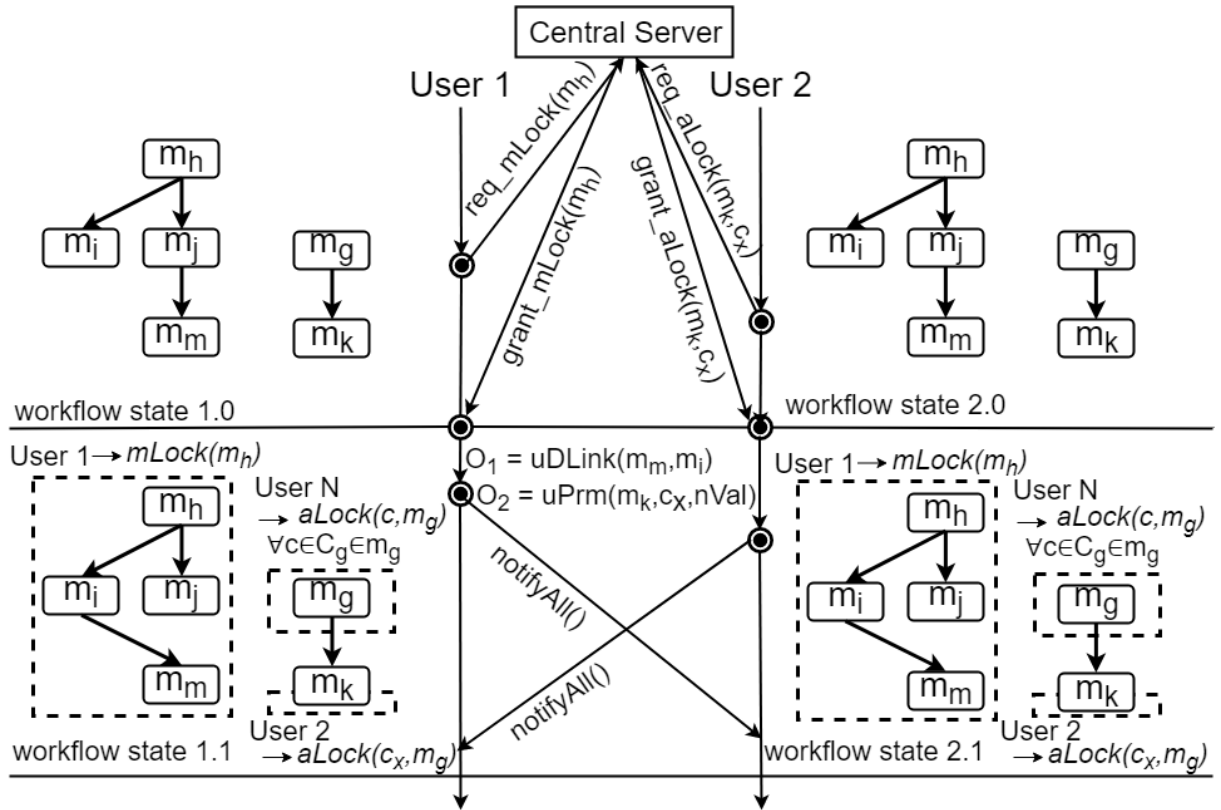


Figure 3.4: Collaborative Workflow Consistency Management via attribute level granular concurrency control Technique

a list comprising of sub-workflow w_i , ($1 \leq i \leq n$), that is currently locked by different collaborators and w^U represents the remaining unlocked sub-workflow that is currently not being accessed by any collaborators.

For a new lock request for sub-workflow w_r , with root module node m_r , the lock manager first checks for both locked and unlocked sub-workflow list, w^L and w^U respectively. The requested lock is granted if all of the descendant modules m_x , of requested root module m_r (e.g., $\mathcal{D}(m_r, m_x)$) belong to the unlocked sub-workflow w^U , otherwise, the request is pushed to a waiting list. w^U and w^L are updated accordingly for any lock/unlock state changes of the workflow.

Figure 3.4, shows a hierarchical descendant module lock, $mLOCK(m_h)$ by User 1 with root module m_h . Any update operation by only User 1 on this locked sub-workflow facilitate maintaining causal orders [173] on notifying other collaborating users. This hierarchical descendant module locks are useful when users intend to work on any sub-workflow explicitly.

Lock Extension to Granular Attribute Level

The hierarchical descendant module locking as presented above can alone facilitate the consistency management in the face of concurrent conflicting workflow operations. However, allowing only hierarchical descendant module locking can often be too restrictive in terms of modern collaborative workflow development (as dis-

cussed in Section 3.3). From these considerations, we further extend the lock to a more granular level as defined in the following:

Definition 3.4.3 (Granular Attribute Locking). *For a workflow module, $m_l = \langle id_l, C_l, S_l \rangle \in M$ of a workflow, $W = (M, E)$, an attribute lock, $aLOCK(m_l, c_i)$, grants Write access to attribute, $c_i \in C_l$ of the workflow module m_l .*

This granular locking allows additional controls in conjunction to hierarchical descendant module locking for explicit sub-workflow locking (e.g., Definition 3.4.2). Attaining an attribute level lock by a user ensures only single operation execution on the modular attribute at any given time to facilitate consistency management. This adds the scope for higher concurrency as it does not necessarily impose possible redundant locks to all its descendant modules (presented in details in our empirical study in Section 3.3). Besides, by Definition 3.4.3, the granular lock can be expanded to include any workflow module, $m_x \in w^U$ by iteratively imposing locks to all its attributes, $c_i \in C_x, (1 \leq i \leq \mathcal{P}_{C_x})$.

Figure 3.4, illustrates an example use case, where User 2 attains concurrent lock on any attribute, $c_x \in C_k \in m_k$, while another user (e.g., User N), attains module lock on the workflow module, m_g by expanding the granular attribute lock to its corresponding attribute set, C_g .

3.4.2 Lock Management Algorithms

Based on the above definitions and proposed locking scheme, we developed six algorithms (e.g., Algorithms 1-6) for managing the workflow component lock and unlock requests from collaborators. A central server keeps track of incoming lock requests for workflow components (i.e., sub-workflows, attributes or datalinks). The lock management algorithms grant an incoming lock request if it is compatible with the current locked workflow components, otherwise the lock request is pushed back to a waiting queue. On any workflow component unlock, the entire request waiting queue is traversed for any possible lock grants.

Algorithm 1 handles any incoming sub-workflow lock requests. The algorithm also applies for any single module locking, i.e., a sub-workflow such that, there is no other dependency relation with any other modules of the workflow. Lines 1-8, checks compatibility of the requested sub-workflow lock from the information present in w^U and w^L . For a sub-workflow lock request with root module, m_l the algorithm first check its own lock state for compatibility in lines 1-3 (i.e., lock state of the module and its corresponding attributes). Note that the straightforward extraneous details have not been shown (such as, initialization of the lists) to keep the pseudocode shorter while maintaining its preciseness. Similarly, the compatibility of the descendent workflow component for the root module m_l is checked in Lines 4-8. In case of the lock compatibility, the lock state of the sub-workflow root module and its corresponding attributes are first changed in lines 9-12. Finally, in lines 13-16, the lock state of the descendent workflow component is updated recursively. It also uses Algorithm 3 (i.e., in line 14) to lock the datalink from a source to destination relation.

On the contrary, Algorithm 2 is responsible for unlocking of a sub-workflow with root node module, m_l .

Algorithm 1: Lock Sub-Workflow

Result: Locks m_l and its descendants $\in W = (M, E)$

```
1 if  $m_l \in ListOfLockedModules$  or  $c_i \in C_l, (1 \leq i \leq \mathcal{P}_{C_l}) \in ListOfLockedAttr.$  then
2   | add  $m_l$  to RequestQueue
3   | return false
4 end
5 foreach  $m_x$  such that relation,  $\mathcal{D}(m_l, m_x)$  holds do
6   | if  $c_i \in C_x, (1 \leq i \leq \mathcal{P}_{C_x}) \in ListOfLockedAttr.$  then
7   |   | add  $m_l$  to RequestQueue
8   |   | return false
9   | end
10 end
11 Add  $m_l$  To ListOfLockedModules
12 foreach  $c_i \in C_l$  do
13   | call Lock Module Attribute,  $(m_l, c_i)$ 
14 end
15 foreach  $m_x$  such that datalink,  $(m_l, m_x) \in E$  do
16   | call Lock Data Link,  $(m_l, m_x)$ 
17   | call Lock Workflow Module,  $m_x$ 
18 end
19 return true
```

Algorithm 2: Unlock Sub-Workflow

Result: Unlocks m_l and its descendants $\in W = (M, E)$

```
1 if  $m_l \notin ListOfLockedModules$  then
2   | return false
3 end
4 foreach  $m_x$  such that relation,  $\mathcal{D}(m_x, m_l)$  holds do
5   | if  $m_x \in ListOfLockedModules$  then
6     | return false
7   | end
8 end
9 Remove  $m_l$  From  $ListOfLockedModules$ 
10 foreach  $c_i \in C_l$  do
11   | call Unlock Module Attribute,  $(m_l, c_i)$ 
12 end
13 foreach  $m_x$  such that datalink,  $(m_l, m_x) \in E$  do
14   | call Unlock Data Link,  $(m_l, m_x)$ 
15   | call Unlock Workflow Module,  $m_x$ 
16 end
17 return true
```

Algorithm 3: Lock Data Link

Result: Locks $(m_l, m_x) \in E$, of Workflow $W = (M, E)$

```
1 if  $(m_l, m_x) \in ListOfLockedDatalinks$  then
2   | add  $(m_l, m_x)$  to  $RequestQueue$ 
3   | return false
4 end
5 Add  $(m_l, m_x)$  To  $ListOfLockedDatalinks$ 
6  $(m_l, m_x) \leftarrow Read \ \& \ Write$  Access
7 return true
```

Algorithm 4: Lock Module Attribute

Result: Locks attr., $c_i \in C_l$, of workflow module, m_l

```
1 if  $(m_l, c_i) \in ListOfLockedAttr.$  then
2   |   add  $(m_l, c_i)$  to RequestQueue
3   |   return false
4 end
5 Add  $(m_l, c_i)$  To ListOfLockedAttr.
6  $(m_l, c_i) \leftarrow$  Read & Write Access
7 return true
```

Algorithm 5: Unlock Data Link

Result: Locks $(m_l, m_x) \in E$, of Workflow $W = (M, E)$

```
1 if  $(m_l, m_x) \notin ListOfLockedDatalinks$  then
2   |   return false
3 end
4 Remove  $(m_l, m_x)$  From ListOfLockedDatalinks
5  $(m_l, m_x) \leftarrow$  Read Access Only
6 return true
```

Algorithm 6: Unlock Module Attribute

Result: Locks attr., $c_i \in C_l$, of workflow module, m_l

```
1 if  $(m_l, c_i) \notin ListOfLockedAttr.$  then
2   |   return false
3 end
4 Remove  $(m_l, c_i)$  From ListOfLockedAttr.
5  $(m_l, c_i) \leftarrow$  Read Access
6 return true
```

Lines 1-2 first checks the unlock state of the root module. The unlock request is aborted in case the sub-workflow root itself is not in the locked state, i.e., $m_l \notin w^L$. A hierarchical sub-workflow lock ensures that all the descendent workflow components maintain a similar lock state as indicated in Definition 3.4.2. To maintain this property the states of parent modules of m_l are checked in lines 4-8. On a successful condition for the unlock of the sub-workflow, the requested root module (i.e., lines 9-12) and its corresponding descendent workflow components (i.e., lines 13-16) are recursively released from their locked state.

Algorithms 3 and 4 operate on the granular data link and attribute levels respectively. In addition to their invocation by Algorithm 1 for a given sub-workflow lock, the algorithms are also responsible for handling the lock imposition on granular workflow components (i.e., module attribute and datalink relation between a source and a destination module).

3.5 Experiments and Evaluations

3.5.1 Implementation Details

We implemented a prototype of the proposed method as a proof of concept. The prototype implementation is a cloud-based system hosted in a Linux Server. We used Python 2.7 as the server side language. On the other hand HTML5, CSS and JavaScript were used for client-side programming. We also used Ajax for asynchronous server communications.

Fig. 3.5 shows a screenshot of the collaborative workflow composition panel. We adapted the proposed locking scheme for consistency management while collaborative workflow composition. The module and attributes are color-coded to represent their corresponding lock states to the collaborators. For example, the green color-coded sub-workflow (i.e., comprising of Modules 3, 4 and 5) denotes the locked sub-workflow by this collaborator, the red color coded sub-workflow (i.e., comprising of Modules 1 and 2) shows the sub-workflow currently locked by other remote collaborators and the remaining white colored workflow components (i.e., Module 6 and 7) represent no collaborators currently hold locks on those corresponding components. Collaborators can request, release or see the current lock status of any corresponding workflow components. For example, the similar options has been invoked (i.e., with right-click on the mouse) in the sub-workflow with root node - Module 6. We also implemented other existing locking schemes [199, 201, 56] for our experimentation on comparative study. The details on the corresponding experiments are presented later in this section.

3.5.2 Experimental Setup

For our experiments, we considered six basic workflow operations as presented in Table 3.3. Majority of the complex workflow operations are composed of sets of such basic workflow operations. Workflow collaborators were simulated using independent threads. To simulate *short-read, long-thinking* pattern [201, 199], as

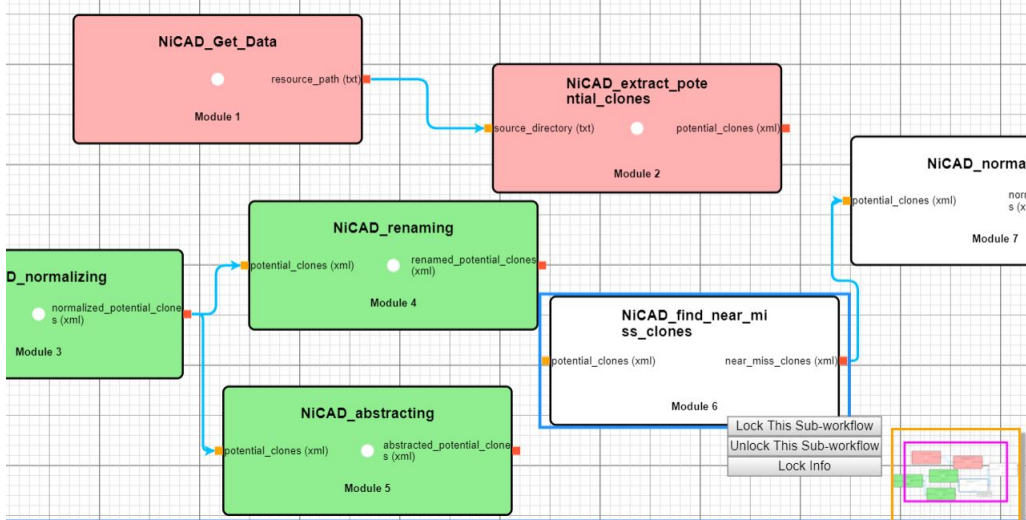


Figure 3.5: Sub-workflow Locks in Collaborative Workflow Composition

adopted by related works [201, 56], we considered random *thinking time* [201] interval ranging from 10 ms to 15 ms in between any basic workflow operation execution by a collaborator. If the next thinking time is relatively longer (e.g., considered, >10 ms), the corresponding collaborator releases any accessed object, making it available for other collaborators of the group. The relatively shorter interval time results in the possibility of generating relatively more conflicting operations, and hence has been used for testing the performances of the algorithms in extreme conditions [199, 201] (e.g., while the shorter interval time is good for the performance testing of the algorithms in simulated environment and adapted by related studies [199, 201], the human thinking time can often be relatively longer or non-deterministic in real-world setups. We also present our study on such real-world setups in Chapter 6). The considered access request or release controls have been presented in Table 3.4. To mitigate any possible biases from the results, the experiments were repeated three times, and their average values were used for convergence.

Table 3.3: Considered Primitive Workflow Operations.

Index	Workflow Operation
1	Adding a New Module to the Workflow
2	Adding a New Datalink Relation From a Module
3	Adding a New Datalink Relation To a Module
4	Updating a Configuration Attribute of a Module
5	Updating the Source of an Existing Datalink Relation
6	Updating the Destination of an Existing Datalink Relation

Table 3.4: Primitive Operations For Component Access in Collaborative Workflow Composition Environment.

Operation Type	Collaborative Operation
Component Access/Update	Sub-workflow Access Request
	Sub-workflow Access Release
	Module Attribute Access Request
	Module Attribute Access Release
	Module Attribute Update
DAG Layout & Views	DAG Node Location Update
	DAG Datalink Location Update

3.5.3 Study on Average Waiting Time and Throughput

Fig. 3.6 illustrates the experimental results on *average waiting time* and *throughput* for varying number of group sizes for collaboration.

The waiting time, δ_i^j of a user, U_j for a lock request $R_i^j \in \mathbb{R}^j$ (i.e., where \mathbb{R}^j is a set of all requests from the collaborating user U_j), in a collaborative workflow composition environment is calculated as the total amount of time delay between the access request and its corresponding access grant [201, 199]. So, for a group of n collaborators the average waiting time, α is calculated as, $\alpha = \frac{1}{n} \sum_{j=1}^n \sum_{i=1}^{|\mathbb{R}^j|} \delta_i^j$. A lower value of average waiting time denotes higher responsiveness of the system resulting in better overall productivity of the collaborating group. As noticeable from the graph, all the locking schemes show around a similar waiting time when the group size is relatively smaller (i.e., maximum of two collaborators). However, as the group size increases significant differences are noticeable among the locking schemes. The graph depicts that the turn-based locking scheme is comparatively more sensitive to the group size, as it steadily increases with the number of collaborators. The difference between strict module and proposed locking schemes are also noticeable as the group size increases beyond six. For example, the average waiting time for the proposed attribute level locking scheme with 18 collaborators is around 165 ms in comparison to 2495 ms and 433 ms for turn-based [199] and strict descendant module locking schemes [56, 201, 165] respectively.

While the average waiting time is somewhat correlated with the responsiveness, the throughput or workflow updates count per unit time hints the overall concurrency support of the collaborative system. So, we were also interested to investigate the throughput of the corresponding locking schemes (as illustrated in Fig. 3.6). Up to a group size of two, the turn based locking scheme shows a better average throughput in comparison to the other locking schemes. A possible reason for this behavior is the turnaround time between the request and its access grant on smaller components, in comparison to a fewer number of access requests in case of turn based locking scheme (i.e., access request on the whole workflow object). However, as the

group size increases the throughput for the turn based locking scheme decreases noticeably. The proposed locking scheme shows significant improvement in the throughput in comparison to other locking schemes with the increase of group size. For example, the proposed method shows consistent better performance when the group contains more than five collaborators. The workflow update count per minute for turn based, strict and attribute level locking schemes are around 4886, 12880 and 20143 respectively for a collaborative group size of 18. The graphs also show a similar trend for a higher number of collaborators, which is promising.

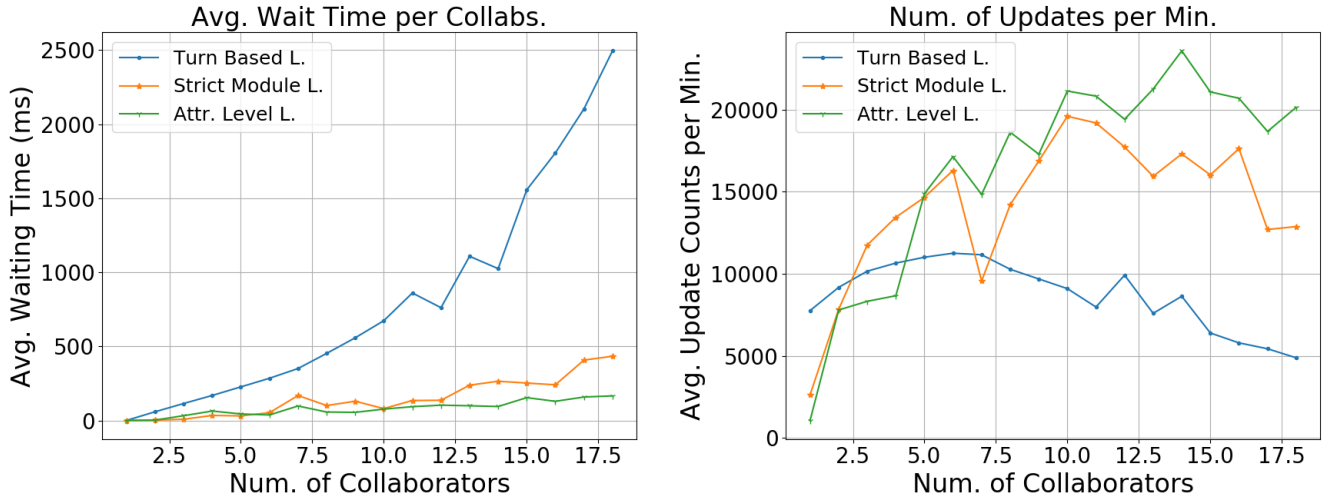


Figure 3.6: Average Waiting Time and Throughput Comparison of Locking Algorithms for Collaborative Workflow Composition.

3.5.4 Study on Workflow Composition Time and Efficiency

The workflow composition time is considered as the total required time to complete the execution of all the intended updates from the collaborators towards finalizing a workflow composition collaboratively. To evaluate the composition time of the locking schemes, every collaborator were assigned to execute a specific number of updates operations at random from Table 3.3. For our experiments, we assigned 25 such workflow update operations at random to each of the collaborator in a group. The total workflow composition time is then calculated as the total required time to finish the update execution from all the collaborators (i.e., for a group size of n , the total number of updates operation to execute are $25 * n$). Fig. 3.7 illustrates the obtained results from the experiments on the locking schemes. As noticeable from the graphs, the turn based locking scheme is very sensitive to the group size for the workflow composition time. As an inclusion of a collaborator to the group, adds extra waiting time to all the other collaborators in addition to an extra set of update operations, the effect is clearly noticeable from the graph of turn based locking scheme.

Zhang *et al.* [201] used efficiency value for a similar comparative study that denotes the ratio of task

occupancy in a given time frame and calculated as following Eq.:

$$E = \frac{\text{throughput} * \text{unitTime}}{\sum_{i=1}^{\text{numOfTasks}} \text{unitTime} * \text{numOfTasks}} \quad (3.1)$$

The figure also illustrates that the efficiency values (i.e., Eq. 3.1) for the existing locking schemes decreases significantly in comparison to the proposed method with the increase in collaborative group size (e.g., especially the behavior is noticeable in the graph when the group size is more than 3).

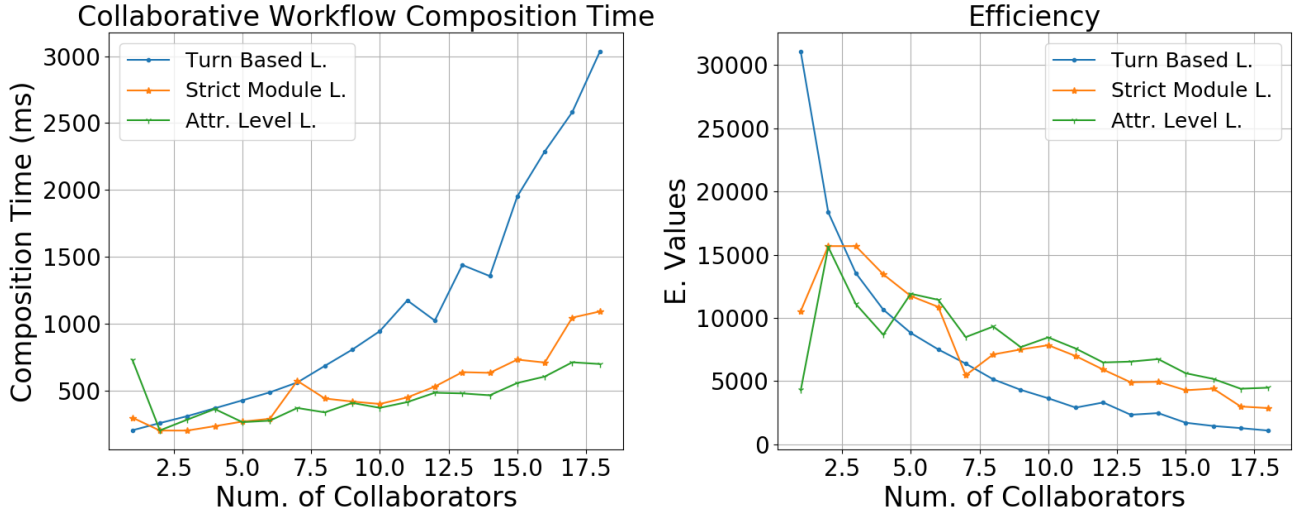


Figure 3.7: Comparative Study on Collaborative Workflow Composition Time and Efficiency of Locking Schemes.

3.5.5 Performance Study on Varying Node Access Requests Topology

The locking algorithms follow different techniques for serving the sub-workflows on the requests by the collaborators. For a given collaborative workflow, $W = (M, E)$ and a sequence of collaboration locking requests R^S , a locking algorithm, L partitions W into w^L and w^U , i.e. locked and unlocked sub-workflows respectively, i.e., $L(W, R^S) \rightarrow W = w_1, w_2, \dots, w_n \in w^L \cup w^U$ [164]. The topology of the granted requests can vary for different locking algorithms. In addition to the differences in locking approaches by different algorithms, the topology of the granted requests also depends on several other factors. For example, Sipos et al. [165, 164] mentioned three factors that determine the decision (i.e. granting/denying of a collaboration request) of a locking algorithm:

- (i) The current state of the workflow graph W , comprising of different modular tasks and corresponding datalink relation among them
- (ii) The topology of already locked n sub-workflows, $w_1, w_2, w_3 \dots w_n \in w^L$
- (iii) The topology of l lock requests, $R_1, R_2, R_3, \dots R_l \in R^S$

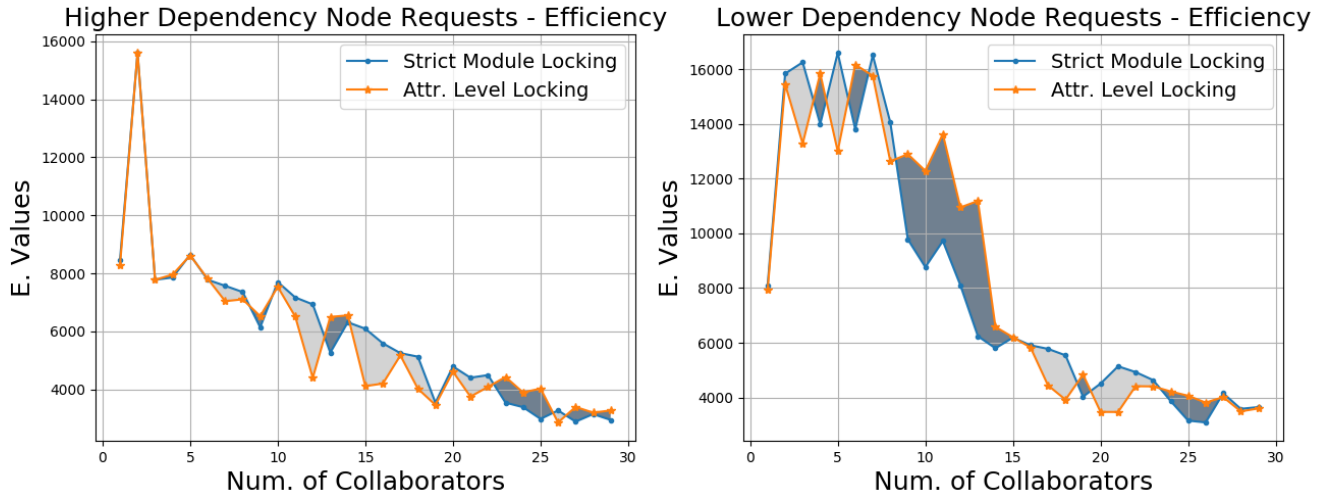


Figure 3.8: Algorithm Efficiency on ‘Best’ and ‘Worst’ Case Scenario of Collaborative Node Access Request Topology.

So, we were interested to investigate the algorithms performances on varying access request topologies. To test the performance in extreme cases, two types of request topologies were considered - a new access request always targets the available: i) node with lowest dependency degree, and ii) Oppositely, node with highest dependency degree, ϕ (e.g., as defined in Section 3.3). Figure 3.8 illustrates the obtained results by the algorithms.

As it is noticeable from the graph of *higher dependency node requests* topology, that they do not show any recognizable patterns in their differences. As the proposed method also allows explicit lock on any entire sub-workflow as per the requirement of a collaborator, in principle both the locking schemes follows somewhat similar patterns for their lock access grants in this case; which is a possible reason for such behavior by the algorithms. Similarly, although the proposed method shows a better result in case of *lower dependency node requests* topology, the difference is lesser prominent, unlike the comparisons of the algorithms in all other dimensions.

The above results suggest that the proposed locking scheme can adapt to both the extreme case scenarios as per the requests from collaborators. For example, for a workflow collaboration with relatively larger group size, collaborators might prefer working on different sub-workflows of varying size independently [201]. The proposed locking scheme can also adapt to such cases in addition to the finer component level locking of the workflow object.

3.5.6 Analysis Study in terms of Varying Workflow Tree Structures

In Section 3.4, we suggested a hypothesis that the lock on a module with a higher dependency degree ϕ , can largely impact the overall collaboration scope of the workflow. We thus conducted several experimental studies with varying workflow tree structures to test the hypothesis. We considered six different dependency

relations of the workflow trees as presented in Figure 3.9.

The 2, 3 or 4 *regular workflow trees* in the figure represent three different structures, where every non-leaf workflow module has exactly 2, 3 or 4 child module respectively. The 2, 3 or 4 *all connected workflow trees* on the other hand, represent some-what similar structures, but with higher dependency degree considerations where, any workflow module of level l , is dependent on all of the modules of level $(l - 1)$ by direct incoming dataflow relation among them (i.e., except for the root workflow module). The experimental results as illustrated in the figure, show a significant increase in average waiting time with the increase of overall dependency relation in case of strict module locking scheme.

For example, in case of strict locking, for 2 all connected workflow tree the average waiting time raises up to around 1520 ms in comparison to proposed attribute level locking scheme, which is approximately 958 ms. That is, the average ratio of reducing the overall waiting time by the proposed method is around 0.63 (i.e., $958 * 100 / 1520 = 63\%$) in case of 2 all connected workflow tree structures. Similarly, in case of 2 regular workflow tree structures, the average waiting time with 29 collaborators raises up to around 1243 ms and 726 ms for strict module locking and proposed locking schemes respectively (i.e., approximately 58% reduction by the proposed locking scheme). It is also noticeable from the graphs that, the difference in average waiting time between the locking schemes increases significantly with the increase of collaborating group size. For example, both the locking schemes show more or less similar average waiting time when the group contains five or less number of collaborators, however, the average waiting time increases noticeably with the increase of group size for strict locking scheme in comparison to the proposed locking scheme. These results suggest higher concurrency of the proposed locking scheme.

3.6 Threats to the Validity

In our simulated experimental studies, we adopted *short-read, long-thinking* pattern [201, 199] with a pre-defined thinking time range to imitate the human collaborators' working behavior. However, the human working pattern can be more diverse in nature (e.g., longer *thinking time*, inter-collaborator communications and so on) and thus it can be often challenging to exactly imitate in a simulation study. While this is a common threat for any simulation studies, the existing state of the art simulation based related techniques [199, 201, 56] used this approach for evaluating their studies with success which gave us confidence on our evaluation as well. Furthermore, in order to mitigate any biases in the results, the exact experimental settings were applied to all of the locking schemes, the experiments were conducted on the same machine and also repeated a number of times to use their average values for convergence. We also conducted the experiments in several dimensions (as presented in Section 3.5) to validate the performance comparison studies.

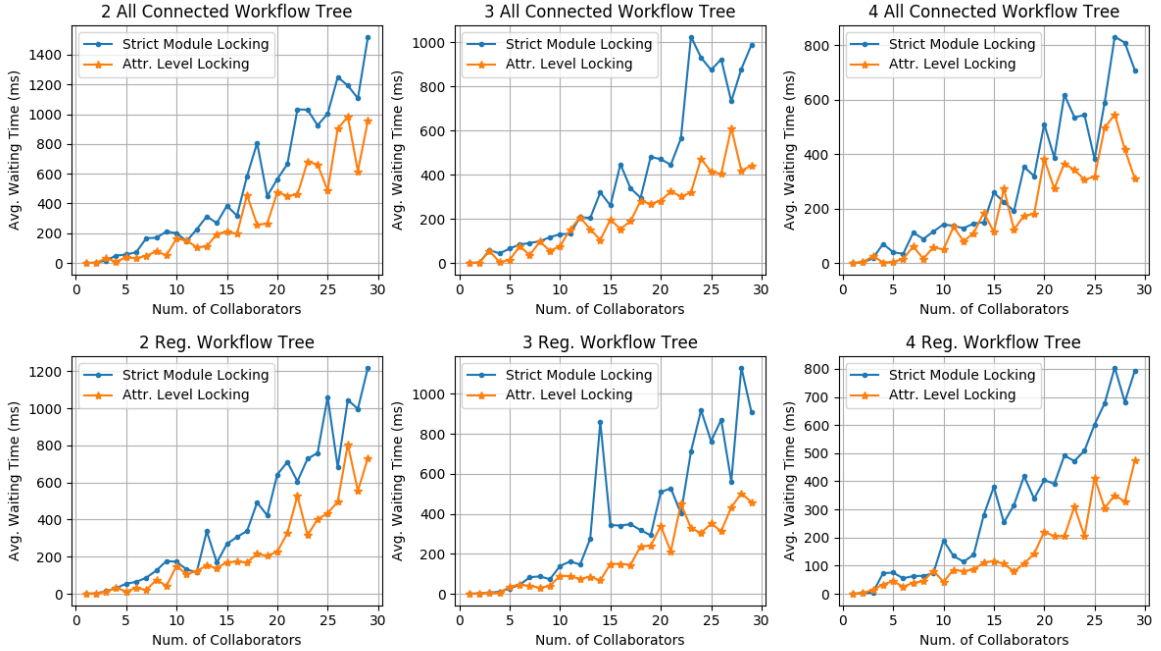


Figure 3.9: Performance Analysis in Terms of Varying Workflow Tree Structures

3.7 Related Works

Scientific workflow management systems have gained much popularity for data intensive analysis and has been adopted by different research domains [47, 159, 115, 66, 141]. Several workflow management systems have been proposed and developed over the last decade focusing on specific research branches. As the scientific data complexity, dimension and volume increase significantly in recent time, researchers of different domains try to exploit the Computer Supported Cooperative Work (CSCW) to accelerate the analysis process efficiently. These real-time collaborative techniques have also been extended to scientific workflow management systems in recent years.

SWfMSs have gained much popularity in the past few years and are widely used for data-intensive analysis, simulation, visualization and so on [12, 111]. Some of the popular modern SWfMSs are: *Taverna* [141], *Galaxy* [66], *Kepler* [115], *Pegasus* [47], *VisTrails* [31], *Triana* [181], *VIEW* [109], *DiscoveryNet* [152], *GridNexus* [24] and so on. However, none of the existing SWfMSs support collaborative workflow composition.

Lu *et al.* [114] studied several motivations and opportunities for collaborative SWfMSs from the perspective of large-scale and multidisciplinary research projects. In recent years, several methods have been proposed for consistency management of the shared workflow in a collaborative environment. Floor control or turn based locking schemes (e.g., the entire collaborative object is locked in turns by collaborators) are widely used for consistency management in a collaborative work environment. Zhang *et al.* [199] studied the concept in

the context of collaborative workflow management systems. While such turn based approach better matches with human communication protocol (e.g., Robert's Rules of Order (RRO) [118]), it has several issues such as only a single collaborator can work on workflow update at any given time (e.g., the concurrency count is significantly low), longer average waiting time even for a medium-sized collaborative group and so on.

Each collaborator generally has only the *Read* access to the shared workflow. Collaborators request and compete for the floor for carrying out any update or transaction on the workflow (e.g., *Read & Write* access). The collaborative workflow management system server maintains a request queue for handling the floor control requests. If the floor is not currently occupied, the system pops out and grants the floor access to the appropriate collaborator (e.g. following some request dispatching protocols: first come first serve or collaborator priority based on requester roles and so on) from the request queue. Any workflow update operation information by the floor owner is sent to all other collaborators by simple message passing techniques. As only a single collaborator works on workflow update at any given time, the consistency maintenance for all the collaborators is much simplified in this case. However, the concurrency count is significantly low in this method. So, the average waiting time for getting floor access can often be much higher, even in case of a medium-sized team.

Fei *et al.* [56] and Zhang *et al.* [201] presented locking schemes by allowing only descendent module locks (e.g., descendent nodes of the workflow DAG [111]) instead of imposing the lock on the entire workflow. Though the collaboration concurrency is increased in this case in comparison to turn based locking [199], these modular locking schemes show a significant reduction in the concurrency count as the workflow complexity grows over time with an increased number of modules and complicated datalink dependency relation among them (Section 3.3). Because a modular lock in these cases, in turn, locks major portions of the collaborative workflow. In an attempt to lower the redundant sub-workflow locks (e.g., any intended update on a module, strictly locks all of its descendants modules due to the extension of the locking set [164]), using multiple modes of sub-workflow locks have also been proposed. Sipos *et al.* [164] used two lock modes - *User* and *System* locks. *User* locks are applied to only the module where a collaborator intends for any update, while *System* locks are applied recursively to all its descendants. As two '*System Locks*' are considered compatible in this proposed method, it can provide slightly better concurrency than strict descendant modules locking at some conditions.

Fei *et al.* [56] proposed a lock compatibility matrix for a set of six pre-defined modes of locks. While multiple modular locks can avoid a few of the redundant locks depending on the defined compatibility relation, the improvement is almost negligible for a larger collaborative group due to their several lock conflicts. Techniques have also been studied for extending the single-user Grid portals to a collaborative environment [168, 165]. Dou *et al.* [49] studied context and role-driven scientific workflow development pattern in a collaborative environment. However, the extension or generalization of the method is challenging as defining non-conflicting roles can often be much complex in terms of consistency and depends largely on the given collaboration domain. The contention and releasing phases are much similar to request and release of locks

respectively of turn based collaboration. The workflow update operations after the contention phases are synchronized for all the collaborators in the editing phase. The corresponding sub-workflow phases are managed by maintaining a copy of the collaborative workflow to a server as a global workflow [168].

As a valid scientific workflow is usually a DAG [111, 12, 164], several methods have been proposed for maintaining different DAG properties in a collaborative environment. Kavitha *et al.* [101] proposed a method for identifying loops in a complex workflow involving multiple organizations or departments. The proposed method uses Petri Nets for the cycle or loop detection in the workflow graphs. Sipos *et al.* [167] proposed a locking scheme for avoiding cycles or invalid edges in the workflow graph in any concurrent workflow update operations.

However, the existing locking schemes operate in the modular level and thus often result in significantly low concurrency count in modern scientific workflow collaborations (Section 3.4). To mitigate the similar problems, collaborative research works on other domains such as text or graphics editing systems have considered locking on finest component levels. For example, Sun *et al.* [172] studied fine-grain locking scheme in the character sequence levels for collaborative text editing systems as previous studies [131] show that finer grained locking allows higher concurrency in collaborative environments. To the best of our knowledge, our work is the first in the context of collaborative SWfMSs to consider finer attribute level locking in comparison to workflow module level locking.

3.8 Conclusion

In this chapter, we presented our investigation results of existing locking schemes in terms of consistency management of modern scientific workflow collaboration in the face of concurrent conflicting operations. From our study, we found that considering module level workflow locks can often be a strong assumption resulting significantly low concurrency. We proposed a fine-grained locking scheme by further extending the modular locks to attribute level. The proposed attribute level locking scheme attempts to accelerate the collaborative workflow development process by lessening redundant sub-workflow locks. We got promising results from our simulation studies on multiple collaboration scenarios with a reduction of average waiting time by up to 36% while an increase of average workflow update rate by up to 15% in comparison to existing descendent modular level locking techniques.

While the locking scheme ensures the consistent workflow composition in the face of conflicting concurrent update operations, a collaborative SWfMS also requires access control technique to efficiently manage the access of the workflow components among different collaborators [17, 114]. Hence, we propose a role based access control technique in addition to the locking scheme towards efficient design of a collaborative SWfMS. We present our proposed role based access control technique in the next chapter.

4 MODELING A COLLABORATIVE SCIENTIFIC WORKFLOW MANAGEMENT SYSTEM USING CIAM: A CASE-STUDY WITH PLANT PHENOTYPING AND GENOTYPING

While the consistency management is one of the primary requirements of a collaborative system [173] (e.g., as discussed in Chapter 3), collaborative SWfMSs involving multiple disciplines often need to consider access control technique to manage or orchestrate the entire process of collaboration [72, 124, 202]. In other word, collaborative SWfMSs need to provide some ways of managing the access controls of different workflow components among collaborators, while still allowing the collaboration on the shared scientific workflow and its components [17, 114]. We adapt *Collaborative Interactive Application Methodology (CIAM)*[122] for efficient design of access control techniques in terms of in collaborative SWfMSs. We show an use-case scenario using *Plant Phenotyping and Genotyping* research domain as an evaluation of the proposed access control technique.

In this chapter, we first discuss the motivation and importance of access control techniques in terms of collaborative SWfMSs in Section 4.1. We then present related existing works on this research domain in Section 4.2. In Section 4.3, we provide an example scenario of collaboration in terms of data analysis and then we discuss our proposed method of role-based access controlling of the workflow components in Section 4.4. Section 4.5 presents the evaluation of the proposed method. We finally draw conclusion of the chapter in Section 4.6.

4.1 Motivation

Several recent studies demonstrate the necessity of collaborative systems towards conducting complex scientific experiments involving multiple researchers of varying domains [41, 91, 68, 201]. The studies envisioning the collaborative SWfMSs similarly, have gained significant focus among researchers over the past few years [114, 200, 78, 201, 199, 165, 164]. However, the design of such collaborative or groupware systems comprising multiple users of varying roles is often progressively extended in comparison to single-user oriented systems [122, 196]. The design of such collaborative systems often raises added issues such as modeling cooperative procedures, roles of multidisciplinary users and spaces for sharing information [122]. We adapt *Collaborative Interactive Application Methodology (CIAM)* [124] towards addressing and analysis of these requirements set from collaborative SWfMSs perspective. CIAM leverages *Collaborative Interactive Applications Notation*

(CIAN) for considering Software Engineering designing methodologies while taking into account the requirements from Computer Human Interaction perspective [124]. We present our studies of collaborative SWfMSs with a use-case of *Plant Phenotyping and Genotyping* research domain.

With the rapid increase of the world population every year, ensuring the required amount of food consumption rate worldwide has been a challenge in recent years. Tilman et al. [182] studied on predicting the future consumption demand. Their study shows that agricultural production must have to be doubled to fulfill the consumption demand of a rapidly increasing population by the year 2050. The situation even gets worse with unfavorable climate changes and the overall reduction of agricultural lands for the accommodation of the increased human population worldwide [182]. So this has been a huge challenge in the recent years and researchers are trying to come up with solutions that can accelerate agricultural production in comparatively smaller agricultural lands and that can better adapt with those unfavorable climate changes and other environmental impacts [57]. Thus, plant Genotyping and Phenotyping are important for meeting the future consumption demand. The research on plant Phenotyping and Genotyping involve researchers from multiple disciplines. For example, research on Plant Phenotyping relies increasingly on image processing to organize the observations of different behaviors of the plants and to quantify those observations in pursuit of better understandings of different factors that correlate with several plant diseases or hampers the healthy life cycle of plants. Plant Genotyping on the other hand largely involves research works on bioinformatics to extract important information from plant gene sequences that possibly correlates with Plant Phenotyping or other important factors that affect a healthy plant life cycle [80]. Besides, to get a successful result, the research requires constant monitoring of the plants and thus generates huge amounts of gene sequences or image data to analyze [57, 7]. So, the research area also involves Big Data, High Throughput, Distributed computing [52]. A collaborative research involving all those multidisciplinary researchers to develop and improve an automatic system for a given scenario often becomes a challenging work. For example, image processing researchers often lack the time and resources to engineer their code for robustness and compatibility, while plant scientists feel the need to try new developed technologies or algorithms but this whole process is slowed down for the lack of well designed and tested user interfaces to make practical use of them by plant scientist. Managing a common free time for all the researchers for arranging some physical meetings often become difficult. On the other hand, even if the communication is done via such physical meetings or some electronic medium it lacks sharing and testing of works across multidisciplinary researchers in addition to just communication or discussion about the work progress. Oppositely on the fly collaboration among researchers for sharing and testing of works in addition to just communication can help to monitor the overall work progress and more importantly finding any potential problems or opportunities available in any of the modules in earlier phases of the research and development.

Various frameworks (e.g., Galaxy [3], iPlant Collaborative [121], GenAp [105] and LemnaTec [1]) have been developed to automate the scientific workflows management and support the computational needs of this domain. One of the challenges of these frameworks is that associated stakeholders (e.g., agronomists, data

specialists, image analysts and tool developers) work in isolation to perform their tasks towards developing a workflow or pipeline. As a result, it is often difficult for stakeholders to perform their tasks effectively. For example, if an agronomist wants to compose and execute an image processing pipeline, they encounter various problems such as the difficulty of accessing an appropriate set of data, difficulty to execute a tool comprising of several configuration parameters or unavailability of tools that he wants to add in his pipeline after analyzing its output. On the other hand, a tool developer encounters difficulty to define appropriate input parameters and output of a tool. These kinds of problems of the stakeholders could be solved if they could communicate and collaborate with each other effectively while working towards building a workflow.

In order to address these shortcomings of the existing frameworks, we propose a cloud-based collaborative SWfMS where various stakeholders can compose pipelines on-the-fly by getting help from each other. For example, an agronomist should be able to send a message to a tool developer along with sharing the pipeline in order to integrate a tool (e.g., a data transformation tool) that they wish to add in the platform after analyzing their pipeline. Using the framework the tool developer should be able to add their tool on-the-fly without even recompiling the system. Moreover, the platform should allow users testing different algorithms and techniques via simple and intuitive user interfaces collaboratively. For example, plant scientists can customize different algorithms from the user interface by changing different parameters instead of dealing directly with source codes to analyze and give feedback about the result to the image researchers right on. The proposed method thus works as a communication layer among multi-disciplinary researchers and thus rapidly and efficiently handling the research growth in collaborative setups.

While we present the underlying architecture of our proposed collaborative SWfMS in Chapter 5, we first present our studies leveraging CIAM to identify different roles and to model the responsibilities and processes in the context of collaborative SWfMS. The platform thus allows specific role-based users integration, e.g., plant Phenotyping researchers will log in with a plant researcher role, whereas image analysis tool developer will log in with an image researcher role and so on with some functions of access controls on the workflow components. As a result, using the platform image researchers should be able to get real-time feedback from the plant researchers for their developed algorithms.

We evaluated our framework by getting feedback from three different stakeholders such as, bio-informatician, a tool developer and an agronomist—which were promising. Although we present a comprehensive evaluation of our proposed collaborative SWfMS in Chapter 5 and 6 for the detailed architecture and user-studies respectively, here we focus our presentation on the high-level collaboration methodologies using CIAM on the SWfMSs perspective.

4.2 Related Works

We propose a framework using which users can compose pipelines for plant Phenotyping and Genotyping. In addition, users should be able to test a developed tool for understanding its usage. Considering these two use

cases of our developed tool, we divide the related work into two areas: (i) frameworks or tools that support scientific pipeline composition, and (ii) frameworks or tools that support rapid API testing.

4.2.1 Frameworks or Tools Supporting Pipeline Compositions

Plant Genotyping and Phenotyping analyses involve numerous steps including physical plant sample collections, data curation, data conversion into different steps for generating users' expected end results, and making analysis results available to researchers and practitioners if needed [76]. There are a number of challenges involved in automating the process of plant Genotyping and Phenotyping, e.g. reproducibility of experiments, high throughput processing of large amounts of data in various formats (e.g. structured, semi-structured and unstructured), identification of appropriate meta-data for the diverse uses of the data, collecting, abstracting, and loading data into easily accessible structures. Several frameworks such as, GenAp [105], iPlant Collaborative (or iPlant) [121], Galaxy [186, 3], and LemnaTec [1] targets different aspects of plant Phenotyping and Genotyping. These technologies also attempt to tackle other problems, such as security, workflow management and accessibility of public datasets. These frameworks offer a visual interface for composing pipelines. iPlant offers both genomic and image processing pipelines. However, iPlant offers less interactive workflow composition interface than Galaxy. GenAP is an integrated architecture for supporting genomic pipelines which basically runs Galaxy in their high-performance computing environment. LemnaTec is a desktop-based commercial application that supports a high throughput image processing pipeline called HTPPheno. Our proposed framework mitigates the issues of workflow component access and management of the existing systems on collaborative SWfMSs, which we evaluate on a cloud-based plant Genotyping and Phenotyping collaborative SWfMS.

There are some command line based tools available for composing bioinformatic pipelines such as Mothur, QIIME, and Phonix 2 [169]. However, these tools do not support collaboration among different groups of researchers. Google Dataflow [5] offers a programming model for composing pipelines. However, the target users need to have a deep knowledge about the language to make pipelines. Users without any experience of working with programming languages will find much difficulty in grabbing the programming style and syntax for developing pipelines for their own jobs. Confucius [201] is a tool for supporting collaborative scientific workflow composition. However, in this work, two different perspectives were particularly focused on provenance and reproducibility. Techniques that can help collaborate those two features were discussed. A service-oriented model is also proposed. It was finally applied upon effective concurrency support control.

4.2.2 Tools Supporting API Testing Environment

Hoffman et al. [82], in their work on Java API testing, identified three kinds of commonly used API testing techniques: in the first category, automatic input test cases are generated analyzing the program execution path of the supplied source code, in the next category, formal specifications (for example mathematical expression, algebraic functions and so on) are used for testing output for the corresponding input and in

the third category, tester's knowledge is given to automatically generate test cases. Whittaker et al. [193], proposed a method for statistical testing of software components using Markov chain model. In their proposed method, several test cases are developed using multiple probability distributions, which can generate diverse input sets for testing. de Souza et al. [45], on their study, showed the importance of knowing more about how an API works instead of just using it as a black box. From their field study, they identified several problems that a collaborative organization might face by treating APIs as some black box structure instead of knowing much about it. The similar problems can also be found for plant Phenotyping or Genotyping research organization. The problem can even be worse in case of collaborative plant Phenotyping research as it is difficult to do API testing that involves images. Because in the case of images, it is challenging to define the exact output in comparison to some text-based input-output systems.

4.3 A Motivating Example Scenario

Sally, an MSc student of computer science department started her research in developing a new microbial pipeline called Phoenix2 [169]. The pipeline works on post-processed DNA sequencing data in order to determine highly prioritized OTUs (Operational Taxonomic Units). Her research is important for determining bacterial impacts on plants growth. She has two co-supervisors, one is from computer science department and the other one from soil science department. She implemented the pipeline in Python. She used PyCharm IDE to write and execute her programs. She basically used Anaconda 3 in order to install the python 3.5 interpreter, the Jupyter notebook and other commonly used python packages such as Panda and Numpy. After implementing the pipeline, she informed her soil science supervisor about the pipeline and the supervisor wanted to test the pipeline with the data available in his local machine. As a non-computer scientist, the supervisor could not install all the necessary tools to execute the pipeline. On the other hand, Sally's computer science professor wanted to determine the performance of her algorithm on different data sets. He basically wanted to run the pipeline for three different amplicons (such as bacteria, archaea, and fungi). However, he did not have the data to run the pipeline. So he contacted the soil science professor for the data. However, the data was not available to him as well. He contacted with Genome Quebec to get the raw sequencing data for the three amplicons. After collecting the data, he was not sure how to create OTUs. The Soil Science Professor got to learn about two bio-informatic tools, such as QIIME and Mothur for calculating OTUs. Again, those were command line based tools and he found it difficult to install and use them. He contacted with University's cloud research group to install QIIME in a virtual machine. He also requested to install Anaconda and PyCharm in the VM so that he can execute Sally's pipeline. Finally, he was able to provide the data to Sally and the computer science professor. However, in order to get feedback on different sets of data, Sally was having different physical meetings for demonstrating the tool. They were also chatting in Slack to exchange different information.

In the above scenario, we see that Sally and the two supervisors face various problems (e.g. installing

software, contacting with data specialist, finding a suitable time for a meeting, running tools with unintuitive parameter setting) for executing the custom pipeline. If a web-based collaborative available were available, it would have been really easier to compose and execute the pipeline on-the-fly.

4.4 Proposed Approach

We propose a cloud-based framework for building and managing scientific pipelines for plant Phenotyping and Genotyping research. As we have discussed earlier that a multidisciplinary research area like this often faces different challenges in collaboration among different user groups. For example, an organization working on this area might encounter difficulties on creating a clear team vision, defining or assigning specific problem statements to solve for some of the research groups, identifying possible threats or opportunities in earlier stages, merging or monitoring the overall teamwork progress and so on. Considering those issues, in our proposed framework we have tried to take advantages of collaborative works from different research groups for enhancing the effectiveness and efficiency in building and managing scientific pipelines on plant Phenotyping or Genotyping. However, the roles played by different research groups can be an important factor for ensuring a successful and time-efficient solution to a given problem. So we have identified different user roles for the proposed framework that helps to understand the exact problem statements to solve for the different users. In addition to the collaborative support, the framework should be able to handle other different technical facilities in maintaining the pipelines. For example flexibility in inter-operability of different image processing or bio-informatics tools, monitoring pipeline failure or progress, contributing on the same pipeline building by multiple researchers and so on.

The Collaborative Interactive Application Methodology (CIAM) [124] that we have followed for the effective collaborative system design undergoes the following main steps: *Sociogram development*, *Responsibilities modeling*, *Process modeling* and then *Collaborative task modeling*. Sociogram creates a higher level network showing the interaction and collaboration among the identified user roles in a system. Responsibilities modeling can be divided again into two steps: firstly participation table is created showing the interaction and collaboration for solving a particular task and secondly responsibilities modeling are created for each of the identified roles for detailing all their identified tasks and interaction among different roles. On basis of those responsibilities modeling, Processing modeling then creates a transition graph showing the data flows and condition of collaboration among different user roles. Finally, the Collaborative task modeling, using the UML class diagram shows the access control of different object by different collaborative users in detail while performing some collaborative tasks that have been identified in the previous phases.

4.4.1 User Roles and Sociogram

Collaborative works involving different researchers or stakeholders face different challenges in its development phases. For example, creating a clear team vision or specifying the problems to solve by any particular

research groups can often be confusing and thus reducing the overall team potential. This problem with role conflict, ambiguous problem statements to solve or overlapping of works by different researchers cost significant amounts of time and money [22]. This type of ambiguity and unclear definition of problem statements to the worse create stress and dissatisfaction among the research team slowing down the whole process [202]. So creating several distinguishable roles and assigning some specific tasks to each of them might show many possibilities in the improvement of productivity of the whole research team. From the perspective of plant Phenotyping and Genotyping research, we identified five different roles in the proposed collaborative system. The roles with their corresponding tasks can be listed in the following ways:

1. **Data Specialists:** The users of this role are responsible for collecting and uploading the required data for plant Phenotyping, and Genotyping research. The users of this group can be of two types:
 - (a) Phenotyping Data Specialists: They provides Phenotyping data required for the research. For plant Phenotyping those are mainly raw image data captured by different mechanisms like via drones, stand-alone cameras and so on. In addition to just raw images, it might include several other metadata (i.e. geographic, weather information etc.) and may vary according to the requirements of the research groups.
 - (b) Genotyping Data Specialists: Similarly they are responsible for providing data related to bioinformatics or Genotyping. The Genotyping data might include DNA, RNA, Peptide sequences and so on as per the necessity of the research groups.
2. **Tool Developers:** The responsibility of this user group is to write image processing and bio-informatics tools for performing different tasks on the provided data sets. This particular user role includes:
 - (a) Image Processing Tool Developers: For plant Phenotyping, most of the work are done via automatic image processing and usually done by image processing researchers. For example, image processing tools might include tasks like image registration, stitching, segmentation, clustering and also several other feature extractions out of those processed images (e.g. plant growth measurement, flower counting and so on).
 - (b) Bio-informatics Tool Developers: The users of this role develop tools for plant Genotyping which are different Bio-informatics scripts in most of the cases, for example, Bowtie2, BWA, nvBIO, Fasta, Fastq and so on.
3. **Pipeline Composers:** The role of this user groups are developing pipelines by combining different image processing tools. Building pipeline with optimal settings can be very important for better outputs out of the provided data sets. The users of this group can be of two types:
 - (a) Image Analysts: They compose pipelines out of the written image processing tools from another research groups. These pipelines will usually extract important information for analysis. For

example identifying plants with possible lower growth, detecting any possible plant disease via automatic image processing and so on.

(b) Bio-computation Analysts: Similarly they are responsible for developing pipelines for plant Genotyping from available Bio-informatics tools written by other research groups.

4. **Plant Scientists:** Plant scientists apply the complete developed pipelines for analysis purpose on the data sets as end users of the system. The user group includes:

(a) Geno and Pheno Mapper: They work on analyzing any possible correlation between Phenotyping and Genotyping of certain plants.

(b) Agronomist: Agronomists uses the developed pipelines by other research groups for automating different tasks and analysis to improve the overall production of agriculture. For example effect of particular fertilizers on certain plant growth, factors affecting good flowering of the plants and so on.

5. **Admin:** The users of this role are responsible for maintaining the whole research team. For example assigning the appropriate role to different team members of the research group, arranging some events and so on.

After the identification of the user roles, we developed the Sociogram for the proposed method as shown in Figure 4.1.

4.4.2 Responsibilities and Process Modeling

After defining the possible user roles for collaborative Phenotyping and Genotyping, we then identify the possible interaction and collaboration among the user. We used CIAM (Collaborative Interactive Application Methodology) [124], to design the process model for the collaborative system of the proposed method. For process modeling, we chose to use CIAM, because of its support for collaborative system design. However, a successful process modeling for ensuring all the important interaction and collaboration among the users requires detail information about the user's responsibilities. So before designing the process model, we first investigated detail individual or collaborative responsibilities of different identified user roles using 'Participation Table' and 'Responsibilities Modeling'. Participation table helps to get a higher level of abstraction about the individual and collaborative responsibilities. We then used Responsibilities Modeling for detailing the responsibilities on the basis of different user roles.

Figure 4.2, shows the designed participation table for the proposed method. A mark in the table cell (T_i, R_j) denotes the participation of role R_j for completing the task T_i . Marking the entire table for each of the tasks and roles helps us identifying the work type (i.e. individual or collaborative). The last column of the table shows the identified work type for the corresponding tasks. We identified seven higher level abstract tasks from the participation table in sequence. At first, Admin assigns appropriate roles to different users of

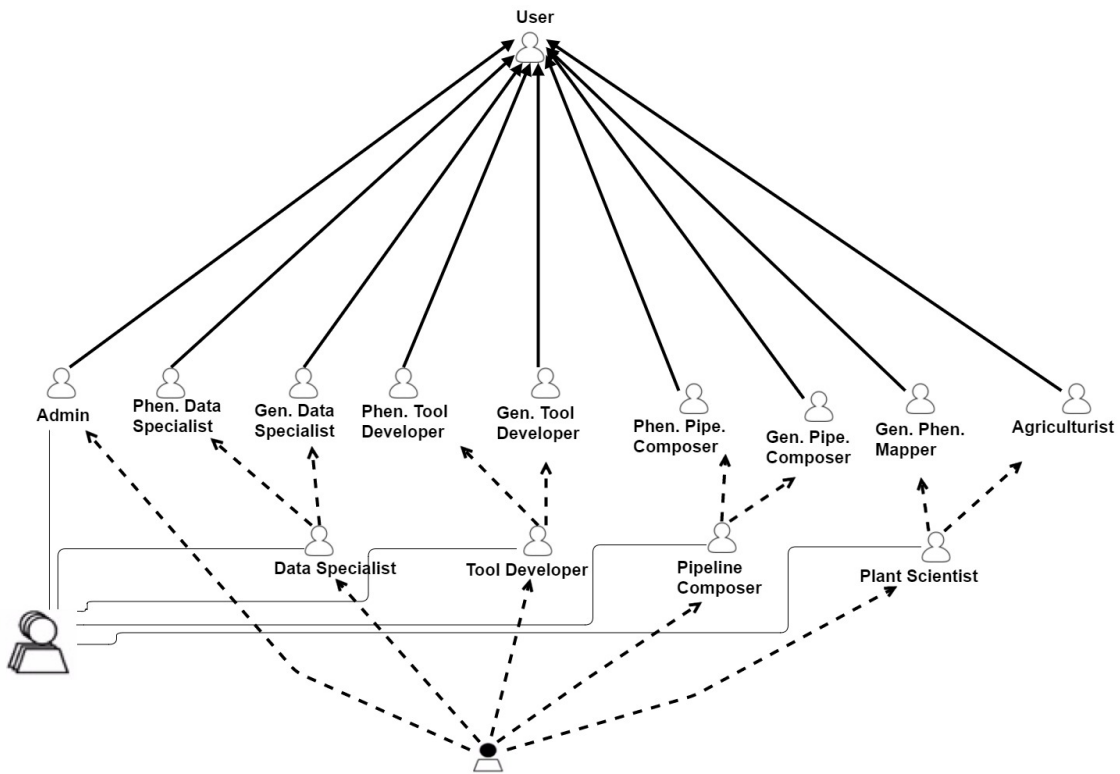


Figure 4.1: Sociogram for the proposed Collaborative Plant Phenotyping and Genotyping.

Tasks \ Roles	AD	DS	TD	PC	PS	Work Type
Assigning user roles	X					
Event Creation	X					
Defining the problems to solve by the team	X	X	X	X	X	
Data Management		X				
Tool Development		X	X			
Pipeline Composition			X	X		
Analyzing the Result for production				X	X	

* AD: Admin, DS: Data Specialists, TD: Tool Developer, PC: Pipeline Composer, PS: Plant Scientists

Figure 4.2: Participation Table for the Proposed Collaborative System.





Responsibilities	Task Type	Object (Domain Model)	Prerequisite	
			Task	Data
Assigning User Roles		<u>R/W</u> : User	User Registration	User
Event Creation		<u>C</u> : Event	Assigning User Roles	User
Event Management		<u>R/W</u> : Event	Event Creation	Event
Defining the problems to solve by the team		<u>R/W</u> : Event	Event Creation	Event

Figure 4.3: Responsibilities Model for Admin.





Responsibilities	Task Type	Object (Domain Model)	Prerequisite	
			Task	Data
Initializing research data (meta and raw data structure as per the team requirement)		<u>R/W</u> : Research Data	Defining the problems to solve by the team	Research Data
Uploading Research Data		<u>R/W</u> : Research Data	Initializing research data	Research Data
Management of Research Data		<u>R/W</u> : Research Data	Uploading Research Data	Research Data
Defining the problems to solve by the team		<u>R/W</u> : Event	Event Creation	Event

Figure 4.4: Responsibilities Model for Data Specialist.

the team. Admin also creates different event as per the team requirements. The work type of both of the tasks is 'individual'. Once an event is created, the whole team collaboratively decides on the next tasks to solve by the team. The event might undergo several iterations to clearly define the problem definition or deciding different other factors to consider by the team members including nature and formats of data or metadata for the research team. According to the requirement Data specialist then does the task of data management as an individual work type. On completion of problem definition and availability of data set in the right format, the next two phases are collaborative works for developing tools and pipelines respectively. Once the pipeline is ready for testing, plant scientist uses them to analyze or map result in between Phenotyping and Genotyping results. Pipeline composer collaboratively helps plant scientist in the last phase for pipeline management aiming towards getting the expected results and summarizing the feedback for next events.

From the participation table, we got a higher level of abstraction of the tasks for users of different roles and necessary collaboration among. From the participation table entry, we now map the tasks to all the corresponding roles to help to detail the role basis responsibilities using responsibility modeling. In addition





Responsibilities	Task Type	Object (Domain Model)	Prerequisite	
			Task	Data
Developing Research Tool		<u>C</u> : Research Tool	Event Creation	Research Data
Tool Management (simple versioning or on feedback from users)		<u>R/W</u> : Research Tool	Developing Research Tool	Research Data
Pipeline Management (in collaboration with Pipeline Composer)		<u>R/W</u> : Research Pipeline	Developing Research Pipeline	Research Pipeline, Research Data
Defining the problems to solve by the team		<u>R/W</u> : Event	Event Creation	Event

Figure 4.5: Responsibilities Model for Tool Developer.

to investigating detail responsibilities for a particular role, their access control (R, Reading; W, Writing; C, Creation) for different objects and any prerequisites tasks or data are also identified for analyzing the collaborative dependency among different user roles. For example Figure 4.3 shows the responsibility model for Admin. From responsibility model of Admin, we first one listed is 'Assigning User Roles'. The task type is individual and Admin gets Read and Write (i.e. R/W) access on the 'User' object. The task also has prerequisite task (i.e. 'User Registration') that needed to be performed by other users and a prerequisite data about corresponding User to work on. The second task on the list from Admin responsibility model is 'Event Creation'. This is also an individual work for Admin role. The task has 'Creation' (i.e. C) access control that enables him to create a new object of type 'Event'. Similarly, the next task (i.e. 'Event Management') is about event management and of the previously created 'Event' object. Finally the last task from Admin responsibility model - 'Defining the problems to solve by the team' - is a collaborative task that has a prerequisite task (i.e. 'Event Creation') and data (i.e. 'Event' object). Admin has both the read and write access control on the 'Event' object for collaborating with the team on some decision making. Similarly, we identified responsibilities for all the other user roles and have been shown in their corresponding responsibility model in Figure 4.4, 4.5, 4.6 and 4.7.

4.4.3 Tool Design

After we identified different user roles and designed the system model for their necessary collaborative support, we now focus on designing the tools to ensure the shareability, usability or execution of newly created tool on-the-fly by different user roles. As the different research groups of Phenotyping and Genotyping are inter-dependent, each of the groups likes to take advantages from use-able codes or resources from other groups for




Responsibilities	Task Type	Object (Domain Model)	Prerequisite	
			Task	Data
Developing Research Pipeline		<u>C</u> : Research Pipeline	Developing Research Tool	Research Tool, Research Data
Pipeline Management (in collaboration with Tool Developer and Plant Scientists)		<u>R/W</u> : Research Pipeline	Developing Research Pipeline	Research Pipeline, Research Data
Defining the problems to solve by the team		<u>R/W</u> : Event	Event Creation	Event

Figure 4.6: Responsibilities Model for Pipeline Composer.




Responsibilities	Task Type	Object (Domain Model)	Prerequisite	
			Task	Data
Pipeline Management (in collaboration with Pipeline Composer)		<u>R</u> : Research Pipeline	Developing Research Pipeline	Research Pipeline, Research Data
Analyzing Results using Pipelines		<u>R</u> : Research Pipeline	Developing Research Pipeline	Research Pipeline, Research Data
Defining the problems to solve by the team		<u>R/W</u> : Event	Event Creation	Event

Figure 4.7: Responsibilities Model for Plant Scientist.

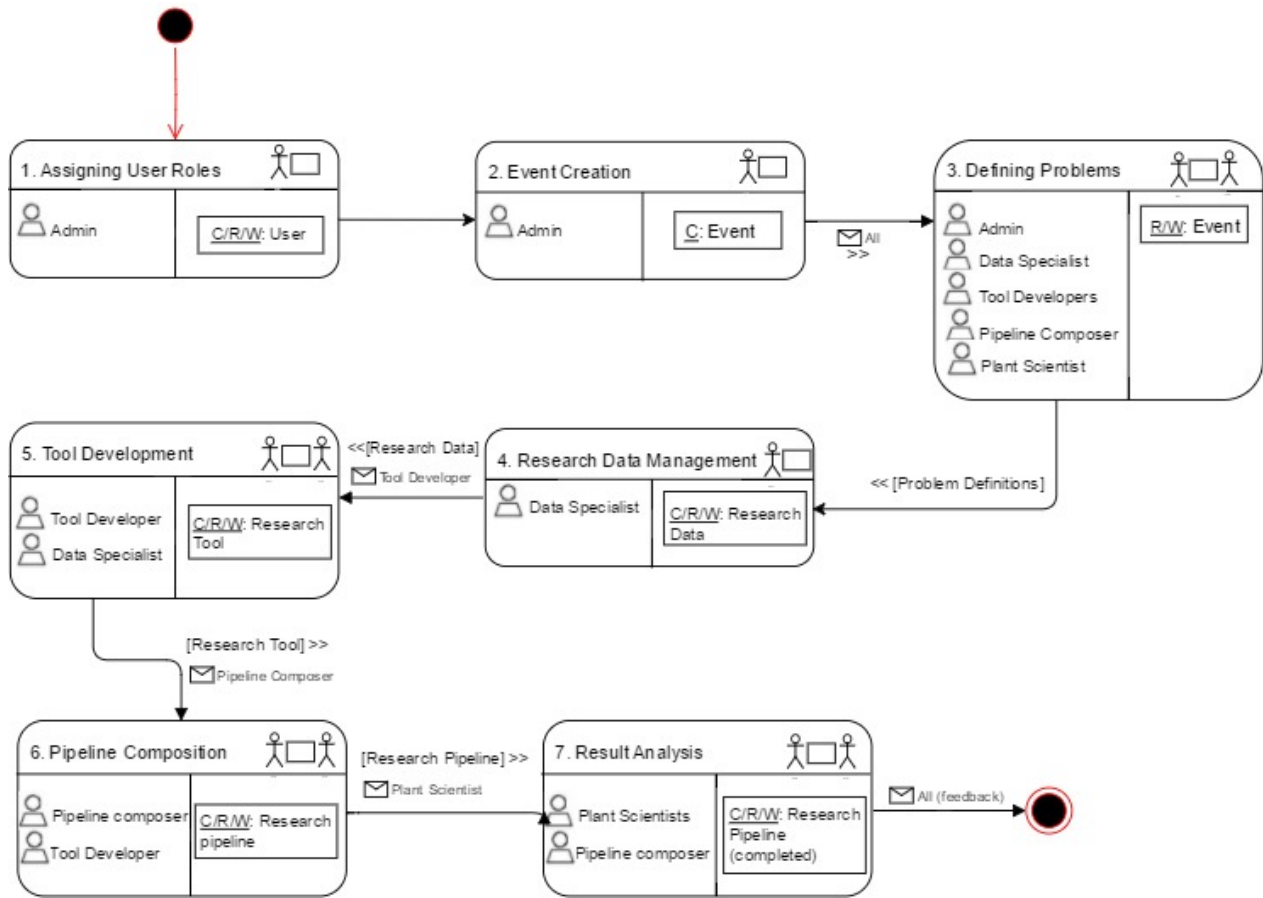


Figure 4.8: Process Model for collaborative plant Phenotyping and Genotyping.

testing their works on real data instead of using some dummy data. However different scattered development and varying code structure makes it much challenging for such support on-the-fly among the groups. It also becomes difficult to visualize the overall work progress by merging all the scattered works done by different research groups. So on-the-fly shareability or providing some easy ways of using and testing of inter-groups' works and the corresponding feedback might open up several opportunities for exploiting the advantages of collaborative work more efficiently and effectively. In our proposed method we used a common framework for each of the Phenotyping or Genotyping tools so that they become easier to integrate, share or use by different groups on the development phase. As each of the Phenotyping or Genotyping tools are targeted to perform some particular tasks it gives us the opportunity to modularize them abstracting their implementation details. For example, image processing tools for plant Phenotyping might include different tasks like pre-processing (i.e. noise removal, extraction of some particular color channels, image registration, image segmentation etc.), feature extraction (i.e. key point extraction, edge detection, flower counting etc.) and so on. On the other hand, plant Genotyping tools might target finding different information from the supplied sequences. For example like 'BLAST' (Basic Local Alignment Search Tool) are some set of tools for comparing sequences against a protein or DNA sequence database, 'COPIA' (COnsensus Pattern Identification and Analysis) works on discovering motifs from given protein sequences and so on. These task-specific nature of the tools make them easier to modularize and test the corresponding outputs for given some inputs. Besides to ensure the inter-usability of the tools we applied a common layer on top of the individual tools. Hiding all the implementation details of a tool, the applied layer takes two types of information: input/output destination and different settings that are required throughout that particular tool. Depending on the tool task specification the input/output can be some file systems (i.e. raw image, files with DNA sequences etc.), entire directories with all different contents or from/to other tools. On the other hand, the settings are different parameters required throughout the implementation of that particular tool. Figure 4.9 shows this abstraction layer of the main modular implementation. This common layer gives the support for collaborative usage of the developed tools among users of different roles and helps in developing pipelines by connecting tools in different orders.

4.4.4 Pipeline Composition

As shown in the process model (Figure 4.8), on the availability of research tools in the system, pipeline composer in collaboration of the tool developer composes pipeline aiming to solve some predefined problems for plant Phenotyping or Genotyping research. As the tools were abstracted hiding main implementation detail in the previous phase, customized pipeline development becomes straight forward and easy in this steps. Pipeline composer chooses the required tools from the library of already developed tools, defines streaming or execution sequences and tunes settings for expected output to develop the required pipelines as shown in Figure 4.10. As this step is collaborative, pipeline composer and tool developer exchange their feedback for possible modification of any of the tools or pipelines. On getting the expected result, pipeline composer

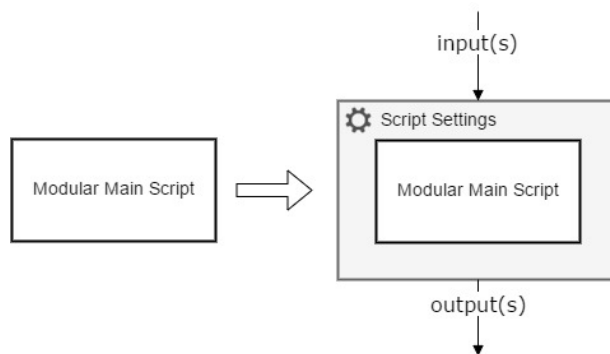


Figure 4.9: Abstraction Layer on modular Tools for collaborative use.

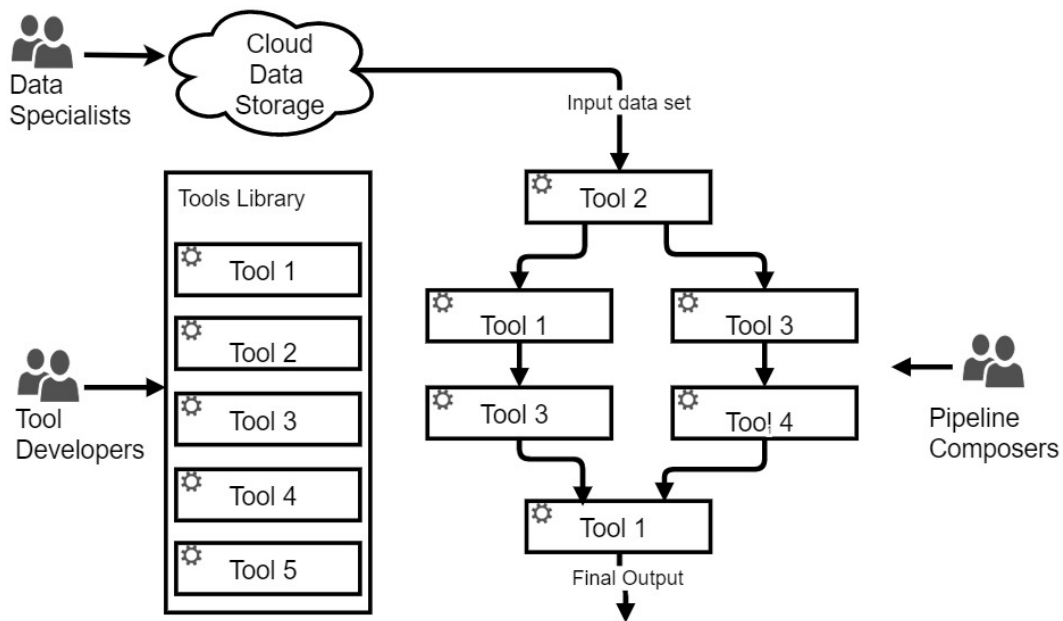


Figure 4.10: Collaborative Pipeline Design using modularized tools.

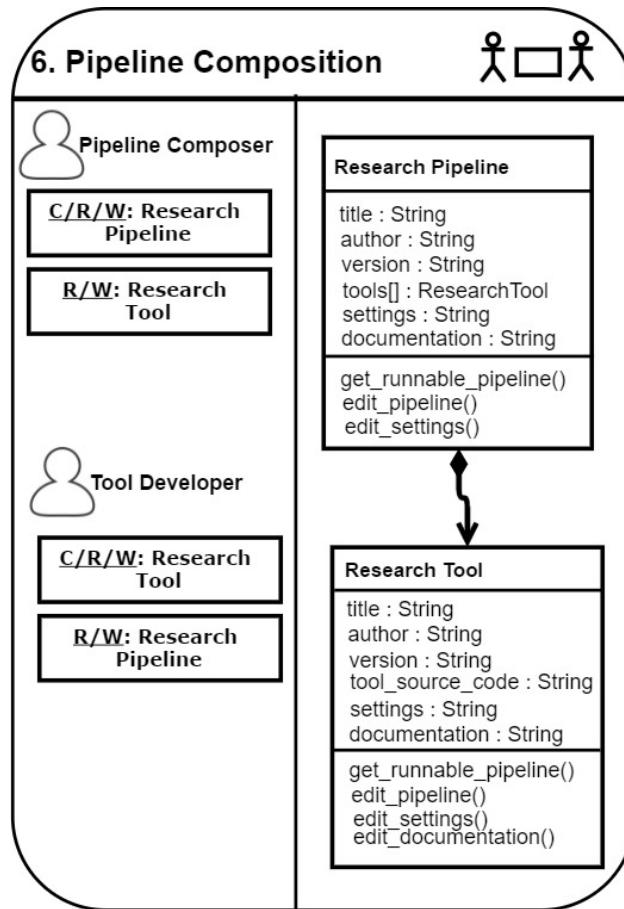


Figure 4.11: Collaborative Task Modeling for Pipeline Composition.

shares the saved pipeline with plant scientist.

4.5 Evaluation

We evaluated the framework in two different ways. First, we demonstrated the usage of the framework in terms of an example case study where showed collaborative development and management of a pipeline. Second, we involved real users in the usage of the framework.

4.5.1 Case Study: Collaborative Development and Management of a Pipeline

In this section, we show an example case study for the development and management of a pipeline that involves the participation and collaboration of different roles of users using the proposed method. We choose the example with a very simple Phenotyping pipeline for keeping focus on the proposed method but the same method can be applied for developing more complex pipeline both for Phenotyping and Genotyping. We then demonstrate our prototype implementation of the proposed method.

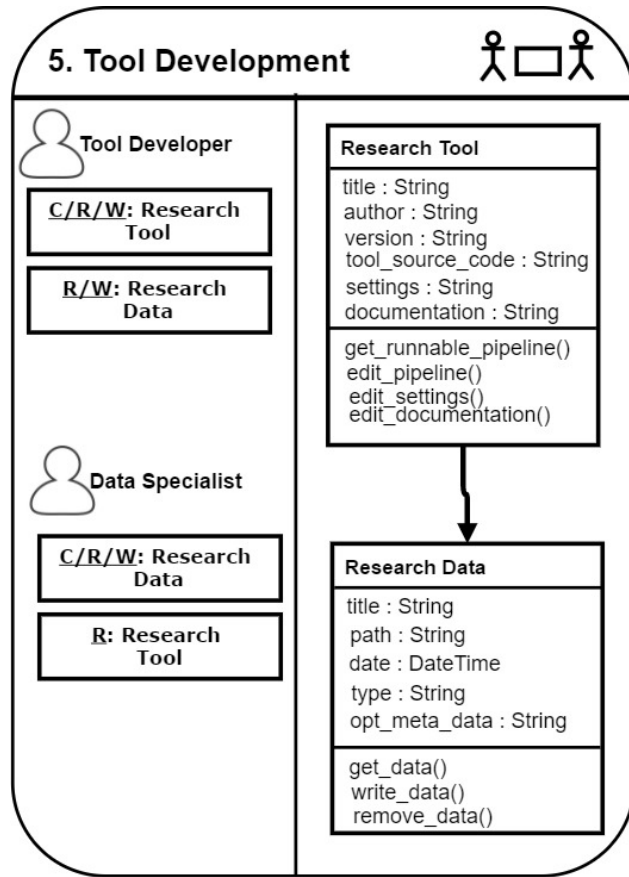


Figure 4.12: Collaborative Task Modeling for Tool Development.

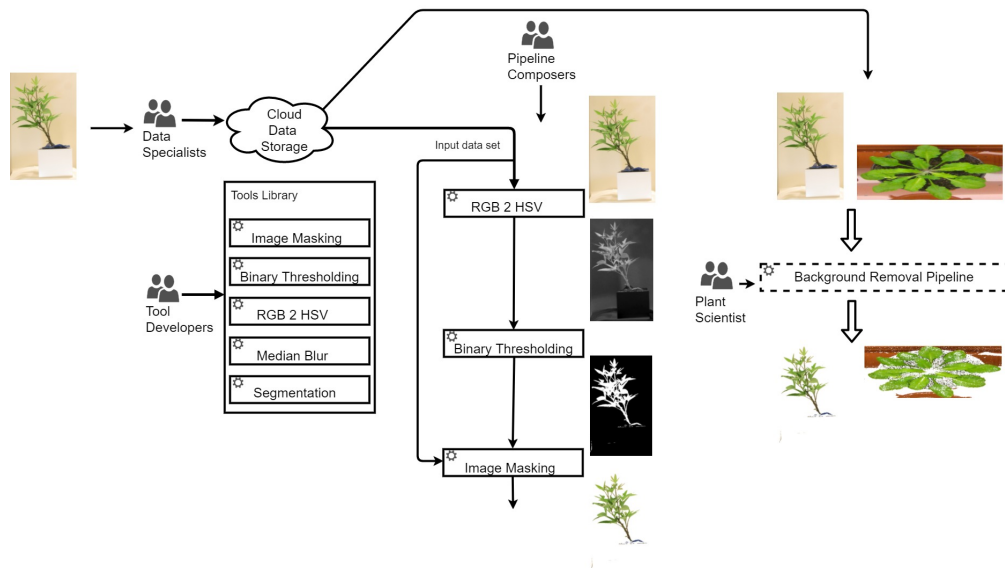


Figure 4.13: Usage example for collaborative work by the proposed method for Phenotyping.

From system to system plant Phenotyping might follow different methodologies for image acquisition of the plants. For example, Hartmann et al. [76], in their work on plant Phenotyping considered two different views for capturing plant images for extracting some of the plant phenotype information (e.g. plant height, width and so on) via image processing. Each of the plants under experimentation is placed on top of a conveyor belt. As the conveyor belt moves, two standalone cameras: one situated at the top and other at a side, capture the images of the plants. In addition to just the plant, the captured image also contains unnecessary background information (e.g. wall, part of conveyor belt etc.) that we need to remove for further image processing.

For keeping things simple, we assume that at this point the research team decides to work on the pre-processing step. As shown in Figure 4.13, the data specialist collects and uploads the similar captured image to the cloud storage. There are several image processing algorithms available for image background removal that the team might decide to use. For example, Fahlgren et al. [51], in performing a similar task in their Phenotyping work first converted the supplied RGB channel image to HSV channel and then extracted only the saturation channel from it. The resultant image was then binary thresholded to some value and finally masked with the original image to get the background removed image. Tool developer accordingly develops different tools which are then collaboratively used by pipeline composer (Figure 4.13) for solving the problem for the given scenario. The tool development and pipeline composition allow the users for testing the outputs for each of steps (as the intermediary or final image results shown for pipeline composition in Figure 4.13 were generated using the prototype of the proposed method). Once an initial implementation of the pipeline is completed, it is shared with Plant Scientist for testing. As seen in the figure, the pipeline is then used directly on the data set from the cloud storage by plant scientist abstracting all the intermediary phases of implementation. However, the pipeline can also be edited by plant scientist for the required output by tuning the available settings of the pipeline. Besides plant scientist also can give feedback on-the-fly to the pipeline composer for any issues or possible modifications which can then be collaboratively solved by the team.

We have implemented a prototype of the proposed system. It is a cloud-based system hosted in Linux Server. We used Python 2.7 for the server side coding. On the other hand HTML5, CSS and JavaScript were used for client-side programming. We also used Ajax for making some asynchronous server communications. For the use cases, we collected source code from different research student working on plant Phenotyping or Genotyping. Most of the Phenotyping code was written using OpenCV2 (<http://opencv.org/>), an open source library for image processing. On the other hand for Genotyping, the source codes were mainly written using commonly used python packages such as Panda, Numpy and so on. We used plug-in based architecture [153] in order to integrate the Phenotyping and Genotyping on-the-fly, i.e. tool developers do not need to recompile the platform for this purpose.

Figure 4.14, shows a sample interface of the prototype of the proposed method. The prototype has been tested for integration and execution for both Phenotyping and Genotyping work. In the figure, the panel which has been labeled as 'A' contains the listing of all the available tools developed by both Phenotyping

and Genotyping. Listing of all the saved and shared pipeline with this particular user is also accessible from this menu. As the assigned role for this user is 'Phenotyping Tool Developer' (as can be seen at the top of this panel), he can collaborate with pipeline composer to compose the pipeline and so the option for composing pipeline is also available for this user (entitled as 'Design Pipelines' in panel 'A'). The panel 'B', gives the support for developing or customizing pipeline by making use of different available tools (e.g. from panel 'A'). As seen in the panel 'B', the individual tools come with corresponding documentation, settings and source code. The settings for any of the tool can be tuned for getting the required output. Besides from here, one can communicate with the corresponding tool author, edit code and so on as per the role assigned to the user. The panel 'C' contains information about all available and uploaded data set by the user of role 'Data Specialist'. The pipeline can be executed for testing the outputs (we tested with real research data and pipeline for both Phenotyping and Genotyping as discussed in detail in 'User Study' section) or saved from the panel 'D'. The user can communicate with other users of different roles from panel 'E' for any collaboration or issues. The online or offline status is shown next to the users of different roles. Users of different roles can start live chatting (individual or group) to discuss or solve any issue collaboratively as shown in panel 'G'. Panel 'F' shows the corresponding notifications of such events.

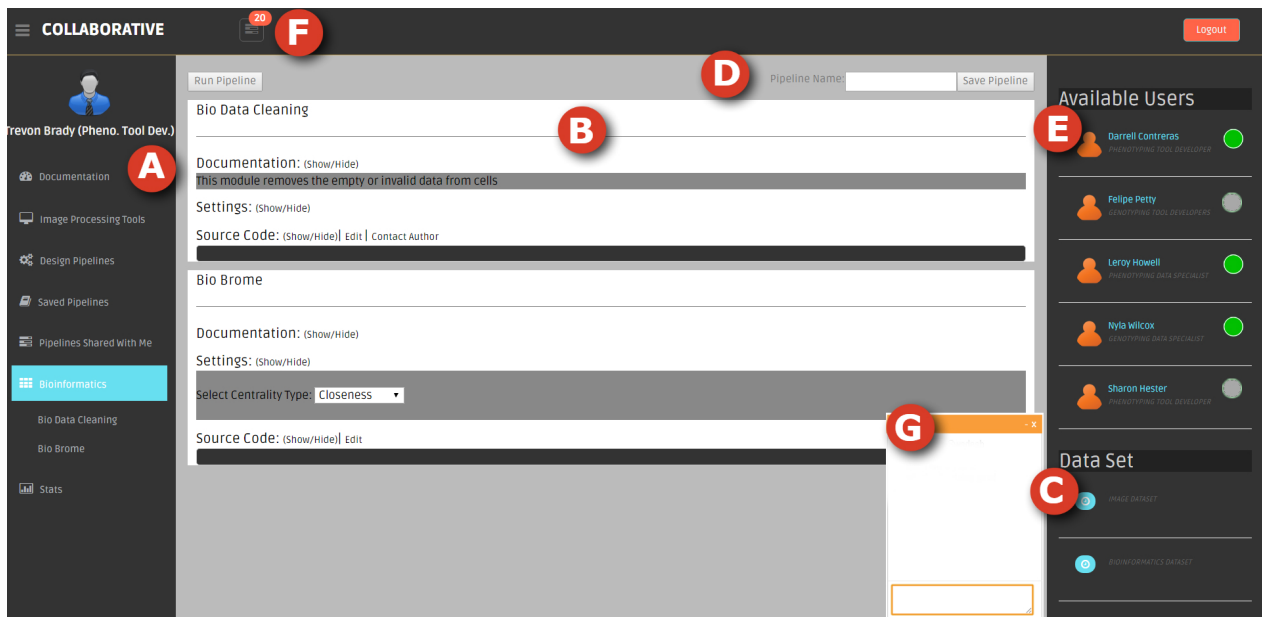


Figure 4.14: Process Model for collaborative plant Phenotyping and Genotyping.

Now, for example, a tool developer for testing one of his newly developed tools requires some updates on the data set or might need to include some meta that are currently not available. He can look for the required data specialist from panel 'E' and can instantly communicate via live chat or sending messages. On getting the data the tool outputs are tested on the cloud and if expected output is found the tools are shared with other users. Pipeline composer on the other hand collaboratively works on a pipeline which can then

be shared with Plant scientist. The plant scientist also communicates with the developers, share the outputs or give feedback to the other users. All the users thus can work on their own area, yet sharing and making use of others' work and thus accelerating the work progress collaboratively.

4.5.2 User Study

In our initial version of user study, we involved three kinds of stakeholders such as an image tool developer, and a bio-informatician and a plant scientist. In the following, we discuss their experience with the tool.

1. **Bio-informatician:** This user is a professor of soil science. He was using a command line based tool for running for post-processing of microbial DNA sequencing data. In order to run the pipeline, he was required to install Anaconda, Pycharm, and Jupyter. However, as he was not that expert in setting up a programming environment, it became a very difficult task for him. Eventually, he was not able to install all the necessary tools and ended up corrupting system files and leaving his laptop Python free. We demoed the tool to the bio-informatician by integrating the command line pipeline tool. He reacted with overwhelm to see the capability of the pipeline composition tool. He commented that "What a fantastic platform you and your team have developed. I think it will fit the bill exactly".

The bio-inforamtician has experience using QIIME and Galaxy. He liked Galaxy but he mentions about three problems with Galaxy: one is that it lacks Phenotyping interface, second it does not provide control over allocating resources, and finally, he has to compose pipelines without having the collaboration with others. He shared with us that he is not comfortable working with QIIME as it is a command line based tool and he faces a lot of difficulties using them, as he is not from computer science background.

2. **Agronomist:** The involved user for this possible role was a research student working on Canola plant Phenotyping. She expressed her difficulties on getting and secure storage of lots of data set on her own local machine. We demoed the prototype of the proposed method and she gave very positive feedback for having such a common workspace where she can test her works and also make use of others' work easily. She also expressed that it can be very helpful for her for dataset management, as she faces difficulties with getting or secure storage of lots of data for plant Phenotyping. For example, in her feedback, she wrote - "I can store my data as a backup in case I lost them in my own system."
3. **Image tool developer:** This user, a research associate of image processing group often runs back and forth to agronomist for demonstrating newly developed tools to get their feedback. He also runs around for collecting image data from field researchers. He wrote - "I think it could be useful to test out algorithms and show the results to the Plant Scientists. It is especially useful for them to be able to run the algorithms themselves".

4.6 Conclusion

In this chapter, we presented the concept of access control techniques in the context of collaborative SWfMSs and also proposed a role based method for workflow component access controlling. We adapt the *Collaborative Interactive Application Methodology (CIAM)* [124] for our proposed method of role-based access control in collaborative SWfMSs. We evaluate our proposed method with use-cases of Plant Phenotyping and Genotyping research domain. We used various Plant Phenotyping tools (e.g., written in PlantCV API [2]) with varying user-access controls. We collected image processing tools (such as image registration, segmentation, and flower counting) from an image processing group who closely work with agriculture researchers. We collected bio-informatic tools from the bio-informatics group in the University. Feedback from different users in our user study shows promising results with the usage of proposed method of role-based access control technique in terms of collaborative SWfMS. While locking scheme facilitates consistency management in real-time collaborative workflow composition (e.g., as we discussed in previous chapter; Chapter 3), the discussed role-based access control technique discussed in this chapter can be used for overall management of workflow components among collaborators in a collaborative environment of data analysis. Our study results reveal that the role-based access control on the workflow components are one of the primary requirements for the management and handling of the workflow components among collaborating groups. The role-based access control technique also demonstrates its aid towards the locking scheme, e.g., by access permission on workflow components by different collaborators.

Leveraging the locking scheme for real-time collaboration (e.g., as presented in Chapter 3) and role-based access control technique (e.g., as presented in this chapter), we propose an architecture of collaborative SWfMSs, — which we discuss in the following chapter (e.g., Chapter 5).

5 SciWorCS- TOWARDS A COLLABORATIVE SCIENTIFIC WORKFLOW MANAGEMENT SYSTEM

Collaborative SWfMSs often have a different set of challenges and requirements in contrast to the single user based SWfMSs [114, 199]. Research studies on this domain show different such challenges and requirements, for example—handling independent sub-workflow execution, *backdoor communication* among the sub-group collaborators [199], maintaining relationship between scientific workflows and collaboration models [114]. In an attempt to address these challenges and requirements, in this chapter we present our proposed architecture towards a collaborative SWfMS. In our proposed architecture, we leverage our *fine-grained locking scheme* and *role-based access control* (e.g., as presented in Chapter 3 and 4 respectively) for the management of the collaborative environment. As a proof of concept of the proposed architecture, we also implement a collaborative SWfMS— *SciWorCS*. We evaluate SciWorCS with different scientific workflows of Bioinformatics, Software Repository Analysis and Machine Learning based Classification— where it demonstrates promising results and significant potentials.

In this chapter, we first present the importance and motivation of our proposed architecture of collaborative SWfMSs in Section 5.1. In Section 5.2 we present the related works on this research domain. We then present our proposed architecture in Section 5.3. Based on the proposed architecture of a collaborative SWfMSs, we implement *SciWorCS* — a collaborative SWfMS. We discuss the implementation details and technical features of SciWorCS in Section 5.4 and Section 5.5 respectively. We present different use-case evaluations in Section 5.6 and finally we draw conclusion in Section 5.7.

5.1 Motivation

A number of SWfMSs have been proposed and developed in recent years to facilitate the scientific experiments [114, 111]. For example some of the popular SWfMSs are: Taverna [140], Galaxy [66], Kepler [115], Pegasus [47], VisTrails [31], Triana [181], VIEW [109] and so on. The existing SWfMSs support only single user for a given data analysis process [201, 199, 56], however, modern scientific research projects are often collaborative in nature —involving multiple researchers of diverse domain and expertise [114, 201, 165, 167]. For example, the *Large Synoptic Survey Telescope (LSST)* [113] experiment requires a collaboration of around 1,800 scientists and engineers, the *Cancer Biomedical Informatics Grid (caBIG)* [133] project by the *National Cancer Institute (NIC)* aims to accelerate the domain research by collaborating entire research community and so

on, as referred by Zhang et. al [201] and Lu et. al [114] respectively in the similar context of collaborative SWfMSs. Besides, some scientific domains essentially require collaboration as they are highly correlated among multiple research disciplines [201]. These use-cases hence motivated several research studies towards collaborative SWfMSs in recent years [114, 165, 167, 168, 166, 199, 200]. Studies show that Collaborative SWfMSs often have a different set of challenges and requirements in contrast to the single user based SWfMSs [200, 114]. For example, collaborative SWfMSs raise the challenges of consistency management (e.g., as discussed in Chapter 3) in the face of concurrent conflicting operations [114, 165, 167, 168, 166, 199, 200]-like other *Computer-Supported Cooperative Work (CSCW)* based collaborative systems, such as collaborative document writing [142, 144], graphics editing systems [37, 60] and so on. In addition to the consistency management, there are several other aspects and challenges those we need to consider for a successful collaborative data analysis environment in the context of SWfMSs. For example, Zhang [199, 201] presented that, often there can have several sub-groups working collaboratively on different sub-workflows of entire scientific workflows. A collaborative SWfMS needs to handle such independent sub-workflow execution and back-door communication among the sub-group collaborators [201]. Lu et. al investigated the possible challenges towards a collaborative SWfMSs, such as maintaining a collaborative provenance model, the relationship between scientific workflows and collaboration models, and so on [114].

For our proposed architecture we tried to address such studied challenges in the context of collaborative SWfMSs. As a proof of concept of the proposed architecture, we developed SciWorCS — a Collaborative Scientific Workflow Management System. We present different experimental use-cases for the evaluation of the proposed architecture.

5.2 Related Works

Workflow management systems were initially adopted by business community. The workflow management systems mainly follow two types of architectural approaches: *Service Orchestration* and *Service Choreography* [12]. Several process modeling languages and techniques thus has been proposed and developed by the community [12], such as: *Business Process Execution Language (BPEL)* [6], *Yet Another Workflow Language (YAML)* [187], *WfMOpen* [110] - an open source workflow engine, *Web Service Choreography Description Language (WS-CDL)* [100] and so on.

Though the scientific workflow management systems uses the similar concepts, unlike the business workflows that works usually at a programming language level, the scientific workflows generally operate on a higher abstract layer. Scientific workflows are composed of re-usable modular tools (e.g. different analytical steps) in different combinations to prove a scientific hypothesis and are widely used for data analysis, simulation, visualization and so on [12, 111]. The examples of some popular modern scientific workflow management systems are: *Taverna* [141], *Galaxy* [66], *Kepler* [115], *Pegasus* [47], *VisTrails* [31], *Triana* [181], *VIEW* [109], *DiscoveryNet* [152], *Chiron* [138], *GridNexus* [24] and so on.

Taverna [141] is an open source workflow management system that follows a service oriented architecture. Each of the component of *Taverna* is either a task specific Web service or a processor [141, 201]. It uses a high level XML-based conceptual language called *Simple Conceptual Unified Flow Language (SCUFL/XSCUFL)* [139] for defining a scientific workflow comprising of different services and datalink relation among them. Java Beanshell is used for scripting the corresponding services by this workflow management system.

Galaxy [66] - a web-based workflow management system, designed with a goal to ease the complex data analysis process with intuitive GUIs for managing the datalink relation among the tasks, which is widely used for high-throughput DNA sequence analysis in recent time. However, the architecture of the workflow system allows data analysis of different domains (for example, image processing) with corresponding tool integration [3]. *Galaxy* is implemented using Python programming language. It uses simple XML for storing different tool information, such as: tool configurations, datalink relation, input validation criteria and so on. *Galaxy* presents a simple web-based UI to the users by rendering the corresponding tool XML [3].

Kepler [115] workflow engine is built on top of *Ptolemy II* [28] and uses actor-oriented design. Much like the services of *Taverna*, actors of *Kepler* workflow system are re-usable modular blocks that are responsible for specific computations [12]. The *actors* are linked in different combinations for solving a given problem whereas a *director* controls and monitors the executions of the such actors.

Similar to *Galaxy*, *Triana* [181] workflow management system provides intuitive graphical UI features, such as: dragging tools, visual datalink connection, zooming functionalities to the workflow components and so on. *Triana* supports multiple languages [12], for example: *Web Services Flow Language (WSFL)*, *Business Process Execution Language (BPEL)* [6] and so on.

GridNexus [24] workflow management system mainly focuses the workflow execution in Grid environment. However, the designed workflows can be executed in local environments as well. *GridNexus* uses proprietary XML based language - JXPL, for defining a workflow comprising of tasks and datalink relations among them [12]. *GridNexus* GUI is built on top of *Ptolemy II* [28] for sophisticated workflow composition. *Askalon* [53] is also another workflow management system that is designed for Grid environment. *Askalon* supports Unified Modeling Language (UML) for workflow composition in the presentation layer [111].

Pegasus [47] - uses Artificial Intelligence for mapping and planning the workflow execution in distributed environments. The artificial agent targets mapping the available resource in the distributed environments, such as Grid, to the corresponding input port for a successful execution of the workflow. It uses DAX - a proprietary XML based language for representing directed acyclic graphs [12].

Besides, several scientific workflow management systems has been developed in recent years focusing on specific domains, features, high-throughput executions and so on. A few examples of such workflow management systems are: *BioWBI* [163], *GridBus* [30], *Magenta* [191] and so on. From the above discussion it is noticeable that, though the modern scientific workflow management systems often targets different specific research domains, majority of them share the similar architectures and concepts (e.g. library of independent re-usable tasks/services, building DAGs of such tasks/services and so on). For accelerating the analysis

process, the modern workflow management systems also adopt several techniques, such as: allowing high-throughput Grid execution, advanced data visualizations, intuitive UIs for workflow composition and so on [111]. However, to the best of our knowledge, none of the workflow management systems support collaborative works of the scientists directly. Workflow collaboration is done via manual sharing of the composed workflow via email or publicly shared spaces, like *myExperiment* [44] (e.g., *myExperiment* supports sharing of *Galaxy* [66], *Kepler* [115] or *Taverna* [141] workflows). Realizing this limitation, several methods has been proposed in last few years for supporting real-time collaborative workflow management systems [199, 201, 56, 165, 164].

5.3 SciWorCS Architecture

The data analysis is powered by set of reusable computational modules, which are integrated to SciWorCS as different plugins by the collaborators. The collaboratively designed workflows for the data analysis are scheduled and executed by the collaborators for further visualizations and analysis. Figure 5.1, illustrates the high-level architecture for SciWorCS implementation.

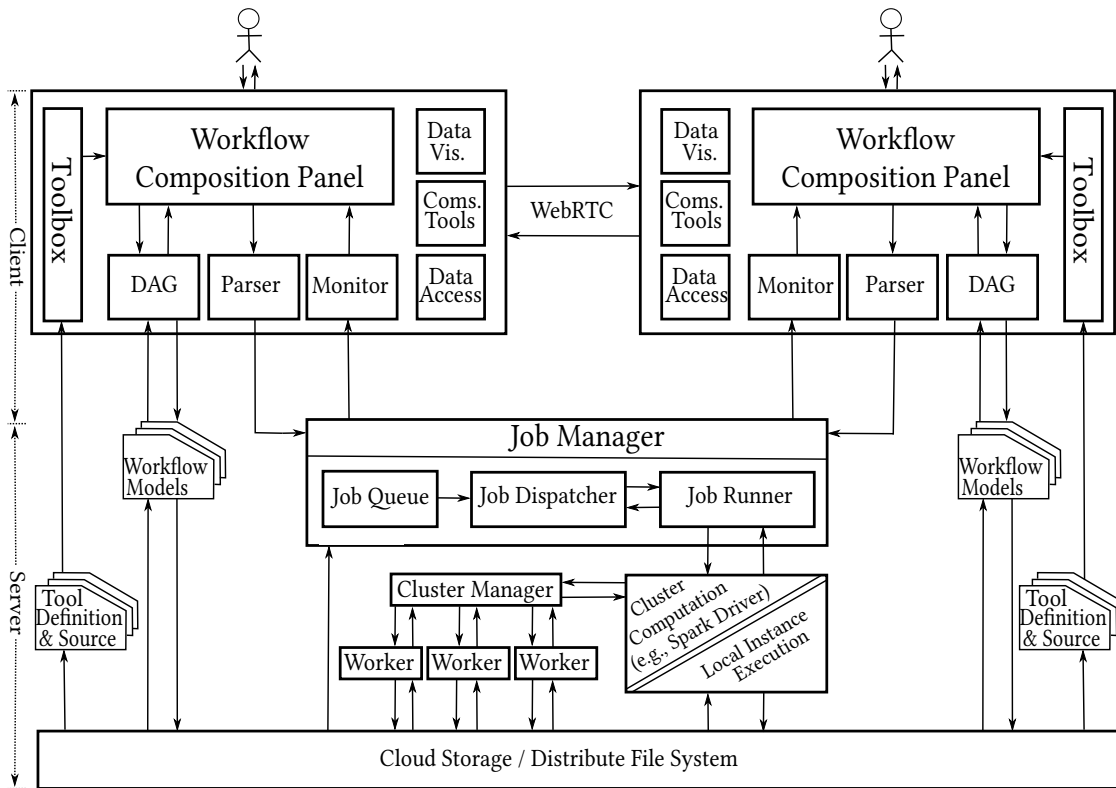


Figure 5.1: High-Level Architecture of SciWorCS.

5.3.1 Toolbox: Set of Reusable Computational Steps

A workflow for data analysis is often represented as a Directed Acyclic Graph (DAG), $W = (M, E)$, where M is a set of n finite number of modular tasks, $m_i, (1 \leq i \leq n)$ and E is a set of directed edges, $e_{ij} = (m_i, m_j), (1 \leq i \leq n, 1 \leq j \leq n, i \neq j)$ denoting the dependency relations among the modular tasks m_i and m_j [148, 62]. A module, m can be responsible for some independent computation tasks, such as any module, m_i can be responsible for some statistical analysis on a given dataset, while another module, m_j can be responsible for applying a machine learning model on the dataset and so on. The computation of any module, m_l is often further customized (e.g., number of *nodes*, *hidden layers* in case of a Neural Network Classifier and so on) by its corresponding configuration sets, C_l - a set of \mathcal{P}_l available parameter configurations, $c_i \in C_l, (1 \leq i \leq \mathcal{P}_l)$. Hence, a SciWorCS module is generalized as Definition 5.3.1.

Definition 5.3.1 (SciWorCS Computational Module). *A computational task module is generalized as a tuple, $m = \langle id, I, O, C, S, T \rangle$ where id is the unique identifier of the module in a workflow, I and O are the corresponding set of supported input and output data formats respectively by the module, C is a set of \mathcal{P} different parameter settings or configurations, S is the modular source code that executes based on I, O, C and finally, T is the set of possible states of the module (i.e., ready, running, success, failed, aborted) in an execution.*

The generalized definition of SciWorCS computational modules, allows collaborators to plugin, reuse and share the set of tools among the collaborators for a given data analysis task. For example, Listing 5.1 and Listing 5.2 demonstrate a sample computational task module. Listing 5.1 is the module definition containing the meta data (i.e., I, O and C sets information) in XML format for its corresponding modular source code, S . The id and T of the module are dynamically generated at the workflow composition and execution phases respectively.

```
1 <SciWorCS>
2   <toolInputs>
3     <toolInput> ... </toolInput>
4     <toolInput> ... </toolInput>
5     ...
6   </toolInputs>
7
8   <toolOutputs>
9     <toolOutput> ... </toolOutput>
10    <toolOutput> ... </toolOutput>
11    ...
12  </toolOutputs>
13
14  <toolConfigurations>
15    ...
16  </toolConfigurations>
```

```

17
18 <toolDocumentation>
19 ...
20 </toolDocumentation>
21 </SciWorCS>

```

Listing 5.1: An Example of SciWorCS Module Definition Template.

```

1 ##imports for the modules
2 #...
3
4 ##loading input dataset from references
5 #...
6
7 ##configuring the module as per the module definition
8 #...
9
10 ##module implementation
11 #...
12
13 ##writing out the output to references as per the module definition
14 #...

```

Listing 5.2: Sample Module Source Structure and Binding for the Definition in Listing 5.1.

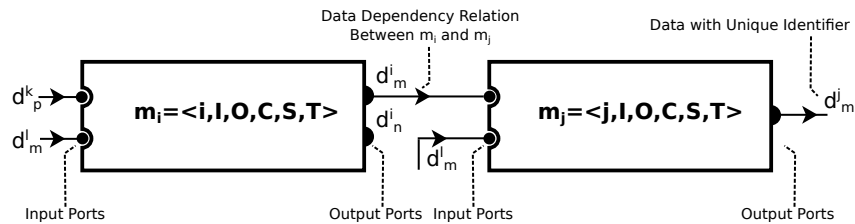


Figure 5.2: SciWorCS DAG Formulation for Workflows.

5.3.2 DAG Formulation for a Data Analysis Workflow

While a computational module from the toolbox is responsible for a given data analysis or manipulation task, a set of such modules are combined together forming a workflow towards solving a more complex data analysis problem. SciWorCS follows a dataflow oriented architecture for the data analysis tasks where the modules are associated with one another by the dataflow relation among them. Fig. 5.2 illustrates a dataflow relation of a workflow between two arbitrary computational modules- m_i and m_j . SciWorCS assigns unique identifier to each of the modules comprising a given workflow, such as $m_i = \langle id : i, I, O, C, S, T \rangle$ and $m_j = \langle id : j, I, O, C, S, T \rangle$. As a module from the toolbox can be used zero to multiple times, the identifiers are used to uniquely identify all the input-output ports in a given workflow. For example, we

consider an arbitrary module- m from the toolbox, that contains two input ports- x, y , and one output port- z . We assume the module get a unique identifier- i on usage to a workflow. Hence, its input-output ports are encapsulated with the identifier, such as $m_i = \langle id : i, I : [x_i, y_i], O : [z_i], C, S, T \rangle$ -to uniquely identify in the workflow. The output dataset generated from an output port of a module is also labeled as the port identifier, which is used for forming the data dependency relation among the workflow modules. For example, as illustrated in Fig. 5.2, the module m_i is a predecessor of the module m_j where, one of its output datasets with reference- d_m^i from an output port is linked as the input dataset reference for an input port of module, m_j . The dataflow linking is also validated for the matching data format between the input and output ports of the corresponding modules. For example, we consider dataflow linking from the port- p_1 to p_2 -of arbitrary modules- m_i and m_j respectively. The linking is validated for the matching data format, $df(p)$ of the port p , such that $df(p_1) = df(p_2)$. The allowed data format for the ports are obtained from the corresponding module definition as discussed in Section 5.3.1.

For portability the generated workflow DAG definition is represented and written in a simple XML format. The workflow definition file is saved and used for later restoring for the corresponding data analysis task. Besides, the simple XML format for the workflow definition enables easier portability among different SciWorCS instances. SciWorCS uses the XML definition file for parsing the module information with corresponding configurations and data dependency relation among the modules, -which is later used for job management and execution as discussed in Section 5.3.4.

5.3.3 Collaborative Composition

The data analysis workflow, $W = (M, E)$ is composed by selecting a set of computational modules, M from the toolbox and by defining the dataflow relation, E among the modules. However, in a collaborative composition of W , the concurrent conflicting operations might results in its inconsistent states across different collaborators [173, 130, 201]. For example, we consider that at any given version of workflow, W_v there is a datalink dependency from module m_i to m_j , - defined by the directed edge, $e_{ij} = (m_i, m_j) \in E$. With the same workflow version, W_v we assume two collaborators independently execute the operations - $O_1 = updateDatalink(m_j, \mathbf{m}_m)$ and $O_2 = updateDatalink(m_j, \mathbf{m}_k)$, for updating the incoming data dependency of, m_j . Since, the two concurrent operations target the same component (i.e., workflow component, m_j) but with different attribute values (i.e., $m_m \neq m_k$), it results in inconsistency of the workflow across the collaborators [173, 174, 130]. Besides, unlike some unstructured object (such as text documents), the scientific workflows modules are often highly dependent on other modules for dataflow dependency relations [130, 201, 165]. Hence, some configurations changes on a module might highly impact the execution behavior of other modules [164, 130, 56].

Hence, we adopt locking scheme for facilitating the consistency management and access control in a collaborative workflow composition environment. Fig. ?? illustrate the SciWorCS architecture for collaborative workflow composition with *Attribute Level Locking* [130] of the workflow object components. Any

update on the current state of the workflow by a collaborative from corresponding SciWorCS composition panel is notified to the other collaborators via message passing (such as using WebRTC). The concurrent update operations are controlled and managed by imposed component locks by collaborators [201, 130, 165]. The collaborative workflow objects are finally merged as SciWorCS workflow model to the server for later reusability and execution.

5.3.4 Job Management and Execution

SciWorCS job manager is responsible for the scheduling and execution of the job (i.e., representing a modular task) from the workflow models. It manages the dependencies and execution order of the jobs to maintain a dataflow oriented execution plan - a job is ready and ordered for execution only if all of its required input datasets are available (i.e., input dataset or produced without errors from prior jobs). It also maintains a job queue for managing the multiple execution requests from the collaborators (such as, a sub-workflow of the entire data analysis steps for which a collaborator is responsible for and so on). The queued jobs are dispatched for local or cluster executions as per the configurations and implementation of the computational modular step, m . In case of cluster execution, the dispatched jobs are submitted to a cluster manager (such as, *Apache Spark* cluster manager [170]), which in turns are distributed across different worker nodes for parallel execution. The job manager also sends back the job execution status (e.g., running, success or failed) to the collaborators for real-time monitoring.

5.4 Implementation Details

We implemented the SciWorCS tool as a proof of concept of the proposed collaborative scientific workflow management system. The implementation is a cloud-based system. The SciWorCS is implemented in Python programming language (e.g., *Python 2.7*). We used Python to leverage the larger number packages and library supports for different domains of data analysis. For example, a great numbers of bioinformatics [3, 186], image processing [2, 23], machine learning [143, 171] and so on tools and libraries are recently implemented in Python for its trending popularity. Besides, the tools those are written in a different programming language can also be added to SciWorCS toolbox using Python wrapper for the command line access. We demonstrate a simple use-case of such scenario in Section 5.6.1. Python also supports writing large-scale cluster computing applications using its *PySpark* [170] API. We present a use-case for such large-scale cluster computing workflow from SciWorCS in Section 5.6.

We used *HTML5*, *CSS* and *JavaScript* for the client side programming and creating user interface for easier accessibility of SciWorCS core. SciWorCS provides an intuitive graphical user interface for defining the workflow DAG, -where the selected tools from the toolbox can be graphically connected, reorganized, zoomed in/out, modified and so on. The interactive and intuitive interface allows increased accessibility of SciWorCS. We used *GoJS* [67] -a JavaScript library- for implementing interactive diagrams in HTML. We

used JavaScript Ajax for asynchronous server communications for obtaining different information from the server, such as availability of a dataset in the server, job status for a workflow execution and so on.

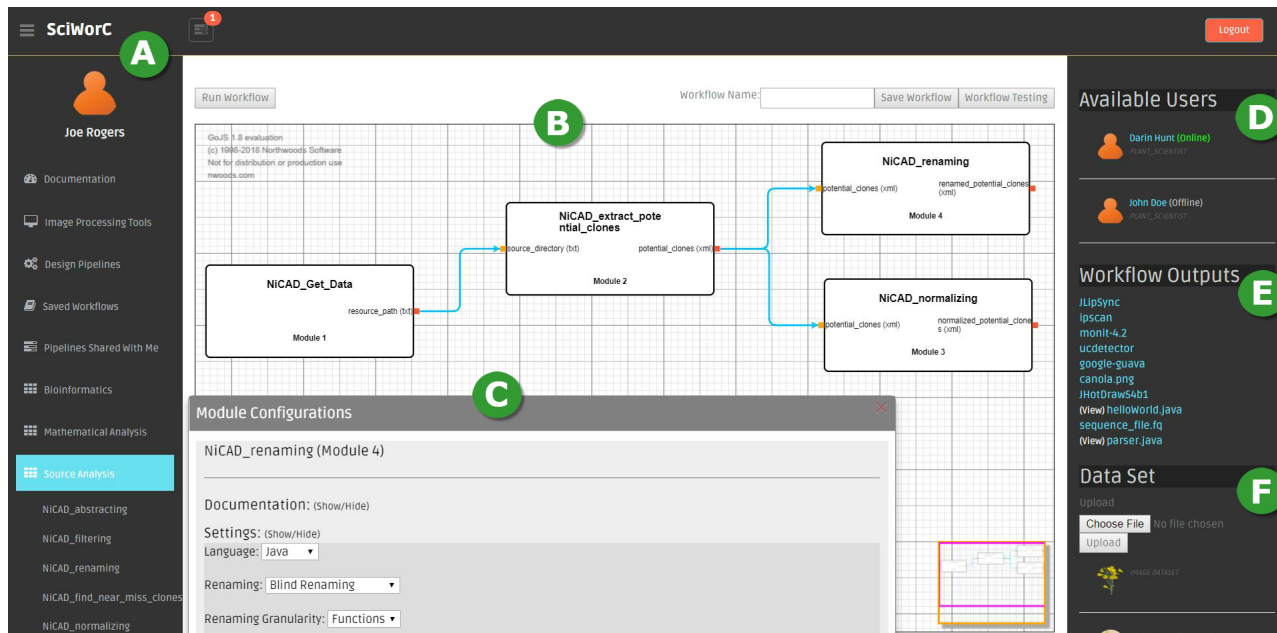


Figure 5.3: User Interface Overview of SciWorCS.

5.5 SciWorCS Task Specific User Interfaces

Fig. 5.3 demonstrates an overview of the SciWorCS user interface. The panel labeled as- ‘A’, contains all the workflow components such as, multiple toolboxes categorized as per the general data analysis tasks of the modular tools, saved workflows and shared workflows with collaborators. The composition of the workflow is done on panel ‘B’. The selected modular tools from the toolbox appear in this panel where they can be connected defining the dataflow relation among them. As illustrated in the figure, the workflow data flow relation is presented in intuitive DAG representation. The modules can be configured using the corresponding attributes in the popup panel ‘C’, -which appears on *Mouse Left Double-Click* on any module from the workflow DAG. Panel ‘D’ shows a list of collaborators and their current online/offline status. The list of the workflow outputs are presented in panel ‘E’ and the new dataset can be browsed and uploaded to the server for analysis from the panel ‘F’.

5.5.1 Plugging in New Tools to SciWorCS Toolbox

As presented in Section 5.3.1, SciWorCS architecture provides a plugin based framework for integrating new computational modules or tools to the toolbox. The integrated tools are independent and easily portable discrete unit of computation which can be shared among collaborators and used in different steps towards composing the collaborative workflow as per the requirements. SciWorCS is currently implemented in Python,

so any modular computational unit written in Python can be integrated as a tool to it. Besides, any modular software packages that can be run from command line, can also be integrated to the SciWorCS toolbox making a Python wrapper for the corresponding software package. An abstract tool definition file —comprising the reference variable for input and output binding with the modular source, allowed set of configurations for the execution tuning, and documentation is plugged into the SciWorCS along with corresponding source code of the software package. SciWorCS automatically generates and renders a simple and intuitive interface from the tool definition. The abstract interface layers enable easier portability and serving the tools among collaborators of the group. In addition to tuning the tool from the abstract-level configurations, SciWorCS allow more granular tuning from the source as per the requirements of the specific data analysis task. However, such granular access to the tool source is controlled by the user role as discussed in Chapter 4.

Listing 5.4 shows an example abstract tool definition for the corresponding modular source as shown in Listing 5.3. The Listing 5.3 demonstrate a sample implementation of *Support Vector Machine (SVM)* Classifier on top of Scikit-Learn [143]—a python based machine learning library. Lines 1-7 imports required Python packages and modules for the tool, Lines 10-15 loads the input dataset, Lines 18-22 are responsible for setting up the classifier with set of configurations (i.e., process logic) and finally Lines 25-29 writes the output (i.e., classification accuracy in this case) to the corresponding reference file. Note that, while this is a simple example representing a high-level template of a modular source, the implementation of SciWorCS modules can often be far more complex as per the addressed task of the module, such as comprising multiple input/output data sources, supported configurations and so on.

Listing 5.4 demonstrates the corresponding abstract module definition file. The XML definition file is enclosed inside a root tag—*SciWorCS*. Lines 2-8 lists the set of input datasets for the tool inside the tag—*toolInputs*. Though for this particular example, the tool takes only a single input dataset, in cases—there can have multiple input dataset references in SciWorCS tools. The output references also maintain a similar structure as demonstrated in Lines 10-16. The corresponding reference variables (e.g., Line 5 and Line 13 in Listing 5.4) are linked with the tool source (e.g., Line 12 and Line 27 respectively in Listing 5.3). The labels for the input and output dataset references are used for generating and rendering corresponding interfaces of the tools. The defined data formats are used for discovering the available data source from the repository and data link compatibility while connecting multiple tools composing SciWorCS workflows. The tool configuration as depicted in Lines 18-30 in the definition file is also referred and bound with the source codes of the tool. Finally, the tool documentation (e.g., Lines 32-34 in Listing 5.4) contains the information about the tool and its usage instructions for others in the collaborative group. Fig. 5.4, illustrates the SciWorCS interface for plugging-in the tool (e.g., comprising module source and definition file) to the toolbox. The integrated tools appear in the toolbox—which can be used as discrete computational unit or as steps towards workflow composition (e.g., as presented in Section 5.3.2).

```

1 ##### Required Imports #####
2 #Imports Required
3 import numpy as np

```

```

4 import pandas as pd
5 from sklearn.svm import SVC
6 from sklearn.model_selection import cross_val_score
7
8
9 ##### Reference Input Dataset #####
10 #loading reference dataset
11 dataset = pd.read_csv(csv_dataset_path)
12 #feature set and corresponding target variable
13 X = dataset[featureSet]
14 y = dataset[target]
15
16
17 ##### Classifier Configuration #####
18 #configuring the classifier
19 classifier = SVC(kernel = kernel_type , random_state = 0)
20 # Applying n-Fold Cross Validation for the classifier
21 accuracies = cross_val_score(estimator = classifier , X=X , y=y , cv = n)
22
23
24 ##### Write to Reference Dataset #####
25 #writing out some classification statistics to the module reference output
26 with open(SVM_classification_stats , "w+") as thisModuleOutput:
27     thisModuleOutput.write("SVM:\n = = >\n")
28     thisModuleOutput.write(" Classification Accuracy: " + str(round(accuracies.mean()*100,2))
29         + " %" )

```

Listing 5.3: Source Binding for the Definition in Listing 5.1.

```

1 <SciWorCS>
2   <toolInputs>
3     <toolInput>
4       <label>Dataset</label>
5       <referenceVariable>csv_dataset_path</referenceVariable>
6       <dataFormat>csv</dataFormat>
7     </toolInput>
8   </toolInputs>
9
10  <toolOutputs>
11    <toolOutput>
12      <label>Output Stats</label>
13      <referenceVariable>SVM_classification_stats</referenceVariable>
14      <dataFormat>txt</dataFormat>
15    </toolOutput>
16  </toolOutputs>
17

```

```

18 <toolConfigurations>
19     Feature Set for Training ('f_1 ', 'f_2 ', 'f_n '):
20     <input type="text" class="setting_param" value="featureSet=['f_1 ', 'f_2 ']' />
21
22     Target Variable (e.g., class label variable):
23     <input type="text" class="setting_param" value="target='c_label'" />
24
25     Kernel Type (e.g., to be used in the algorithm, such as 'rbf', 'sigmoid'):
26     <input type="text" class="setting_param" value="kernel_type='rbf'" />
27
28     n-fold cross validation:
29     <input type="text" class="setting_param" value="n=10" />
30 </toolConfigurations>
31
32 <toolDocumentation>
33     Support vector machines (SVMs) are a set of supervised learning methods used for
34     classification, regression and outliers detection. SVM is effective in high dimensional
35     spaces. (Documentation Source:: http://scikit-learn.org/stable/modules/svm.html).
36 </toolDocumentation>
37 </SciWorCS>

```

Listing 5.4: An Example of SciWorCS Module Definition.

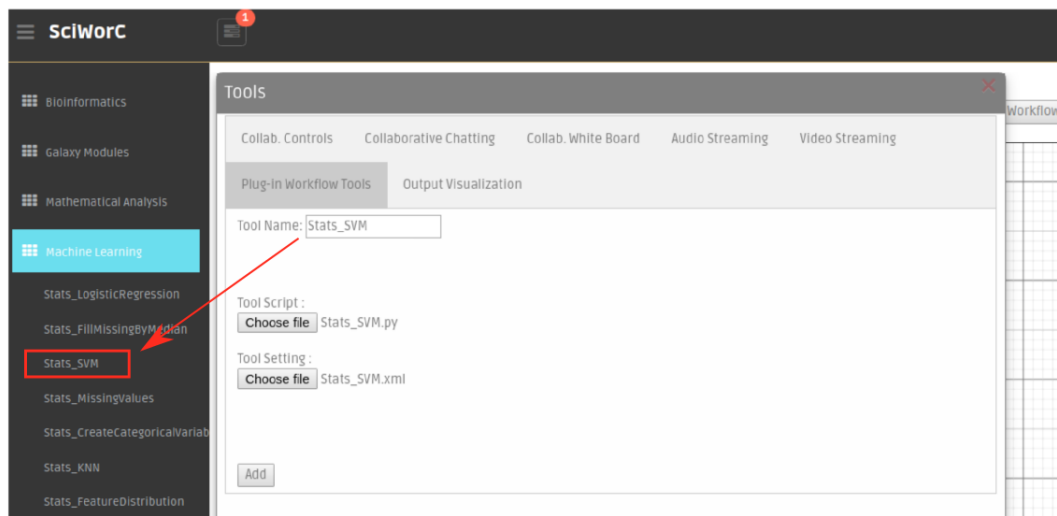


Figure 5.4: SciWorCS tool plugin interface.

5.5.2 Collaborative Workflow Composition

In addition to single user based workflow composition, SciWorCS provides a framework for collaborative workflow composition. Fig. 5.5 illustrates the collaborative workflow composition panel. We use our proposed attribute level locking scheme (e.g., as presented in Section 3) for facilitating the consistency management while collaborative workflow composition. As illustrated from the figure, the workflow modules are color-coded to represent the locking states of the workflow components. For example, the green colored modules—Module 3, 4 and 5 represent the accessible modules for one collaborator, while the red-colored sub-workflow comprising the Module 1 and 2 represents the locked workflow components by some other collaborator. On the other hand, Module 6 and 7 (e.g., no color) represent the currently unlocked workflow components in the collaboration process. The component lock can be requested on the components, as depicted on Module 6 for the sub-workflow lock request. The SciWorCS lock manager maintains the lock requests and serves the requested workflow components among the collaborators.

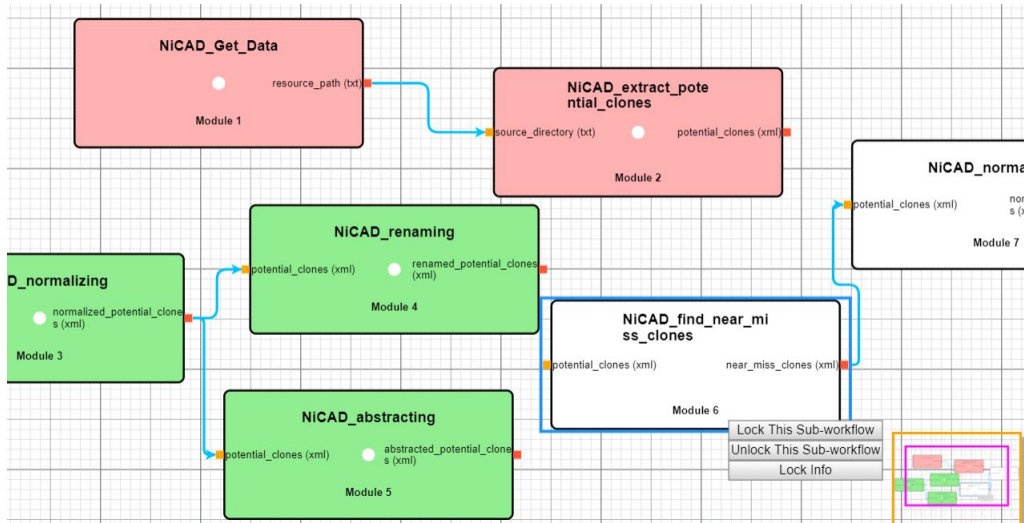


Figure 5.5: Locking Schemes for Consistency Management in Collaborative Composition (A subset of the workflow nodes with corresponding Lock states).

5.5.3 Tools for Aiding Collaborative Workflow Composition

While the consistency management is one of the primary requirements of a collaborative system [173, 172], providing different methodologies for group communication towards problem-solving and decision making are also often very important towards the success of the system [48]. We incorporate different communication tools for the group discussion and decision making in the process of collaborative data analysis. Fig. 5.6 illustrates the implemented SciWorCS communication tools. The textual communication tools include peer-to-peer and group chatting system. In addition to the textual communication tools, SciWorCS provide real-time Audio and Video streaming based communication system among the collaborators. As also illustrated

in the figure, a *Collaborative Virtual Whiteboard (CVW)* also has been implemented in SciWorCS for aiding the group discussion and problem-solving. The CVW contains different simple tools (such as color selector, paintbrush size selector and so on) to manage the discussion process among the collaborators. In real-time groupware systems, *telepointers* (e.g., multiple cursors) are widely used to increase the group awareness [73]. Studies show that, in the context of CSCW, through the simple movement of telepointers, collaborators often communicate their focus of attention, gesture over the shared views and so on [70, 73]. For example, in terms of collaborative SWfMSs, telepointers often might be used by the collaborators to create group awareness about the individuals' focus on attention on sub-workflows, tools and so on. Hence, we also implement real-time telepointer communication systems among the collaborators in SciWorCS.

SciWorCS leverages *WebRTC* for the implementation of the communication tools. WebRTC provides real-time peer-to-peer communication along with audio-video streaming from modern browsers without any further requirements of software packages or tools [112]. Note that, while we have implemented different communication systems for the collaboration, their selection or usage patterns among collaborators might often be impacted by several factors, such as the collaborating group itself, the nature of data analysis task for collaboration and so on. We conducted several user-studies for investigating the usage patterns of the communication tools or adapted styles of work in terms of collaborative data analysis. We present our findings from our empirical studies in Chapter 6.

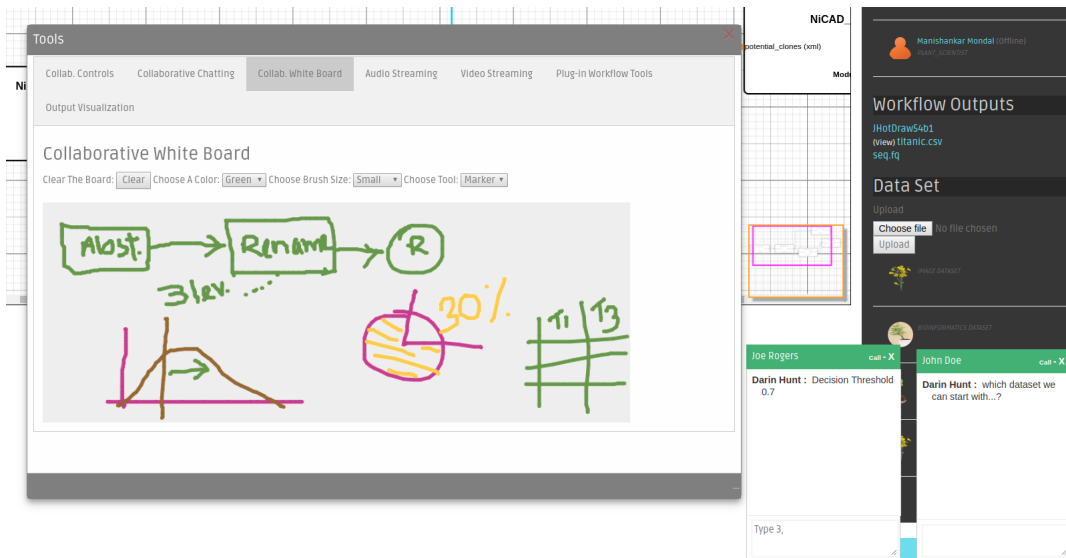


Figure 5.6: Example collaborative tools for aiding collaborative data analysis process.

5.5.4 Data Visualization

Fig. 5.7 demonstrates the SciWorCS data visualization framework. The visualization framework is important for aiding the collaborative data analysis process. The html report, textual (e.g., XML, text, json and so on) and image (e.g., png and jpg)—input or generated output dataset are rendered in the SciWorCS framework for

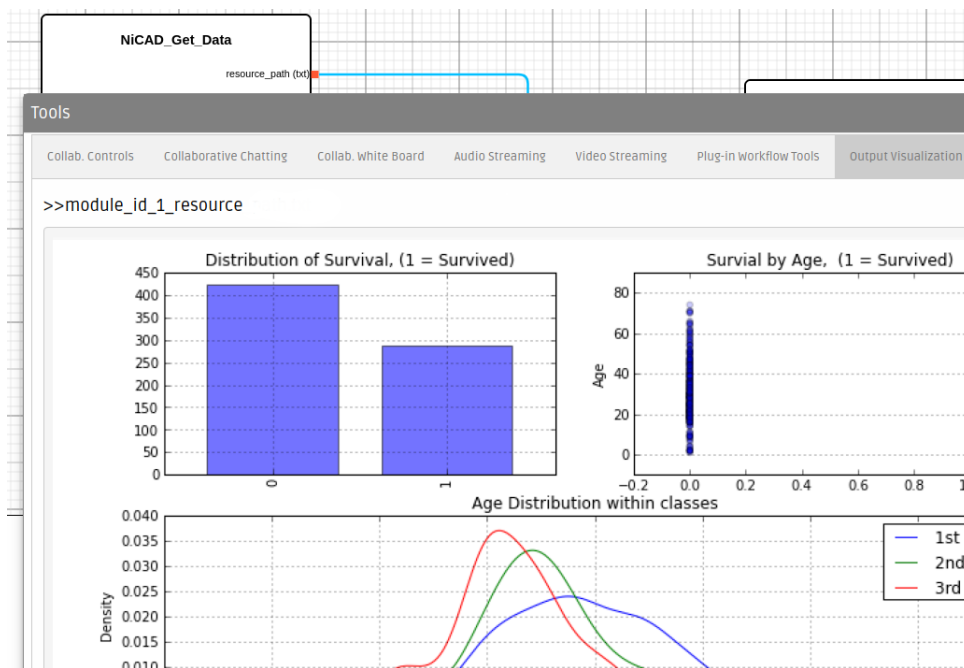


Figure 5.7: Dataset Feature Distribution - An Example Visualization for the Classification.

visualization. Fig. 5.7 illustrates an example visualization of a dataset for feature distribution and statistical analysis. Using SciWorCS, diverse number of visualization tools can be plugged into the toolbox. The visualization tools works on top of some reference input datasets and generates the corresponding visualization output. The generated visualization outputs in supported data format are then visualized in the framework.

5.6 SciWorCS Usage Examples

In this section we present some usage examples of data analysis using SciWorCS.

5.6.1 QC Report of FastQ File with FastQC (Bioinformatics)

Different *Quality Control (QC)* tools are widely used in the field of Bioinformatics to investigate any potential problem or check the quality of the input sequence files [25]. Such quality analysis tools are important to ensure that there are no hidden problems on the sequence files—which might be difficult to detect and recover from- at some later stages of the analysis of the sequence file [54].

FastQ is a popular sequence file containing *Nucleotide* sequence data with associated quality scores [25, 54]. *FastQC* is a widely-used tool for QC report generation and analysis of FastQ sequence files [54]. In addition to the FastQ sequence files, the tool also accepts inputs as *Sequence Alignment Map (SAM)* or *Binary Alignment Map (BAM)* formats—runs a series of tests and generates corresponding QC reports [54]. FastQC also generates tables, graphs and HTML based permanent report for overall visualization of the QC reports.

With this example, we present two use cases of SciWorCS— i) plugging-in tools written in languages other than Python (e.g., for the claim in Section 5.5.1) and ii) plugging-in different visualization tools with the SciWorCS visualization framework (e.g., for the claim in Section 5.5.4). While SciWorCS is implemented in Python, the modular FastQC tool is written in Java language. However, like any other Python modular tools, FastQC is integrated into SciWorCS by writing a Python wrapper. Listing 5.5 presents an overview of the written Python wrapper script for the FastQC. Line 8 uses the input and output references in the Python sub-process for the FastQC execution—which is referred accordingly in the SciWorCS tool definition file.

The plugged-in FastQC tool can be used in SciWorCS workflows. Fig. 5.5 illustrates a sample workflow comprising the integrated FastQC. The figure also illustrates the visualization of the generated report from FastQC. Note that, as FastQC, other visualization tools can be plugged-in to SciWorCS—where the generated outputs are used by the framework for visualization. We present more examples and use-cases on it in Chapter 6.

```

1 ## Required Imports
2 import subprocess
3 import os
4
5 ## Removed extraneous details ...
6
7 ## FastQC Wrapper on top of Java
8 pipe = subprocess.Popen(["perl", "FastQC/fastqc", ref_input_fastq, "--outdir",
    ref_output_fastqc_report , "--extract"]).communicate()
9
10 ## Removed extraneous details ...

```

Listing 5.5: FastQC Python Wrapper Overview.

5.6.2 Machine Learning Based Clone Validation (Software Repository Analysis)

Background and Motivation: Copying and reusing certain pieces of existing code directly or with alteration into another location is a common programming practice in a software development lifecycle [155]. Such copy/paste practice results in similar pieces of code fragments in a system, —called *Code Clones* [156, 155, 98, 94, 176]. Researchers agree upon four primary clone types [155, 125, 176, 157] : Type 1 clones are syntactically identical code fragments, regardless of the presentation style, comments, and white spaces. Type 2 clones are copy and pasted code where identifier names and types have been changed. Type 3 clones are modified copies of the original code with statement-level changes (e.g., additions of new statements, or deletions and modifications of existing ones). Syntactically dissimilar code fragments that implement the same or similar functionality are termed as Type 4 clones. Some of the recent research shows that on average around 7% to 23% of the total code of a software system is duplicated or cloned from one location to another [156], [11], [99]. Though code cloning is often done intentionally to accelerate the development process and also not all code clones are harmful [98], the existence of some of them can inflate software maintenance costs

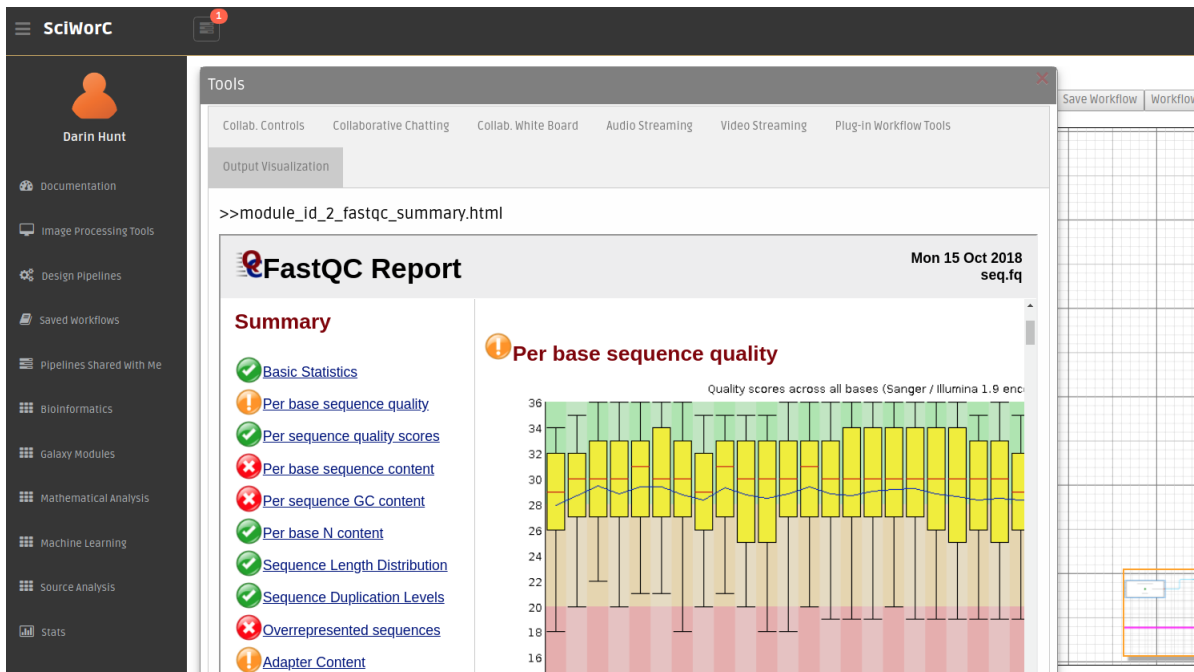


Figure 5.8: FastQC quality measures on example sequence file.

as clones are one of the major causes of creation and propagation of software bugs throughout the system [94], [64], [16]. For example, it becomes very difficult to carry out a consistent change to all the cloned code fragments throughout the software system. This inconsistent changes to the corresponding duplicated code fragments are often responsible for the creation of new software bugs [127], [128]. In addition to the creation of new bugs code cloning also becomes one of the main reasons for bug propagation when programmers copy-and-paste a buggy code fragment throughout the software system for implementing similar functionalities [126], [94]. Detection of such code clones can therefore accelerate the maintenance task of any software systems remarkably [94]. Besides, exploiting the similarities of the detected code clones also help one better understand and improve the overall software design [81].

At least 70 Clone Detection Tools and techniques have been proposed and developed to automate the clone detection process, as a result of extensive research in this specific area over the last decade [96], [10], [50], [15], [157], [88], [177]. These tools return a list of possible code clone pairs or classes available in a given software system. Except Type 1, the other types of code clones (Type 2, 3 and 4) undergo different changes over time and can get too complicated to be detected with a simple string matching algorithm by a clone detection tool. For example, the identifiers or functions names may be changed, some code fragments may be added, modified or removed, a portion of the code clones might undergo several other syntactical changes or even the complete implementations might be changed for the same functionalities in any other locations [157, 96]. All these modifications over time make the searching problem much more complicated [157]. In order to handle those complex source code structures while still detecting all possible code clone pairs, the tools undergo a lot of generalization of the original source codes like pretty-printing [157], normalization of

the identifiers [96], [157], forming syntax tree [104] of the code fragments and so on just to name a few.

As a result of this complex searching problem and necessary generalization or normalization of the source code, the clone detection algorithms often report false positive clones [90, 197]. These are pairs of code fragments that are not similar or possibly are only coincidentally similar or are otherwise considered not a valid clone by the user [90]. Besides, some research shows that the definition of true positive code clones especially in case of Type 3 and Type 4 clones are subjective and might also be different for different users or software systems [102], [34], [197]. For example, Yang et. al. [197] conducted a survey where several users were provided the same clone sets for validation detected by clone detection tools. The study reported significant variations among the users in validating the same clone sets (e.g. for the same provided clone sets, the number of decided true positive code clones varied within a range of 4.76% to 23.81% for different users). For these reasons programmers often need to manually validate if the results of a clone detection are a true clone or not before using these information for the given specific scenarios like: source code refactoring or other software maintenance tasks [197]. Such a manual validation process often becomes a hindering factor even for medium-size software system [197]. Because in that case programmers often find it challenging to extract the actual true positive clones they are looking for from those large set of reported possible code clone pairs by clone detection tools. For example, some previous research shows that JDK 1.4.2 contains 204 K LoC reported code clone which is 8% of the total lines of code [90], [178]. 15% of the total lines of code of the Linux kernel has been reported as code clone which is 122 K LoC [108]. Both of the above scenarios on the number of reported possible code clones by clone detection tools illustrate the huge amount of manual validation work necessary before using the code clone information. Besides the clone detection algorithms of the tools usually work in general irrespective of the specific system requirements or user preferences. Thus, in the best case, even if a tool returns only true positive clones, many of those clones might not be relevant to the tasks at hand of the programmers or engineers (e.g., not suitable for refactoring) [90]. Mining those code clones of interests from the tool generated report is often a time consuming task and thus reduce the usability of code clone detection tools. The scenario even gets worse with the increase of software project in size [197].

Here, we present a machine learning based approach or workflow for automating the code clone validation problem¹ using SciWorCS. The proposed method works on top of any code clone detection tools for classifying the reported clones as per user preferences. The automatic validation process for a user, thus can accelerate the overall process of code clone management and helps faster acquiring of required information out of the clones in comparison to the time-consuming manual validation process. We studied performance and result qualities of different machine learning algorithms in validating the detected clones. We got promising results from our several studies with different experimental setups for the clone validation.

¹The presented workflow for clone validation was published in IEEE 18th International Working Conference on Source Code Analysis and Manipulation, 2018. We refer the reader to our original paper for details: *Mostaen, Golam, et al. 'On the Use of Machine Learning Techniques Towards the Design of Cloud Based Automatic Code Clone Validation Tools.' 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2018.*

Here in this study, we aim to answer the following two research questions using SciWorCS:

- **RQ 1:** Can the manual code clone validation process be assisted via machine learning?
- **RQ 2:** Does the proposed machine learning based validation method work across different clone types and clone detection tools?

We study and investigate two main factors of machine learning based clone classification using SciWorCS. First, we study the data distribution for the clone classification problem with several extracted features with SciWorCS workflows. Our findings on these feature sets and data distribution analysis can help better understand the clone classification problem and thus adds the possibility of result improvement in this research area. Second, we conduct a comparative study with different machine learning algorithms for the clone classification, —where the classification algorithms are different machine learning based SciWorCS modules.

5.6.3 Machine Learning Based Clone Validation Approach

In this section, we discuss the SciWorCS workflow for the clone classification problem. The proposed method uses machine learning models for predicting the user-specific code clone validation. The models are first trained based on manually validated code clone sets from the corresponding users. The trained models are then used for improving the reported code clones from clone detection tools, by predicting the user-specific validation patterns.

Code Clone Validation Workflow

Figure 5.9, shows a high level workflow of the proposed method. The workflow steps can be listed in sequence as the following:

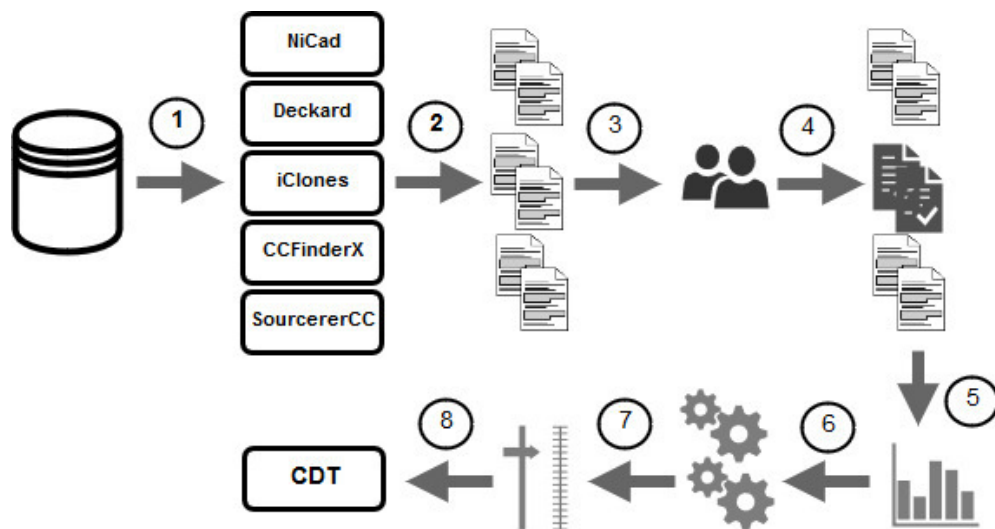


Figure 5.9: High-level Workflow for Machine Learning based Code Clone Validation

1. In Step 1, source codes from Code base are supplied to any of the existing clone detection tools.
2. The detected code clones from corresponding clone detection tools are collected in Step 2. As the proposed method works with any of the existing clone detection tools, clones from multiple tools can be combined optionally for further generality of the training set.
3. In Step 3, the reported code clones from clone detection tools are provided to the user for manual validation.
4. User marks the code clones as true positive or false positive in Step 4. The user-specific manual clone validation results are stored in a database for using as the training set of machine learning models.
5. In Step 5, several features are extracted from the marked code clone pairs for training the machine learning models. The existing related research works (i.e., FICA [197]), used only simple token sequences as features for training the machine learning model and thus failed to predict the validation successfully beyond Type 2 clones. To improve the classification results and to target clones beyond Type 2, we considered calculating clone similarity with several levels of structural pre-processing and normalization.
6. The extracted features are used to build the feature vector for clone classification. Feature vectors for the corresponding manual validation clone classes are used for training machine learning model in Step 6.
7. Next, in Step 7, the trained machine learning model is used for predicting the clone validation pattern for the unknown or test sets. The machine learning model at this stage returns probability score (of being true or false positive) for the given corresponding code clone pairs.
8. Finally, in Step 8, the classified result is sent back to the clone detection tools. The classification result can be tuned based on user preference (of probability score) for the final result. The system can take user feedback based on the classified clones from user for repeating the cycle of supervised learning, thus improving the validation result over time and experiences.

Manual Code Clone Validation for Training

The reported code clones from the clone detection tools are provided to the user for manual validation (Step 3, Figure 5.9). The corresponding user validation results are stored in database which is later used for training the machine learning model. Reported clones from clone detection tools are used to create a clone database, K . Clones from K , are manually marked as true or false positive by the user. Reported code clones are thus grouped into two disjoint sets K_t and K_f - representing true positive and false positive clone groups respectively, such that, $K = K_t \cup K_f$ and $K_f \cap K_t = \emptyset$. K_t and K_f are used for training the machine learning models in a later stage of the clone validation workflow.

Feature Extraction

As machine learning models learn to map the input feature sets to the corresponding class label, it is important to select appropriate features for the given classification problem. For example, Yang et al. [197] targeted Type 2 clones in a similar study of code clone classification problem and hence used simple token sequences as features for using in the classification algorithm. As we intended to enhance clone the classification performance (so that it works efficiently with more diverse types of clones and also shows better prediction score), in addition to improving the whole classification workflow, we also focused on extracting more informative features. Most of the selected feature extraction undergoes two main steps: i) Pre-processing and source code transformation and ii) Similarity extraction from the code clone pairs.

Pre-processing like pretty-printing, comment removal and so on ensures consistent structures for matching and similarity extraction of similar source code pairs (for example Type 1 clones) [157, 40]. Extracting similarity features between the two code clone fragments after this step (i.e., comment removal followed by pretty-printing) gives us the information about how a user sees the code clones for validation. Thus at this point, the extracted features represent mainly Type 1 similarity between the target code clone fragments. In addition to that, different source transformations like consistent normalization of literals, consistent renaming of identifiers and so on are applied to consider the possible changes between the code clone pairs (i.e., for Type 2 and Type 3 clones [157]). For example, Listing 5.6 and Listing 5.7, show the code fragments of one of the detected clones from Weka [192] software system, that needs to be validated.

```
1 try {
2     if (args.length == 0) {
3         throw new Exception(
4             "The first argument must be the class name of a kernel");
5     }
6     String associator = args[0];
7     args[0] = ">";
8     System.out.println(evaluate(associator, args));
9 }
```

Listing 5.6: Sample Code Clone (Fragment 1)

```
1 try {
2     if (args.length == 0) {
3         throw new Exception(
4             "The first argument must be the name of a "
5             + "clusterer");
6     }
7     args[0] = "?";
8     Clusterer newClusterer = AbstractClusterer.forName(ClustererString, null); //object from
9     System.out.println(evaluateClusterer(newClusterer, args));
```

```
10 }
```

Listing 5.7: Sample Code Clone (Fragment 2)

Although the code clone pair exhibit much structural similarity, calculating similarity directly based on original source code pairs have a higher probability of introducing noise (from the perspective of Type 2 or Type 3 clones) due to strict consideration of the modifications of literals and identifiers. So, we also applied different source code transformations before calculating clone similarity for extracting possible Type 2 or Type 3 information.

```
1 try {
2     if (X.X == 0) {
3         throw new X(
4             "string");
5     }
6     X X = X[0];
7     X[0] = "string";
8     X.X.X(X(X, X));
9 }
```

Listing 5.8: Pre-processed and Transformed Code Clone (Fragment 1)

```
1 try {
2     if (X.X == 0) {
3         throw new X(
4             "string"
5             + "string");
6     }
7     X[0] = "string";
8     X X = X.X(X, null);
9     X.X.X(X(X, X));
10 }
```

Listing 5.9: Pre-processed and Transformed Code Clone (Fragment 2)

For example, Listing 5.8 and Listing 5.9, show the transformed clone fragments from Listing 5.6 and Listing 5.7 respectively, after first blind renaming of identifiers and then applying consistent normalization of literals. For example, after blind renaming of the identifiers, all different identifiers takes a common name— X for the structural comparison. Similarly, all the different string literals were transformed to — ‘string’ after consistent normalization of the literals as shown in Listing 5.6 and Listing 5.7. These transformations allow the corresponding modifications of literals and identifiers and thus provides similarity feature information for Type 2 and Type 3 code clones.

After applying different pre-processing and transformation for different types of features, we then analyze the differences between the code clone fragments (i.e., output of the previous steps). In prior to calculating numerical similarity values between the clone fragments, we find out the minimal changes required to

transform from one clone fragment to another.

```
1 4c4,5
2 <      "string");
3 ---
4 >  "string"
5 >  + "string");
6 6d6
7 <      X X = X[0];
8 7a8
9 >      X X = X.X(X, null);
10 10,12d10
11 <
12 <
13 <
```

Listing 5.10: Difference between the code clone fragments

For example, Listing 5.10 shows the minimal changes or operations required for transforming code clone fragment 1 (i.e., Listing 5.8) to code clone fragment 2 (i.e. Listing 5.9). We use *Unix Diff* utility for the purpose, that calculates the minimum set of insert and delete operations required for converting one file to another. For example, in Listing 5.10, < and > signs represent delete operation, \mathcal{O}_d and insert operation \mathcal{O}_i respectively, that we need to apply on first clone fragment for the required transformation. For example, the first conflicts of the transformed clone fragments in Listing 5.8 with Listing 5.9 is at line 4. The minimum operations needed to resolve the difference is one delete operation, \mathcal{O}_d of original line at 4, followed by two insert operations, \mathcal{O}_i of line 4 and 5 from Listing 5.9. The corresponding change operations has been represented as 4c4,5 in Listing 5.10. We then calculate the similarity value between the two code clone fragments f_1 and f_2 as, $\xi(f_1, f_2) = 1 - \max(C(\mathcal{O}_d)/|f_1|, C(\mathcal{O}_i)/|f_2|)$, where $C(\mathcal{O})$ and $|f|$, represent the count of the corresponding change operation and length of the corresponding code clone fragment respectively. The fragment similarity thus falls in the range of $[0,1]$. As the number of such differences between the two code clone fragments increases, the code clone fragments similarity measure tends towards zero. On the other hand, the fragment similarity is calculated as 1, in case the clone fragments are exactly similar with no further required changes (i.e. $C(\mathcal{O}_d) = C(\mathcal{O}_i) = 0$).

We also used several other features to get more structural information about the two code clone fragments. We tried to mimic several manual validation patterns as per our obtained experiences on manual code clone validation of users. For example, our intuition was, if the code clone fragments are significantly different in sizes, validator may be more likely to mark them as false positive. The corresponding code clone fragment sizes α and β , were calculated as respectively. The difference $|\alpha - \beta|$, provides the information about the variation of fragment sizes. The smaller difference value represents more likelihood of being validated as true positive code clone than that of comparatively higher difference values and thus was considered as one possible feature for the clone classification problem. However, for a clone that is small versus a clone that is

large might have different consideration. For example, for a relatively larger code clone fragment pair, it is possible to have more variance in difference than that of smaller code clone fragments pair. So, to mitigate this possible bias we also considered the average size of the code clones $(\alpha + \beta)/2$. That average value captures sort of the size of the clones and difference captures if the code fragments are rather mismatched in size.

Note that some of the popular code clone detection tools use source transformations like the consistent renaming of identifiers, normalization of literals and so on, as part of their workflow for code clone detection [157, 96]. A few of such clone detection tools like NiCAD [157], and CCFinder [96] are also well known in the research area for their better performance on clone detection. So, with this motivation, for a subset of the features, we also carried out similar transformations before calculating the clone fragment similarities as discussed above. However, to the best of our knowledge, no previous works on clone validation used a similar feature set.

We use several re-usable SciWorCS modules to compose the required feature extraction workflows as discussed above. The discussed feature extraction workflows can be composed of different independent source preprocessing, transformation and numerical feature calculation modules. The preprocessing modules for the problem includes *pretty-printing* and *comment removal* modules, $M_{prettyPrinting}(inputSource)$ and $M_{commentRemoval}(inputSource)$ respectively of the input source codes. For an input source code fragment, the following workflow, W_p can be used for the required preprocessing of the source codes for the discussed clone validation approach.

$$W_p : inputSource \rightarrow out1 = M_{prettyPrinting}(inputSource) \rightarrow out2 = M_{commentRemoval}(out1)$$

We use TXL [39] for writing the SciWorCS modules for these required source code preprocessing. Similarly, we identify and write SciWorCS modules for source code fragment transformation and numeric feature calculation — $M_{sourceRenaming}(inputSource)$, $M_{sourceNormalization}(inputSource)$, $M_{sourceAbstracting}(inputSource)$, $M_{sourceFiltering}(inputSource)$ and $M_{sourceDiff}(inputSource1, inputSource2)$ as discussed above. Note that, some of the modules can be tuned with a different set of configurations [157, 39], which can be re-used in different settings in the SciWorCS workflows to extract features of different levels (e.g., functions or block level granularity in abstracting) for the clone validation problem. Fig. 5.10 illustrates a sample SciWorCS workflow for clone validation feature extraction as presented above. The figure also shows a subset of extracted features from a manually validated dataset (e.g., as presented in following *Data Source* Section) joined together for the usage of machine learning classifier training phase.

Studying Data Distribution using SciWorCS

As the machine learning algorithm tries to recognize any available underlying pattern in the given dataset, it is important how we choose the dataset and which features we extract out of it for training and testing of the system [132]. For example, selecting a smaller or undiversified dataset can make the algorithm biased, resulting in the failure to generalize all the other different types of clones [132]. So to get generalization

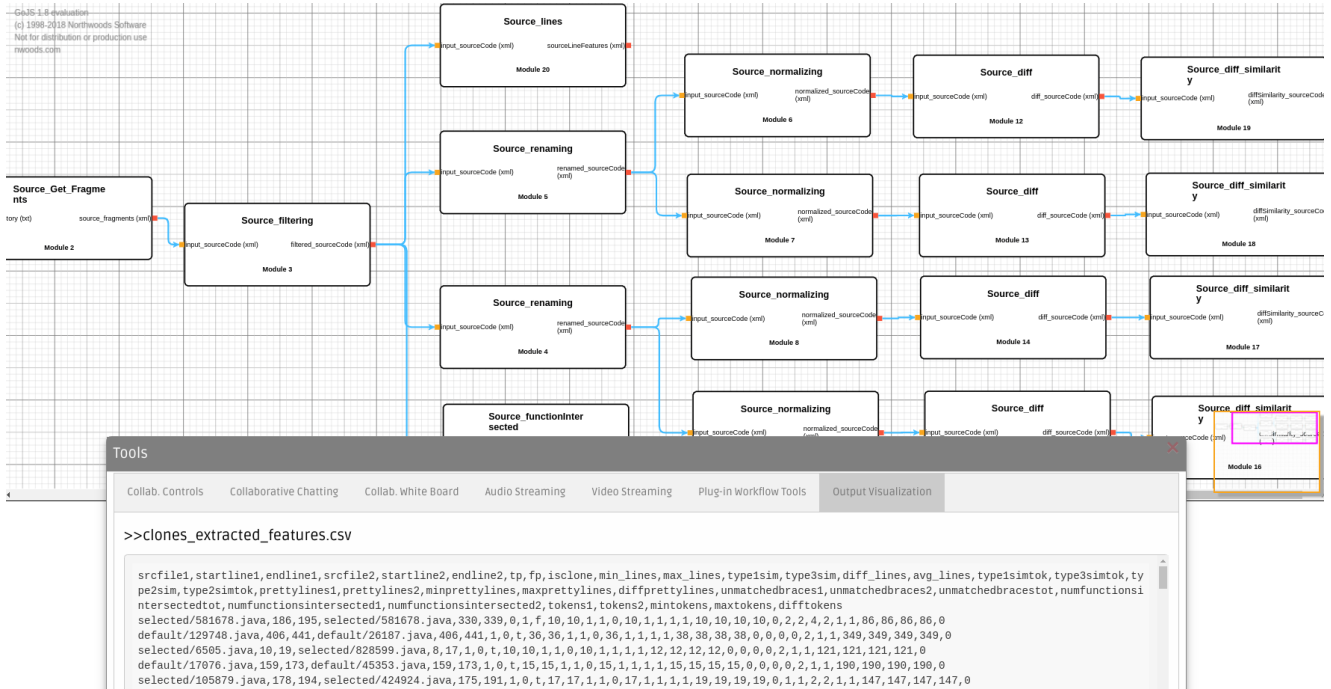


Figure 5.10: Sample feature extraction workflow for machine learning based code clone validation.

in validating different types of code clones by the system we have chosen to use relatively bigger and diverse dataset of open source projects. Besides we also considered clones reported by different existing clone detection tools from multiple open source projects.

Data Source

For training, we used clones from IJaDataset 2.0 [71], — a large inter-project dataset of open-source Java software systems. To test the generality of the proposed method, five different publicly available and state-of-the-art tools namely NiCad [40], Deckard [88], iClones [63], CCFinderX [96] and SourcererCC [158] were used to detect clones separately out of the benchmark. Randomly 400 clone pairs were then selected and manually validated from each of the five clone detection tools separately. We have chosen to work on this dataset because of a good number of recent research works on code clones have been carried out on these open source projects and thus we can have a common ground for evaluating the proposed approach.

High Level Details of the Data Set

Reports obtained from any of the existing clone detection tools on possible code clone pairs are given as input to the proposed SciWorCS workflow for clone validation. Several code clone detection tools were run on the used data source to find the corresponding reports for the possible code clone pairs. The code clone pairs were then manually validated for the training phase of the proposed method. As some recent research shows that the clone validation decision in some scenario depends on users perspective [102], that is given a

possible code clone pair to validate some judges might decide it to be a true positive clone pairs where others might say the opposite (especially, in case of Type 3 and Type 4 clones). So to consider this generalization to the proposed method the whole set of code pairs were split into 5 parts to be validated by 5 different programmers independently. This manual validation decision along with the corresponding possible code clone pairs are given as input to the proposed method for the training phase.

Analyzing Data Distribution for the Clone Classification Problem

Out of those manually validated clones we extracted different features that are used to train the machine learning model. In this section, we discuss different distribution and statistical studies and behaviors of some of the extracted features using SciWorCS workflow.

For every code clone pairs detected by clone detection tools, we found the similar code fragments for a clone pair. These are the similar code fragments for which the tools decided them to be possible code clone pair.

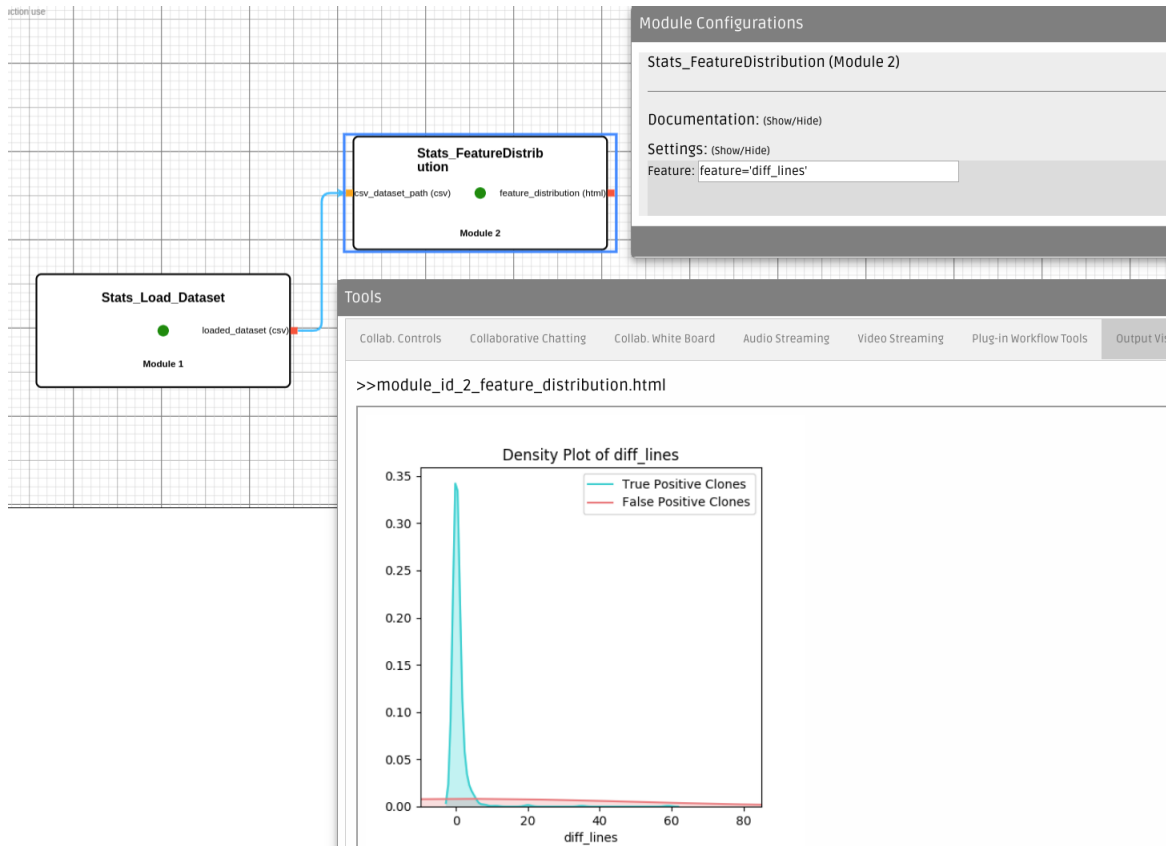


Figure 5.11: Histogram of Code Fragment Line Differences.

We analyzed the feature of the code clone pairs for both true positive and false positive manually validated clones in an attempt to find the contribution score for clone classification. For the extracted feature vector $\mathbf{x} \in \mathbb{R}^n$, a SciWorCS module plots the numerical feature $x_i \in \mathbf{x}$ for the two class labels— true positive

vs. false positive manually validated clone. Hence, a general structure for such data distribution analysis SciWorCS modules is — $M_{dataDistribution}(labeledDataset, feature, targetVar)$.

Figure 5.11, shows the distribution of the average code clone fragment feature $((\alpha + \beta)/2)$, as discussed in Section 5.6.3) for the true positive and false positive clone classes. From the figure, we can notice that the average code fragment size shows much randomness, both for true positive and false positive clones. The distribution of this feature almost overlaps on one another for the two classes: true positive and false positive code clones. This overlapping pattern suggests that this feature provides very minimal information about the two classes and thus yields a very low possible contribution score for training the machine learning algorithm for validation.

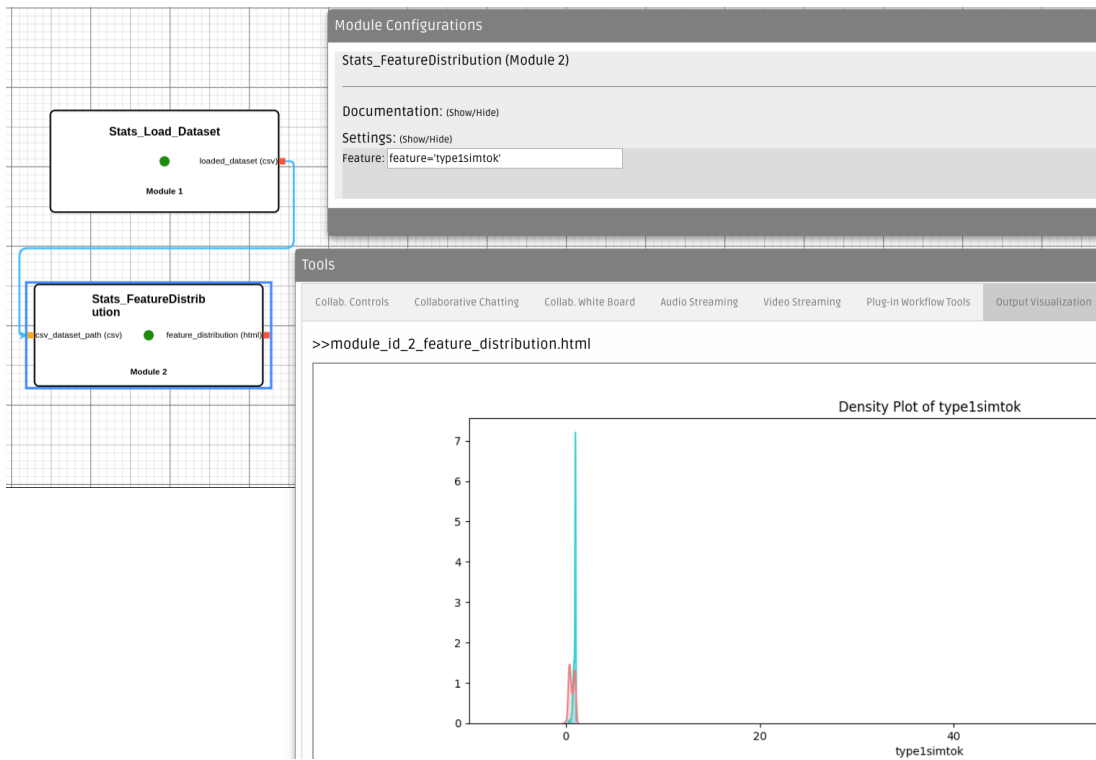


Figure 5.12: Histogram of Syntactical Similarity by Token (Type 1 Norm.)

Besides for extracting some other features we normalized the code clone pairs by 3 levels, namely: Type 1, Type 2 and Type 3 — using the SciWorCS modules presented above for source fragment transformation and normalization. Then for each level of normalizations syntactical similarity was measured by lines and by tokens for the clone pairs resulting 6 different possible features (Section 5.6.3). To view any underlying distribution of the features their normalized histogram was plotted both for true positive and false positive clones. Figure 5.12, shows one of such plottings that is based on the similarity measured by token after Type 1 code normalization. From the figure, it is noticeable that the distribution of the feature is comparatively better than the average code fragment line feature in terms of validation. Though the distribution for true

positive and false positive clones are not completely linearly separable with this feature but still the two classes are somewhat distinguishable. The distribution indicates a possible better contribution score for validation prediction than the average clone fragment sizes.

We also use SciWorCS workflow to carry out several studies to find out any underlying relationships between different features for possible clustering of the two clone classes. For example, we tried to figure out if there is any underlying relationship available for different types of similarity measures that can give any potential information about the clustering of the two clone classes. We plotted our several study result for visualization in an attempt to notice any distinguishable separation or clusters. However these analyses did not show any distinguishable cluster information for the two classes.

As machine learning algorithms try to recognize any underlying pattern available on the working dataset, the detail analysis on the dataset and possible features are necessary for selecting the features and machine learning algorithm. This distribution analysis of different possible features for code clones provide information about their importance and contribution for clone validation. This analysis provides a clearer view of the data distribution and thus helps to pick the appropriate machine learning algorithm and corresponding features for the algorithm. From several analyses on the data distribution we tried to find out the features that have comparatively more distinguishable distribution and provides more contribution for the two classes true positive and false positive clones. Table 5.1, shows a feature set ranked on possible contribution score based on our analysis study. The corresponding distribution mean differences, $\Delta\mu$ for the two classes also somewhat indicate the separability for the classification.

The detailed feature study in terms of class distribution prior to applying any machine learning algorithm is very important, since using any noisy feature (for the specific classification problem) may affect the classification performance and reduces the generality of the classification. Our distribution study, thus also contributes to the research area for further improvement in feature extraction and selection of appropriate classification algorithms. From our study, we built the feature vector as listed in Table 5.1. The other features were not used for the clone classification due to their low contribution scores or noisy behaviors for the classification as discussed above.

Training Machine Learning Models for Clone Classification

As we have presented the workflow of the proposed method in the above discussion, it uses a supervised machine learning algorithm for learning the classification pattern of the user-specific clone validation (i.e. in Step 6). The supervised classification algorithm will be trained on the manually validated dataset, $D = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2) \dots (\mathbf{x}_m, \mathbf{y}_m)\}$, for $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in \mathbb{R}^l$, where n and l represent the extracted clone feature set and clone validation labels respectively. The machine learning algorithm is then trained on the dataset, D to learn a function f , such that f can map from \mathbb{R}^n to \mathbb{R}^l , representing the class probability for being true or false positive for the given pairs of code clones. Hence, the SciWorCS modules — $M_{cloneClassification}(D, featureSet, targetVariable)$ for clone classification problems learns the function f

Table 5.1: Selected Features Based on Distribution Analysis

Feature	$\Delta\mu$	Feature Summary (as discussed in details in Section 5.6.3)
Line Sim. (Type-1 Norm.)	0.3998	Syntactical similarity measured by line after Type-1 Normalization
Line Sim. (Type-2 Norm.)	0.3701	Syntactical similarity measured by line after Type-2 Normalization
Line Sim. (Type-3 Norm.)	0.3602	Syntactical similarity measured by line after Type-3 Normalization
Token Sim. (Type-2 Norm.)	0.3447	Syntactical similarity measured by Token after Type-2 Normalization
Token Sim. (Type-1 Norm.)	0.3105	Syntactical similarity measured by Token after Type-1 Normalization
Token Sim. (Type-3 Norm.)	0.2537	Syntactical similarity measured by Token after Type-3 Normalization
Function Intersected	0.2364	Total Number of functions intersected by the code fragments
Unmatched Braces	0.2078	Total number of unmatched braces across both code fragment

from the labeled dataset, D for predicting the clone validation patterns. In addition to the classifier specific configurations (e.g., number of hidden layers for ANN and so on), the machine learning modules, in general, are configured for the feature set and target labeled variable for classification of the dataset, D .

We use SciWorCS for the comparison study of machine learning models (e.g., which are wrapped as different SciWorCS modules) for the clone classification. We investigated the classification performance using different machine learning algorithms as to the best of our knowledge, we could not find any other previous research works that directly focused on user-specific clone validation using such extracted clone features to target validation of all 3 different types of clones. We studied the performance of multiple machine learning classification algorithms, for example, Random Forest, Naive Bayes Classifier, C48, Decision Table and Artificial Neural Network. While we refer the readers to our original paper [129] for details, here we show a part of the comparative study to demonstrate the use-case of SciWorCS.

Fig. 5.13 illustrates a SciWorCS workflow for code clone classification with different machine learning algorithms. All of the algorithms use the same dataset as we presented above. In addition to specifying the input data source, the classifiers are further tuned with corresponding classifier configurations. For example, the figure demonstrates a set configuration for ‘*Random Forest*’ machine learning classifier. On completion of any such configurations for other machine learning classifier modules, the SciWorCS workflow is submitted for execution. On receiving the workflow job submission, SciWorCS schedules and executes the classifiers to return the classification stats for the corresponding machine learning classifiers. The machine learning modules’ classification stats include accuracy, precision, and recall. Table 5.2 shows the obtained code clone classification accuracy with different machine learning classifiers.

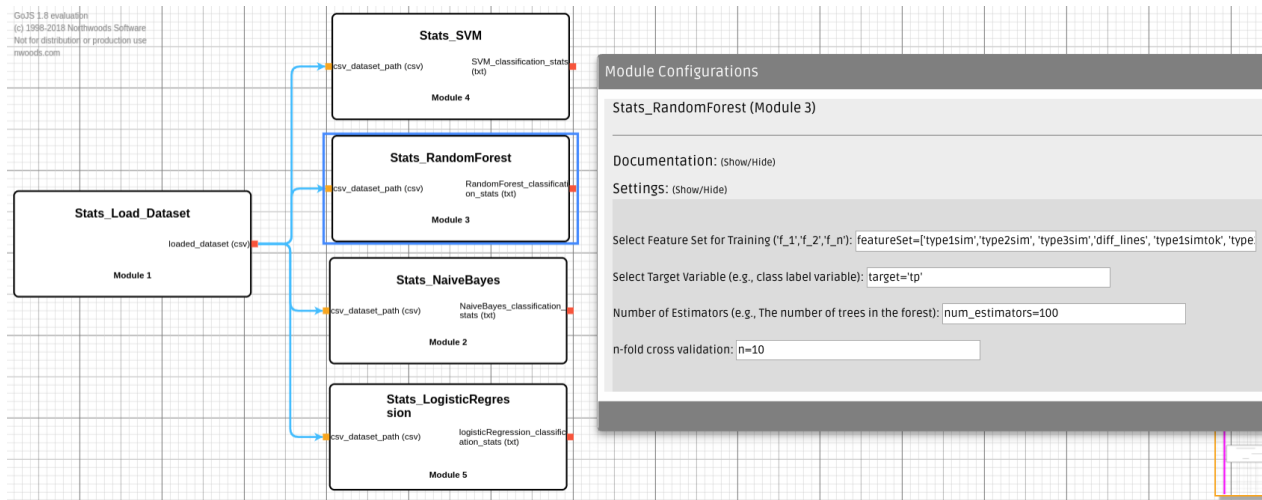


Figure 5.13: SciWorCS workflow for clone classification with different machine learning classifiers.

Table 5.2: Classification Accuracy Comparison for different Machine Learning Models.

#	Machine Learning Classifier	Classification Accuracy (%)
1	Random Forest	84.47
2	Random Tree	79.84
3	Naive Bayes	83
4	Bayes Network	81.79
5	Naive Bayes Updateable	82.99
6	Logistic Regression	85.06
7	K* Classifier	81.79
8	Decision Table	85
9	Artificial Neural Network	87.4

5.6.4 Code Clone Detection (Software Repository Analysis)

Existing studies show several motivations and use-cases towards collaborative SWfMSs [114, 201, 199, 114]. Different experimental studies can exploit the added advantages of collaboration for accelerating the overall process [201].

Code Clone. Code clones are similar pairs of code fragments in software systems [155, 156, 129]. Listing 5.11 and 5.12 demonstrate a pair of code clone. Studies show that, on average software systems often contain 7% to 23% of codes that are copied from one location to another (e.g., code clones) [156, 11, 99]. Developers often intentionally adopt code cloning to re-use the existing pieces of codes towards accelerating the software development process [156]. While not all of the code clones are considered harmful [98], several research studies show that in general code cloning is one the major causes of propagation and creation of new software bugs—due to the inconsistent changes of the clone fragments [155, 156]. Code clones are often responsible for inflating the overall software maintenance cost, and hence their successful detection has been one of the research problem in the domain of software engineering [156, 175]. In addition to the software maintenance, code clone detection is directly or indirectly used across several other research domains [154], such as Plagiarism Detection [160, 147, 119, 87, 14], Library Candidates Detection [43, 29], Origin Analysis [65, 185], Merging [84, 65], Software Evolution [185, 184, 188], Bug Detection [108, 89, 86, 149], Aspect Mining [103, 26], Program Understanding [93, 74, 146], Malicious Software Detection [190, 27], Copyright Infringement Detection [11, 96], Product Line Analysis [35, 97], Automatic Code Completion [9, 8] and so on.

```
1 int factorial(int N){
2     int f=1;//factorial result
3     for(int i=1;i<=N;i++)
4         f = f*i;
5     return f;
6 }
```

Listing 5.11: Sample Code Fragment 1.

```
1 int getFactorial(int Num){
2     int factorial=1;
3     for(int j=1;j<=Num;j++)
4         factorial = factorial*j;
5     return factorial;
6 }
```

Listing 5.12: Sample Code Fragment 2.

Throughout the life-cycle of a software system, the code clone fragments often undergo several changes, such as identifier name changes, addition/edition/removal of several statements, changes in the presentation and so on [155, 156]. For example, in the code clone fragments as demonstrated in Listing 5.11 and 5.12, the function names have been changed in line 1, line 2 demonstrates the difference in comments and also changes in the used identifier—which also has been propagated throughout the entire fragments and so on. These changes in the code fragments over time, thus often result in complication of the code clone detection process via simple string matching algorithms [157]. Code clone techniques hence, undergo several pre-processing and transformation steps, such as *pretty-printing* [157, 155], *normalization of the identifiers* [96, 157], *forming syntax tree* [104] of the code fragments and so on, prior to applying any matching algorithms. While a great many number of code clone detection tools and techniques have been proposed over the last decade (e.g., recent

study reports at least 198 such tools and techniques until 2017 [175]) for addressing specific types or aspects of clones, the tools and techniques often share much similarity in their corresponding steps and can be broadly categorized in some general detection steps or taxonomies, such as source transformation, normalization, pretty-printing, comparison and so on [155]. The clone detection techniques thus can be generalized as set and combination of the processing steps, which can be modularized towards the composition of workflows. This opens up the possibility of re-using the modular steps and also on focusing the case-specific clone detection rather than focusing on the implementation details of the modules—that is exploiting the advantages of scientific workflow concepts and SWfMSs [186, 3, 141]. For example, a workflow comprising the modular steps—*pretty-printing*, *removing comments* and *renaming of identifiers* result in the output as presented in Listing 5.13 and 5.14 for the corresponding input code fragments of Listing 5.11 and 5.12 respectively.

```

1 X X(X X){
2     X X=1;
3     for(X X=1;X<=X;X++)
4         X = X*X;
5     return X;
6 }

```

Listing 5.13: Pre-processed and Transformed Listing 5.11.

```

1 X X(X X){
2     X X=1;
3     for(X X=1;X<=X;X++)
4         X = X*X;
5     return X;
6 }

```

Listing 5.14: Pre-processed and Transformed Listing 5.12.

Fig. 5.14 demonstrates the workflow composition in SciWorCS for code clone detection. For the use-case, we leverage the modular steps from *NICAD* [157] to demonstrate the re-usability of the workflow modules towards clone detection process as per the given requirements. As input, the workflow takes the target software system repository and outputs the detected code clone pairs along with different statistics of the detected clones (e.g., number of detected clones, pair-wise similarity values and so on).

5.7 Conclusion

Realizing the compelling need of collaborative SWfMSs, several methods or techniques have been proposed and developed in recent years. Several methods have been proposed for the consistency management in workflow composition in collaborative setups [201, 199, 165, 164, 168, 167, 78]. However, in addition to the consistency management, studies show that collaborative SWfMSs also need to consider several other aspects, such as backdoor communication among the sub-group collaborators, sub-workflow execution, the relationship between scientific workflows and collaboration models and so on [199, 201, 114]. Besides, from the perspective CSCW, the collaborative systems often need to consider several other factors, such as problem solving, group decision making and so on to be effective [48].

From these study findings, we proposed an architecture for collaborative SWfMSs. In addition to the consistency management for collaborative workflow composition, the architecture address plugin-based tool integration, role-based access control on the workflow components, independent sub-workflow execution and

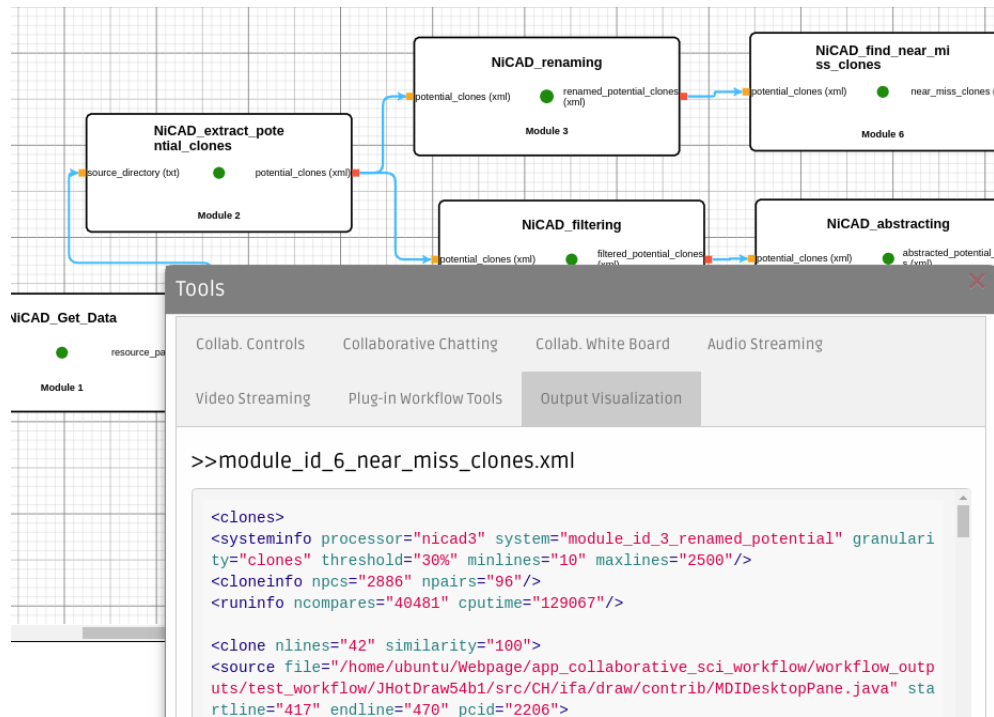


Figure 5.14: Code Clone Detection Workflow in SciWorCS.

monitoring for supporting sub-group collaborations, integrated communication technologies for group discussion, decision making or problem solving and plugin-based visualization framework. As a proof of concept, we also developed a collaborative SWfMS — SciWorCS. In this chapter, we presented different technical features of SciWorCS and also presented several real-world use-cases for scientific workflow composition, execution and data visualization. Our proposed architecture demonstrates promising results when evaluated in terms of different real-world scenarios of scientific workflow collaboration.

We also conducted several empirical and user studies with SciWorCS to study the human work patterns in term of collaborative SWfMSs. We present our study findings in Chapter 6.

6 UNDERSTANDING THE USER BEHAVIOR FOR COLLABORATIVE DATA ANALYSIS

Unlike the collaborative systems of unstructured objects such as text documents, the collaborative data analysis can often have different sets of considerations (e.g., dataflow, statistical analysis, visualizations, high inter-dependence of computational steps, execution scheduling and so on) that need to be addressed for a successful analysis process [114, 199]. Several methods [130, 91, 201, 165, 199, 164, 56] have been proposed in recent years towards a successful design of collaborative SWfMS. However, to the best of our knowledge, none of them considered usability analysis using human collaborators and rather relied only on computer generated simulated studies — where different concurrent threads simulate the collaborators for corresponding workflow updates over time. While the simulated studies are better for rigorous testing of the proposed methods [199, 201], in the context of CSCW, the human-centric studies are also important to evaluate the usability of the collaborative systems in real-world scenarios [142]. Hence, in this chapter, we present different user-studies for scientific workflow collaboration in terms of collaborative SWfMS. We leverage our proposed collaborative SWfMS — SciWorCS (e.g., as presented in Chapter 5) for conducting the user-studies.

In Section 6.1 of this chapter we first present the motivation and importance of such user-studies in real-world scenarios of workflow collaboration. We present the related works in Section 6.2. In Section 6.3, we then give a brief overview of our developed tool - SciWorCS (e.g., the details of the tool was presented in Chapter 5) and its used technical features for conducting our user studies. We then present our conducted study and experimental findings in Section 6.4. Section 6.5 contains the result discussion. We finally draw conclusion with a brief discussion on our future works in Section 6.6.

6.1 Motivation

In the recent big data era scientific experiments need to handle massive amounts of heterogeneous data [111, 115]. While these data-intensive experiments open up several possibilities of interesting knowledge discoveries (such as, by statistical analysis, applying some machine learning models and so on), they also impose several challenges for a successful analysis process such as, failure handling, optimal task scheduling, big data visualization, distributed job execution, real-time job monitoring and so on [111]. These challenges often stands as barriers to focusing on the data analysis task itself, and require significant amounts of time

and efforts [201, 165, 199]. Hence, Scientific Workflow Management Systems (*SWfMSs*) - framework for composition and execution of series of computational and data manipulation - are widely being used in recent years to address these challenges while accelerating the overall data analysis process [201, 199, 111, 55]. In addition to the job management, the visual programming front-end (e.g., a visual graph representation of dataflow) have resulted in significant popularity of SWfMSs for even non-computer science background users for domain-specific data analysis tasks [42, 56, 186, 66]. As a result of extensive research on this domain, several SWfMSs have been proposed and developed over the last decade [66, 111, 24, 181, 138, 109].

However with the rapid increase in complexity, volumes, and dimensions of heterogeneous data, the big data analytic workflows often goes beyond the scope of an individual for a successful data analysis process and hence, requires a collaboration of multiples scientists [199, 201, 165, 168]. As none of the existing SWfMSs supports real-time collaboration [201, 130, 165, 164], researchers often manually send (e.g., via e-mail and so on) or upload the workflows to some social shared spaces such as *myExperiment*[44] for collaboration [201, 199]. For example, around 3910 such scientific workflows have been shared among 10665 members (as last noted in August 2018) for collaboration in *myExperiment*[44]. Realizing the compelling need of such scientific artifact collaboration, researchers have proposed several methods for collaborative SWfMSs in recent years [130, 164, 199, 201, 165, 167]. However, although the existing techniques show promising results (e.g., for consistency management and so on) from several computer generated simulated studies [199, 56, 201] or theoretical use-cases [165, 164, 167], none of the studies considered human factors, such as adapted work patterns, data analysis problem solving, challenges and so on for scientific experiments from collaborative SWfMSs perspective [130]. Unlike collaborative text or graphics editing systems the scientific workflows are often more structured where one module can be highly dependent on another in their execution forming a hierarchical relation among them [201, 66, 130, 111]. Even any minor changes in any part of a workflow can significantly impact the other part of the collaborative workflow in execution and data manipulation [56, 55]- which often make the problem notably different than that of unstructured document collaborative systems, such as text or graphics editing systems [130, 201, 165]. Studying and understanding the human engagement or work patterns for collaborative data analysis, hence is important towards accelerating the emerging data analysis (e.g., by application of machine learning, data mining and so on) process with the aid of CSCW [68, 201, 91, 58].

While researchers have agreed upon the necessity of collaborative SWfMSs [130, 91, 201, 165, 199, 164, 56], there has not been any fully functional one that accumulates the discrete ideas and proposed techniques in recent years [130, 56]. Hence, for the study, we developed a cloud-based prototype collaborative SWfMSs - *SciWorCS* for collaborative data analysis among researchers. For the development of *SciWorCS*, we considered several recently proposed techniques on discrete aspects (such as multiple locking schemes for consistency management [130, 201, 199], job queuing for collaborative execution [66] and so on) envisioning collaborative data analysis with SWfMSs. We studied the impacts of such variants of techniques in the engagement of the collaborators in data analysis tasks. We conducted several user studies involving multiple researchers

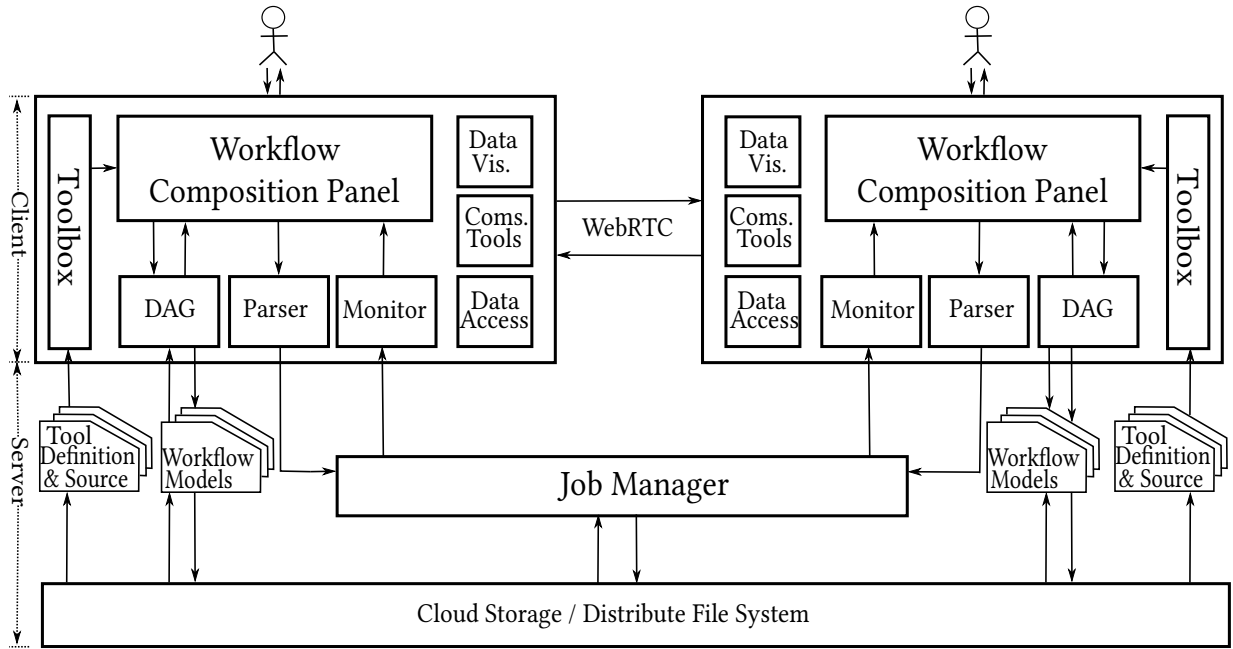


Figure 6.1: High-level Architecture of SciWorCS (e.g., as details presented in Chapter 5)

from a local university. In our study, participants were asked for some data analysis tasks, such as real-world workflows from *myExperiment* [44], building machine learning classification models for a given dataset and so on.

Our study reveals that, while the collaboration, in general, accelerates the data analysis process, the engagement and work patterns of the collaborators often are highly impacted by the nature of the data analysis problem. For example, when the data analysis tasks involve building some machine learning models, collaborators often engaged more in discussion about the possible selection of algorithms, their configurations and so on prior to the composition of the workflow towards attaining maximum performance (e.g., classification accuracy). In such cases, collaborators were also found to be more aligned towards ‘scribe’ - style of work for aiding the discussion process and converging towards the solution, in comparison to some data manipulation or transformation tasks (e.g., as discussed in details in Section 6.4).

Our work makes following two main contributions:

1. We present our findings towards the human factor in collaborative data analysis using SWfMSs. To the best of our knowledge, this is the first work that considered such setups unlike some computer generated simulated studies. Our findings will help towards a better design of collaborative systems for emerging data analysis areas.
2. We present SciWorCS¹- as a byproduct of the study, which can be used for future studies on the domain.

¹<https://github.com/pseudoPixels/SciWorCS>

6.2 Related Works

In this section, we first present the related works on - CSCW in aiding the scientific experiments (i.e., in Section 6.2.1), and we then discuss the recent works towards collaborative data analysis with SWfMSs (i.e., in Section 6.2.2).

6.2.1 CSCW in Aiding Scientific Experiments

Several recent types of research assert the necessity of CSCW in supporting complex scientific experiments that require collaboration among multiple researchers [41, 91, 68, 201]. Jirotko et al. from their investigation studies presented that, while the relation of scientific experiments (i.e., such as e-Science) and CSCW are relatively nascent one, they exhibit significant potentials in answering complex research questions and in important knowledge discovery [91]. Hence, over the past few years, several research studies have been conducted in understanding human behavior and also resulted in different proposed tools and techniques for collaboration in scientific experiments [68, 201, 91, 58].

A number of studies have been conducted in the recent years for gaining an in-depth understanding of scientific work practices such as, how scientific experiments are conducted, how research artifacts are shared, how scientists interact for tools and technologies and so on [91, 116, 150, 134]. While such investigation works often target divergent of scientific experiments (e.g., the Electronic Medical Record (EMR) [77], Breast Cancer Screening [92] and so on), they generally aim in providing important insights on challenges and design implications of CSCW systems towards virtual work-space for collaborative scientific experiments [116, 91].

6.2.2 Towards Collaborative Data Analysis

Large-scale scientific experiments often take advantages of SWfMSs for modeling the overall data analysis and manipulation process comprising of different computational steps for input data loading, transformation, aggregation and so on [111] where, SWfMSs work as a framework for supporting the specification, modification, execution, failure handling, and monitoring of the data-intensive tasks [111, 114]. With the increase of data complexity and volume, extensive research has been done on this domain resulting in a number of proposed SWfMSs architecture and corresponding implementation. Some of the modern popular SWfMSs are: *Galaxy* [66], *Taverna* [141], *Kepler* [115], *Pegasus* [47], *VisTrails* [31], *Triana* [181], *VIEW* [109], *Chiron* [138], *GridNexus* [24]. However, as the scientific data complexity, dimension and volume increase significantly in recent time, researchers of different domains try to exploit the CSCW - methodologies to accelerate the analysis process efficiently [91, 92]. These real-time collaborative techniques hence have gained significant focus envisioning collaborative SWfMSs [130, 201].

Lu *et al.* [114] studied several motivations opportunities for collaborative SWfMSs from the perspective of large-scale and multidisciplinary research projects. A number of methods have been proposed for consistency management of the shared workflow in a collaborative environment. Zhang *et al.* [199] studied the

concept of turn based locking scheme in the context of collaborative SWfMSs for facilitating the consistency management. In such setup, each collaborator generally has only the *Read* access to the shared workflow. Collaborators request and compete for the floor for carrying out any update or transaction on the workflow (e.g., *Read & Write* access). Fei *et al.* [56] and Zhang *et al.* [201] presented locking schemes by allowing only descendent module locks (e.g., descendent nodes of the workflow DAG [111]).

Sipos *et al.* [164] used two lock modes - *User* and *System* locks. Fei *et al.* [56] proposed a lock compatibility matrix for a set of six pre-defined modes of locks. Besides, techniques have also been studied for extending the single-user Grid portals to a collaborative environment [168, 165].

While these proposed methods show promising results in computer generated simulated studies, none of them considered the human factors for usability and engagement in collaborative SWfMSs. Hence, in this chapter of the thesis, we present empirical evidence on the usability of such collaborative SWfMSs. Our study aims to answer several questions on this recent research domain, such as *What styles of works people adopt for collaborative data analysis?*, *What confounding factors impact the collaborative analysis?*, *How people find it different in contrast to some other collaborative systems, such as collaborative text or graphics editing systems?* *Does collaborative SWfMSs even works towards a data analysis problem solving?*

6.3 Implementation Details

Prior to presenting our experimental studies, in this section, for continuity we first give a brief overview of SciWorCS - our developed tool for the collaborative data analysis study (e.g., the details on the SciWorCS has been presented in Chapter 5). The SciWorCS implementation is a cloud-based system hosted in a Linux Server. We used Python 2.7 as the server side language. On the other hand HTML5, CSS and JavaScript were used for client side programming. We also used Ajax for asynchronous server communications.

SciWorCS Editor and Panels

Fig. 6.2 shows a screenshot of the collaborative scientific workflow composition system. The panel labeled as ‘A’, contains all the workflow components such as workflow modules, saved workflows, shared workflows and so on. The collaborative composition of the workflow is done on panel ‘B’. The modules can be configured using the corresponding attributes in the popup panel ‘C’. Panel ‘D’ shows a list of collaborators and their current online/offline status. The list of the workflow outputs is shown in panel ‘E’. New dataset can be browsed and uploaded to the central server for analysis from the panel ‘F’.

Collaborative Composition

Fig. 6.3 shows a screenshot of the collaborative workflow composition panel. We adapted the proposed locking scheme for the consistency management while collaborative workflow composition. The module and attributes are color-coded to represent their corresponding lock states to the collaborators. For example, the green

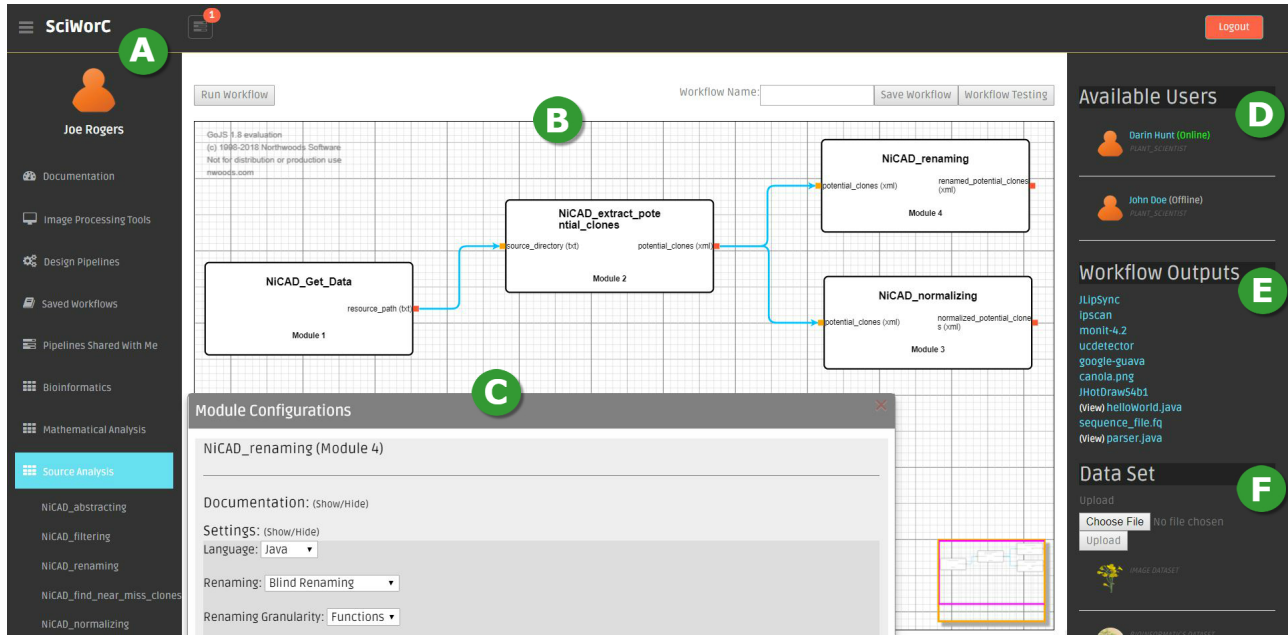


Figure 6.2: Prototype Implementation of Collaborative Data Analysis Framework

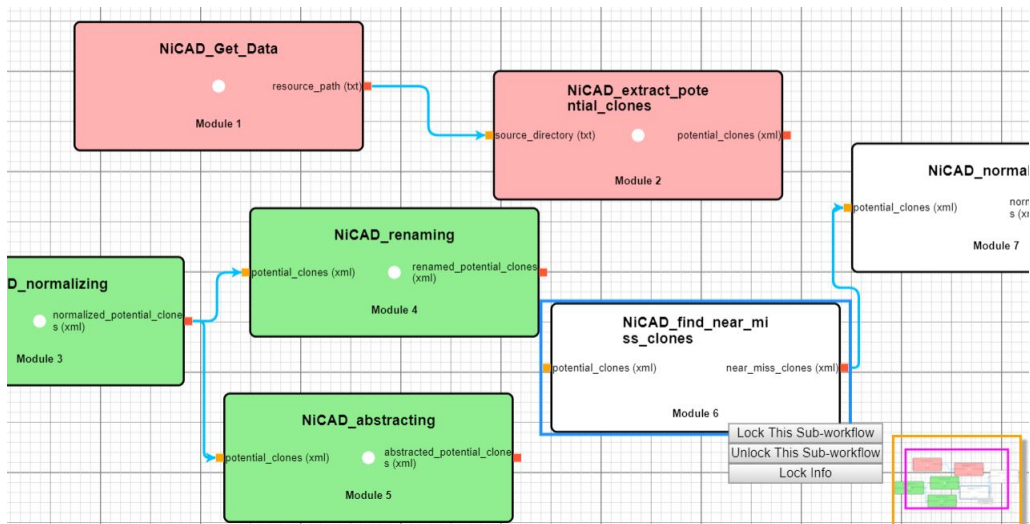


Figure 6.3: Collaboration Process for Consistency Management Handling Concurrent Conflicting Operations

color-coded sub-workflow (i.e., comprising of Modules 3, 4 and 5) denotes the locked sub-workflow by this collaborator, the red color coded sub-workflow (i.e., comprising of Modules 1 and 2) shows the sub-workflow currently locked by other remote collaborators and the remaining white colored workflow components (i.e., Module 6 and 7) represent no collaborators currently hold locks on those corresponding components. Collaborators can request, release or see the current lock status of any corresponding workflow components. For example, the similar options has been invoked (i.e., with *Mouse Right-Click*) in the sub-workflow with root node - Module 6.

6.4 Experimental Studies and Results

6.4.1 Experimental Setups

Primitive Operations for Dataflow Structure Update

Table 6.1: Dataflow Structure Updates Operations.

#	Dataflow Update Operation
1	New Module Addition to the Workflow
2	New Datalink Relation Addition From a Module
3	New Datalink Relation Addition To a Module
4	Attribute Configuration Update of a Module
5	Source Update of an Existing Datalink
6	Destination Update of an Existing Datalink
7	Sub-workflow Lock Access Request
8	Sub-workflow Lock Access Release

6.4.2 Study 1: Collaborative Composition Patterns

In Study 1, we aim to understand the collaborative composition patterns of dataflow structure comprising of different modular data processing steps. Study reports divergent styles of works or strategies adapted by users for varying collaborative systems, such as collaborative *Document Writing* [142, 135, 136], *Software Development & Management* [79, 198, 18], *Computer-Aided Design (CAD)* systems [180, 107] and so on. For a collaborative system, researchers study the adapted styles of works to better understand human behaviour exploiting new sets of tools and technologies for collaboration [19]. For example, Olson et al. [142] recently presented their empirical study on how people write documents together leveraging the modern collaborative technologies and tools such as *Google Docs* and so on. Hence, here we conduct empirical study for investigating the work patterns in collaborative data analysis environments.

Table 6.2: Primitive Operations For Component Access and User Interaction in Collaborative Data Analysis.

Operation Type	Collaborative Operation
Comp. Access/Update	Sub-workflow Access Request
	Sub-workflow Access Release
	Module Attribute Access Request
	Module Attribute Access Release
	Module Attribute Update
DAG View Update	Node Position Update
	Datalink Position Update
User Interaction	Textual Comm. (P2P/Room Chat)
	Audio/Visual Communication
	Collab. Virtual Whiteboard
	Telepointer Communication
	Collab. Input/Output Visualization

In case of complex data analysis process one modular computation is often dependent on other predecessor modular step(s) forming a dependency relation (i.e., DAG). Some of the recent proposed locking schemes for collaborative SWfMSs are, locking entire workflow DAG in turns for collaborators [199], locking hierarchical descendant modules [56, 201, 165], attribute level locking [130] and so on. While these proposed methods conducted several computer simulated studies to test their performances for consistency management, none of them considered usability analysis in terms of human collaborators [130, 201]. Unlike these computer simulated studies, human collaborators might engage in varying strategies for the workflow composition, such as one collaborator composes (i.e., as ‘*scribe*’ in collaborative document writing [142]) while other dictates or discusses the overall process, sequential composition, parallel composition via different roles assignment, parallel composition via divide and conquer of the overall tasks and so on. Besides, the adapted strategies often might be impacted by several other factors such as the number of modules, DAG complexity with higher average dependency degree, group size, used locking schemes and so on. Hence, in this study we answer the following two research questions:

- **RQ 1:** What styles of works do collaborators engage in for scientific workflow composition for collaborative data analysis?
- **RQ 2:** What confounding factors influence the styles of works for collaborative workflow composition?

Table 6.3: Considered Arbitrary Scientific Workflows from *myExperiments* [44] for the Study.

W. ID	Workflow Summary
4095	Paired-end reads assembly after FastQ groomer using a Migale modified version of Velvet tool.
4094	Workflow used when applying the CPB2012 Basic Protocol 3; Peaks for ChIP-seq data using MACS14.
2944	Transform FastQ to FASTA, using the tools, Groomer, Filter FastQ, FastQ Trimmer.
2939	Retrieves Genome and SNP data from UCSC for a particular chromosome. Finds Exons from SNPs.

Tasks and Stimulus

For this study on collaborative workflow composition pattern, we considered five real-world scientific workflows for varying *Bioinformatics* data transformation and analysis tasks. These workflows were selected arbitrarily from *myExperiment* [44] as presented in Table 6.3. In total, these workflows require 19 unique computational modules, which were integrated into our collaborative framework for the study. The computational modules are the building blocks of the workflows for the collaborative composition of the study. To mitigate the bias of problem solving complexity (i.e., we present our study and findings on problem solving in *Study 2*) while acquiring the collaborative composition patterns, the workflow structures were printed and provided to the participants for collaborative composition. For a given workflow, participants had to select (i.e., via Mouse Left-Click) the required computational modules available in the tool panel (i.e., Panel ‘A’, Figure 6.2) of the collaborative framework. The selected modules appear in the composition panel (i.e., Panel ‘B’, Figure 6.2) for devising the required datalink relations among them via *Mouse Left-Click and Dragging* among the corresponding input/output ports of the modules. Participants follow a series of collaborative revisions and updates on the workflow such as, module configuration changes (i.e., Panel ‘C’, Figure 6.2, on *Double-Clicking* a module), *delete/update/addition* operations on the workflow components (i.e., modules or datalinks) and so on by discussion and consensus to complete the composition of the target workflow. Note that the collaborators’ access for such operations is also controlled by the selected locking schemes such as turned based [199], descendent module locking schemes [56, 201, 165] and so on for the consistency management. For example, for turned based locking scheme a collaborator need to request and get the write access prior to any such operations for the workflow composition, hierarchical descendant modules or attribute level locking allow collaborators to work independently on a selected region of the workflow (i.e., sub-workflow) and so on. We present the impacts of the locking schemes in ‘Results’ Section of the study.

While the locking schemes ensure consistent composition in the face of conflicting operations, participants use different available user interaction tools (i.e., as presented in Table 6.2) of the framework for orchestrating

the collaborative composition of the workflow. The telepointer (i.e., multiple cursors) information are passed among the collaborators for real-time group awareness on their location, movement and probable focus of attention [70] in the collaborative workflow. Collaborators also invoke other communication tools (i.e., audio communication, video conferencing, textual group/P2P chatting) of the framework for discussion and convergence on a plan. In addition to that, the collaborative discussion process is also assisted by the framework's data visualization and virtual whiteboard tools.

Experiment Procedure and Data Collection

For the study, participants were provided four real-world dataflow oriented workflows from *myExperiments* [44] for collaborative composition. The participants were introduced and trialed with SciWorCS editor prior to starting the study. To investigate the impact of workflow locking schemes, the study was conducted in two different sessions with different locking schemes. The sessions with different locking schemes were counterbalanced to mitigate any bias or confounding factors. Besides, the used locking schemes were also anonymized to avoid any bias.

All the generated events and operations (e.g., as presented in Table 6.2) by the participants were logged in the background independently for the two sessions. Every log entries were also mapped with timestamps for later analysis. Participants were asked to fill out NASA-TLX [75] questionnaires after each of the sessions of the study.

Results

For the collaborative composition task in this study, we found that collaborators more often adopt the divide and conquer work approach towards completing the composition. The clear target for the mere composition task in collaboration, resulted in more or less fair task splits among the collaborators. Prior to starting the task, the collaborators were found to make plan via discussion for approaching the problem. In addition to the discussion, individual collaborators were also found to directly contribute to the composition (e.g., via module/datalink addition, deletion and so on) - in oppose to the 'scribe' style of work (e.g., where a single collaborator is responsible for entire composition). For example, for a collaborative composition, two collaborators had - 65% and 35% splits of operations (e.g., module/datalink addition, deletion and so on) out of total 105 operations for the workflow composition. Besides, the total engagement in the discussion also showed a fair split of 55% and 45%. The similar trends were also found among other collaborating groups for the composition task.

From an individual perspective in a collaborative group, a participant's results also exhibit contribution in different aspects of the composition. For example, in a collaborative group, a member contributed 15% in discussion, 50% in edit operations (e.g., module/datalink addition, deletion and so on) and 35% in other management operations towards the composition, such as DAG view update, access request/wait/release and so on, out of all the generated events by the collaborator in the composition. Similarly, another collaborator

also found to split the self-events in 23% in decision-making discussion, 47% in edit operations and 30% in other management operations. That is, the results demonstrate that the collaborators in overall, adapted the task splits and also individuals contributed in different aspects of the composition. For example, Fig. 6.4 illustrates the engagement of the participants in collaborative composition. For more granular pattern visualization, a whole collaborative composition (e.g., from the log) has been divided into two sessions and plotted the contribution of the collaborators for the corresponding sessions. The graph illustrates that the collaborators showed their engagement in comprehending the composition problem across different sessions.

The chat log of the collaborators also exhibits similar patterns, such as - ‘Collab. #1: ... Do you have a way to proceed? I was thinking lets get [add to workflow] the input modules first?...’, ‘Collab. #2: ... Alright. Go ahead ...’, ‘Collab. #1: ... I have set all the user inputs [modules]... Would you like to do [add to the workflow] the second layer [from the provided reference]?...’ .

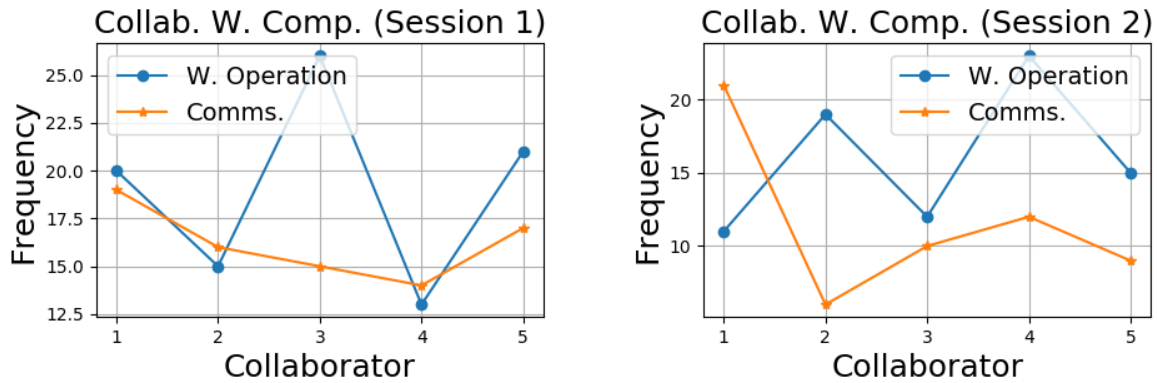


Figure 6.4: Collaborative Composition Work Patterns

Fig. 6.5 illustrates the impacts of collaborative workflow object controlling techniques (e.g., locking schemes) on the participants. The collected NASA-TLX responses from the participants have been plotted in the graph for the two used locking schemes (e.g., turn based locking [199] and sub-workflow locking [130, 201, 165, 56] via attribute) in the study. The graph illustrates some differences in collaborator’s NASA-TLX workload for the two types of locking schemes. For example, turn based locking scheme exhibits lesser mental demand while a higher physical demand in comprehending the problem in collaboration. For turn based locking, an individual collaborator could focus more on her/his assigned task on getting the floor and released the floor on completion of the assigned task. On the other hand, in case of sub-workflow locking schemes [130, 201, 165, 56], such as attribute level locking [130], collaborators had to be aware for releasing the sub-workflow components for other members and also for checking the access releases on other components. The similar pattern is also evident for the required effort. The single threaded nature of turn based locking hence resulted in lower mental demand in comparison. On the contrary, the parallel work-ability in sub-workflow locking resulted in comparatively lesser physical demand while maximizing the performance than

that of turn based locking.

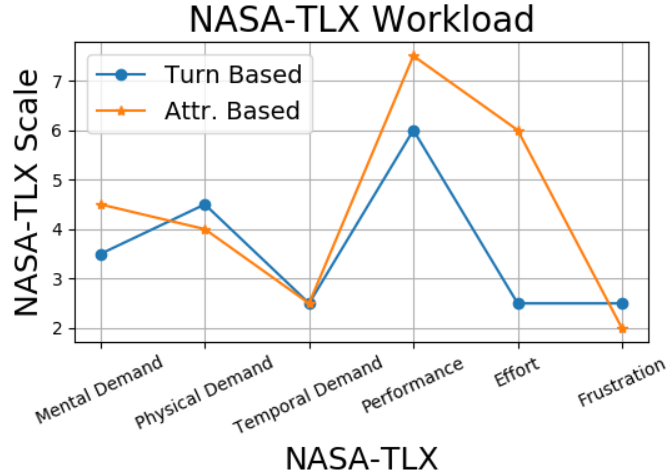


Figure 6.5: NASA-TLX workload for collaborative composition in terms of different locking schemes

6.4.3 Study 2: Collaborative Dataflow Problem Solving and Convergence on a Plan

The advancement of technologically-mediated collaboration has opened up several possibilities of the support of CSCW from management tasks to scientific discoveries [13, 195]. While the sheer volume of data from different science domains has motivated for important knowledge discovery via their efficient analysis, the increasing dimension and complexity such data-intensive experiments often go beyond the realm of an individual and require collaboration among multiple researchers [201]. Hence, application of CSCW on such modern data analysis experiments can accelerate the overall knowledge mining process [195]. In this study, we investigate the potential of collaborative SWfMSs for scientific data analysis experiments.

We generalize the scientific data analysis experiments as *Problem Solving* task, which has been an integral part of CSCW [195, 33]. However, researchers agree upon that the collaborative problem solving tasks can be classified in two categories such as ‘*well-defined*’ and ‘*ill-defined*’, depending on the nature of the problem and solution definitions [33, 13]. The problems where the correct solution and also the required steps to reach the solution are well understood by the collaborators are termed as ‘*well-defined*’. Since the correct solution is known and also the steps are well structured such problems are often amenable to measurements [33]. On the other hand, in ‘*ill-defined*’ problems the solutions are not often exact and also there exists several paths or strategies towards the possible solution [33, 13]. In the case of scientific experiments, the nature of the problems might suggest collaborators to engage in divergent strategies. Hence, in this study we aim to investigate the adapted problem solving strategies for varying nature of scientific experiments with SWfMSs answering the following two research questions:

- **RQ 3:** How collaborators engage in for scientific data analysis problem solving (i.e., for ‘*well-defined*’ and ‘*ill-defined*’ problems)?
- **RQ 4:** How confounding factors influence the problem solving strategies for collaborative scientific data analysis?

Tasks and Stimulus

In this study, we consider two different tasks for the collaborators targeting the ‘*well-defined*’ and ‘*ill-defined*’ problems. Since we target the scientific data analysis tasks with SWfMSs, for a given problem, participants go through several collaborative activities such as brainstorming, reviewing, revision and so on while selecting the required computational modules, configuring module attributes, building dataflow relation among the modules and so on towards the target solution.

Task for ‘*well-defined*’. Studies [33, 179, 13] show that, some mathematical problems - where a given set of mathematical operators are used towards a known solution, often can be classified as ‘*well-defined*’ problems. Hence, for the ‘*well-defined*’ problem solving tasks, we used simple mathematical modules which are responsible for elementary arithmetic operations, such as *Addition, Subtraction, Division, Multiplication, and Power*. The integrated mathematical modules accept one or more numeric values and output a single resultant value as corresponding data files. As tasks, the participants are provided with a set of sample numeric input and output to predict and design the corresponding workflow that generates the similar output as sample dataset. Note that, these five simple mathematical modules can be used multiple times with complex datalink relations among them to yet solve much complicated numeric patterns. However, the prediction of such complex workflows can often be non-trivial making the solution ambiguous. Hence, in order to keep the problem ‘*well-defined*’, we limit the required design of the mathematical workflow to a simpler and fixed structure which are informed to the participants. For example, for our study the used arithmetic equation is $z = Ax^C \mathcal{O} By^D$, where x, y are sample inputs, z is the corresponding sample output, \mathcal{O} is any operator from the available mathematical modules and A, B, C, D are some integer constants. For a given of sets of sample inputs and outputs (i.e., x, y and z), participants predict the constants and the operator, \mathcal{O} . Note that, in the formulated task the solution is exact and known (i.e., matching sample inputs and outputs) and also the solution steps are fixed and well understood (i.e., given workflow structure). In addition to that, we set a smaller range of the constants to limit the exhaustive search space.

Task for ‘*ill-defined*’. Majority of the scientific experiments do not suggest clear direction on how to proceed or on the correctness of the solution and thus often exhibit ‘*ill-defined*’ problem nature [33]. Hence, for the task, we select a machine learning based classification problem. Participants collaboratively work towards a possible solution using the provided machine learning based computational modules from the collaborative SWfMSs. We select the machine learning based problem for its convergence to the recent real-world practices. SWfMSs are gaining significant popularity in the machine learning domain for different data analytic, Natural Language Processing (NLP) problem solving and so on [161, 85, 145] as they provide

Table 6.4: Publicly Available Dataset - ‘*Titanic: Machine Learning from Disaster*’ as Collected from *Kaggle* [95].

Variable	Definition
survival	Survival (0 = No, 1 = Yes)
pclass	Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
sex	Sex
age	Age in years
sibsp	Number of siblings/spouses aboard
parch	Number of parents/children aboard
ticket	Ticket number
fare	Passenger fare
cabin	Cabin number
embarked	Port of Embarkation

several features and services such as reusability, interoperability, monitoring and so on of the machine learning tools [162, 85].

We select a publicly available dataset - ‘*Titanic: Machine Learning from Disaster*’ [95] for the classification task. Table B.3, present the dataset definition as collected from *Kaggle* [95] - a community of data scientists and machine learners. The dataset is about the *RMS Titanic* passengers, where the task is to build a machine learning model to predict which sort of people were more likely to survive the sinking of Titanic. The machine learning model for the binary classification (i.e., whether survived or not) is built by selecting features such as ticket class, age, sex and so on of the corresponding passengers. Note that, the task is ‘*ill-defined*’ in the sense that there exist multiple solution paths and strategies in selecting the feature set, selecting the ‘correct’ machine learning model, configuring the selected machine learning model for training and so on [189, 106]. Collaborators iteratively converge on plans (i.e., feature, model selection and so on) via discussion, reviewing the performance of the trained model and so on. Collaborators’ discussion process is assisted via different statistical computational modules and collaborative visualization tools of the framework. For example, for the given classification task visualizing the data distribution for survival in terms of age, sex, ticket class and so on can be important for selecting the contributing features or the machine learning model.

Experiment Procedure and Data Collection

For the ‘well-defined’ task participants were introduced with the corresponding equation structure and the available arithmetic modules in SciWorCS. Participants were also asked to solve some example problems of similar type to ensure their understandability of the tasks prior to starting the study. In the study, participants were provided three sets of sample inputs and outputs, and instructed to maximize the number of solved problems within a time frame of 15 minutes. The generated events and operations by the collaborators

were logged with timestamps as previous study. Participants filled out NASA-TLX [75] questionnaire at the end of the task.

Participants were introduced with the *Titanic* survival classification problem and the corresponding dataset for the ‘ill-defined’ problem. The provided statistical analysis tools in SciWorCS for the tasks were - *Feature Distribution Analysis, Missing Value Statistics, Fill Missing Values By Median, Convert Categorical Variables to Numeric*. The provided machine learning modules were - *Logistic Regression, Support Vector Machine, kNN Classifier and Random Forest Classifier*. The modules were implemented on top of *scikit-learn* [143] - a Python based library for machine learning and data mining. Participants trialed on the modules (e.g., for available configurations, graph plots and so on) independently prior to starting the task. In the study, participants were asked to build machine learning model with the target of maximizing the classification accuracy. Like the previous studies, the events were logged with timestamps and participants finally filled NASA-TLX questionnaire.

Results

Fig. 6.6 illustrates the contribution of the collaborators for the ‘well’ and ‘ill-defined’ task. For ‘well-defined’ task, as noticeable from the graph, the collaborators engaged in the problem solving by often clear splits of different aspects of the tasks. Also, more or less even amount of communication among the collaborator for the task splits is also noticeable from the graph. For example, one collaborator communicated - ‘...let me create the workflow [as the equation structure] while you figure out the input [for solution]...’ for possible task splits. The task split for different dimensions of the task is also noticeable from the graphs and results. For example, for the problem solving task, the discussion split among a collaborating group found to be 60% and 40%. On the other hand, of the total amount of workflow update operations, the split was noticed to be 66% and 34%. In addition to that, the collaborators also participated other management related operations, such as DAG layout update, access request/wait/release and so on, with a split of 33% and 67%. There was some difference noticeable in the type of update operations or roles, but the overall contributions were found to be often fair, as evident from the event and chat logs. For example, in Group 1, while one collaborator (e.g., Collaborator #2) was more engaged in workflow creation (e.g., module addition, configuration updates or datalink addition), the other collaborator (e.g., Collaborator #1) contributed more by reviewing or revisioning (e.g., by commenting on the current dataflow state, removal of some modules on revision and so on) the dataflow. For example, on initial development and commit of the workflow by Collaborator #1, the workflow was further revised for the correct behavior by Collaborator #2. ‘... if $3*3*2$ [if such multiplication structure is used] the result is 18, [however] the expected result [from the given problem] is 12...’ - Collaborator #2 commented for possible fixes on revision.

However, for the later task the result exhibit some different patterns. For building the classification model, the fair task split and the engagement of the collaborators were not as prominent as other tasks. Although all collaborators were experienced in machine learning and data analysis, the contributions of the

collaborators show significant differences. In general, we found that the collaborators often observed the progress in oppose to active engagement (e.g., module/datalink addition, configuration update and so on), while a single collaborator scribes the dataflow for building the machine learning model. For example, as the graph illustrates the contribution in a collaborating group for ‘ill-defined’ task. In collaborating group 3, one collaborator contributed significantly more in the group. The similar pattern is also noticeable from the graph for group 4. Although there was some communication noticeable, the parallel contribution in workflow creation with updates operation (e.g., with active edit operation on modules/datalinks) were not too prominent for every collaborator, in oppose to previous studies. The communication and discussion were, of course, important for model selection, the configuration of the machine learning models. For example, the chat log of a collaborating group shows ‘Collab. #1: ... got 67.7% [accuracy] with Logistic Regression...’, ‘Collab. #2: ... might need to consider age [feature]...’, ‘Collab. #1: ... should we try knn [kNN classifier]?...’. While the discussion greatly helped in decision making, collaborators adapted scribe style of work towards building the machine learning models for data analysis.

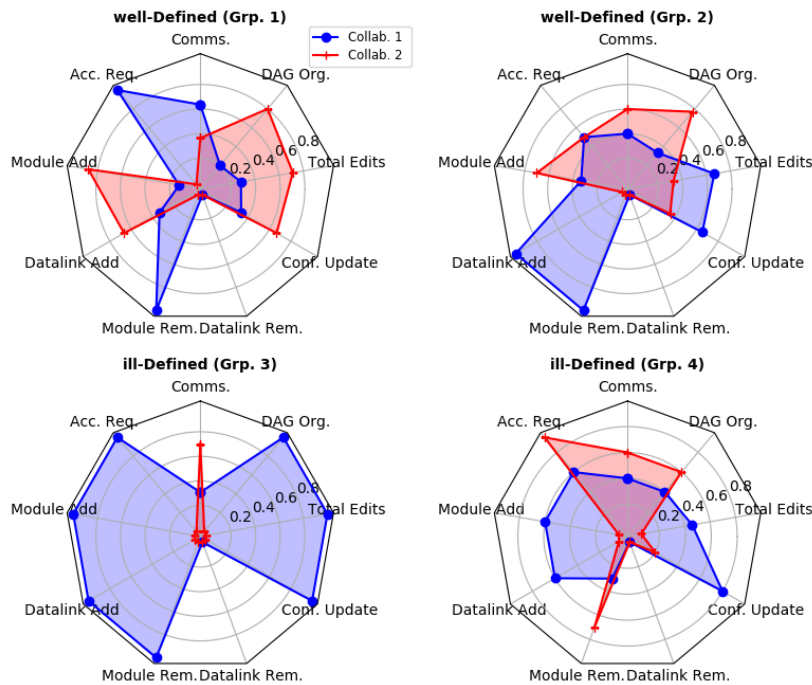


Figure 6.6: Collaborators’ Engagement and Contribution in Collaborative Data Analysis.

6.5 Result Discussion

The results of Study 1, indicate that collaborators engage in ‘*divide and conquer*’ strategies with more or less fair task splits among them when the task is a mere composition based on any fixed references or pre-planned

outline. The target of such composition being clear and straightforward, the adapted style of works among the collaborators for workflow composition often exhibits some similarity to collaborative document writing. For example, for collaborative document writing people often tend to split the task (e.g., in paragraphs, sections and so on) for separation of concern and also for accelerating the overall writing by parallelization, which also undergoes several revisions from time to time by collaborators [142]. The similar pattern was evident from the chat log of the collaborators, such as ‘*Collab. #1: ... How do you want to proceed...?*’, ‘*Collab. #2: ... you can do [add to workflow] the first seven module... this way we can double check each others work...?*’.

However, the Study 2 results exhibit some differences in the work pattern of the collaborators in comprehending the given data analysis task. The difference is even more prominent that involves developing some unstructured workflow models, such as statistical analysis on the dataset, building the ‘correct’ machine learning model and so on. In addition to the ‘ill-defined’ nature of the problems, the strong dependency relation among the computational modules for dataflow results in significant differences in comprehending the data analysis process in collaborative setups. Although the results exhibit some discussion among the collaborators in approaching the problem, the task splits or parallelization with separation of concerns for a given data analysis task was not too evident, unlike other collaborative systems, such as, Study 1, collaborative document writing [142, 144], graphics editing systems [37, 60] and so on. While the discussion greatly influenced the decision making, such as in selecting machine learning models, possible model configurations and so on, collaborators often followed scribe style of work (e.g., where mainly a single collaborator was responsible for addition/updates of modules/datalink) towards completing the data analysis task. The strong dependency nature among the data analysis steps might be a possible reason for such adapted work pattern. However, there might have several ways of improvement that needs exploration given the nature of such strong data dependency relation for data analysis problems. For example, some techniques for decoupling the dependency in the composition phase might help towards efficient task splits and adaption of other work patterns, such as parallelization, divide and conquer and so on, for even more accelerating the data analysis process.

6.6 Conclusion

As the complexity, volumes and dimensions of heterogeneous data increasing significantly in the recent years, the necessity of collaboration for such effective data analysis is even becoming more prominent. Hence, the *collaborative SWfMSs* have gained significant popularity in recent years [114, 199]. Several methods (e.g., locking schemes, access control techniques and so on) have also been proposed towards the design of a successful collaborative SWfMS. For the evaluation, these methods relied on computer generated simulated studies — where different concurrent threads simulate the collaborators for corresponding workflow updates over time. However, while the simulated studies provide better insight about the performance of the algorithms

[199, 201], in the context of CSCW understanding the user behavior is important towards effective design of collaborative systems [142, 81]. In order to address the lack of such studies in the context of collaborative SWfMSs, in this chapter we presented different findings from our conducted user-studies on real-world scientific workflow collaboration.

Our study reveals that the collaborative data analysis pattern can be highly impacted by the nature of the task. Our study demonstrates that, collaborators engage in *'divide and conquer'* strategies, when the solution of the tasks are clear or straightforward (e.g., 'well-defined' problems). The adapted styles of work in such scenario exhibit much similarity to collaborative document writing [142]. On the other hand, for 'ill-defined' problems the study results illustrate that, collaborators often engage in *'scribe'* style of work — where mainly a single collaborator mainly takes the responsibility of addition, update or deletion of workflow components while the collaborating group as a whole exhibits the trend of discussion towards these possible workflow updates. Our findings from the study can significantly contribute towards a more effective and efficient design of collaborative SWfMSs in terms of real-world usage scenarios. For example, the collaboration on 'ill-defined' problem solving exhibit significant opportunity of improvement by aiding the *'scribe'* style of work with the integration of CSCW techniques with collaborative SWfMSs. The study reveals that, the *'scribe'* style of work for collaborative data analysis can be facilitated with more CSCW tools for collaborative visualizations and discussion. On those insights, our future work direction includes real-time annotation, collaborative visualization, decoupling techniques of computational modules in analysis phases.

7 CONCLUSION

This chapter concludes the thesis. We present a summary of the thesis in Section 7.1, and finally, Section 7.2 outlines some future research directions from this thesis.

7.1 Summary

The generation of sheer amount of heterogeneous data on a daily basis by different areas of modern science have resulted in significant focus and popularity in Scientific Workflow Management System (*SWfMS*) [111, 3, 141, 201]. SWfMSs provide techniques for modeling re-usable modular scientific data processing steps and their dependency relations as Directed Acyclic Graph (DAG) [111]. SWfMSs are designed to efficiently support the specification, modification, execution, failure handling, and monitoring of the tasks in a scientific workflow [111, 3]. However, with the increase in complexity, dimension, and volume of scientific data, their effective analysis process is often beyond the scope of an individual and requires a collaboration of a research group instead [199]. Besides, some scientific domains essentially require collaboration as they are highly correlated among multiple research disciplines [201]. Though several SWfMSs have been proposed and developed over the last decade, such as Galaxy [66], Taverna [140], Kepler [115] and so on, none of them directly support collaboration among the users and generally operate in single user mode [201, 199, 200]. For the collaboration of such scientific workflows or artifacts, users need to go through several manual steps, such as directly sending the workflows to collaborators or uploading the workflows to some shared social spaces, such as myExperiment [44]—where the process of update, upload and download are repeated among collaborators a number of times towards the completion of collaborative workflow composition. Even though this manual collaboration process is time-consuming, does not support real-time editing or any management systems for considering different updates by the collaborators [199, 164, 165, 201, 56], still the constant increase in the number of such shared workflows (e.g., around 2895 such scientific artifacts shared for collaboration as *last noted in January 2018*) for collaboration demonstrates the compelling need of collaborative SWfMSs [201, 199].

Realizing the necessity of collaboration, the notion of Collaborative *Scientific Workflow Management Systems* (*SWfMSs*) was introduced and have gained significant focus in the recent years among the researchers [199, 201, 114, 165, 164, 167, 166, 78, 168, 56]. Studies show that Collaborative SWfMSs often have different set of challenges and requirements in contrast to single user based SWfMSs that we need to address towards a successful collaborative SWfMS [200, 114], such as the challenge of consistency management in the face

of concurrent conflicting operations [114, 165, 167, 168, 166, 199, 200], independent sub-workflow composition and execution by collaborating sub-groups, backdoor communication among the sub-group collaborators [201], access control policies among collaborators [17], relationship between scientific workflows and collaboration models [114, 78]. On this context of Collaborative SWfMSs, our thesis makes four major contribution as follows.

7.1.1 Fine Grained Attribute Level Locking Scheme

While the existing locking schemes for facilitating consistency management in terms of collaborative SWfMSs, in general operates on the module level of workflow DAG [199, 201, 165], we proposed a novel approach that works on finer attribute level for workflow component locking. Using the proposed approach, a large set of redundant workflow component locks can be minimized towards ensuing higher concurrency in collaborative SWfMSs. We conducted several computer simulated studies for validating the performance of the proposed method in terms of existing methods. Our studies show that the proposed method can reduce the average waiting time of a collaborator by up to 36.19% while increasing the average workflow update rate by up to 15.28%, which is promising.

7.1.2 Role Based Access Control for Collaborative SWfMSs

Study shows that, adequate access control policies among collaborators are often necessary in terms of collaborative SWfMSs [17]. While the locking schemes facilitate the consistency management in real-time collaborative workflow composition environment, the access control policy manages the sharing of workflow components among collaborators [4, 17]. We studied the concept of access control in the context of collaborative SWfMSs and also proposed a role based method for workflow component access controlling following the *Collaborative Interactive Application Methodology (CIAM)* [124] in terms of collaborative SWfMSs. We present our study on role based access control with a use-case of collaborative Plant Phenotyping and Genotyping research domain.

7.1.3 SciWorCS: Proposed Architecture Towards a Collaborative SWfMS

From our findings and investigations on existing research works towards collaborative SWfMSs, we propose an architecture of collaborative SWfMSs. While there have been several studies on discrete aspects of collaborative SWfMSs in recent years (e.g., consistency management [165, 199, 201], challenges and motivation of collaborative SWfMSs [114, 200], access control [17], to the best of our knowledge, no proposed SWfMS architecture or developed SWfMS directly support collaboration yet. In addition to leveraging the two prior proposed techniques (e.g., consistency management and access control), in our proposed architecture, we also address different aspects as investigated by different existing studies, such as handling independent sub-workflow composition and execution via job queuing [199, 201], relationship between scientific workflows

and collaboration models for provenance [114], seamless integration of the workflow components [78]. In the context of CSCW, study shows that while maintaining high responsiveness and concurrency are two primary requirements [173], providing different methodologies for group communication towards problem solving and decision making are also often very important towards the success of the system [48]. Hence, we also leverages different CSCW techniques and tools (e.g., the concept of telepointer, collaborative white boards, textual communication tools and so on) with the proposed architecture for successful way of group discussion and decision making in the process of collaborative data analysis. As a proof of concept of the proposed architecture, we developed *SciWorCS* — a **C**ollaborative **S**cientific **W**orkflow Management **S**ystem. We present different experimental use-cases for the evaluation SciWorCS, where it showed promising results.

7.1.4 Usability Study

For the evaluation, the existing locking schemes[56, 199, 201, 165, 164, 167] in general conducted several computer generated simulation experiments. While these simulation results depicts the performance of the proposed methods; however, to the best of our knowledge none of them considered the usability analysis of their methods in terms of real world setups. One possible reasoning for lack of such studies, is the unavailability of any existing working collaborative SWfMSs that considers beyond just consistency management [199]. However, in the context of CSCW, understanding the user behavior, styles of work and so on in collaborative environment is important towards designing effective and efficient system [142]. Hence, we make use of SciWorCS—that considers different aspects beyond just consistency management of collaborative SWfMSs as presented by related works. We conducted several user studies with SciWorCS to understand the user behavior and styles of work in the context of collaborative SWfMSs. Our studies demonstrate that, collaborative SWfMSs can significantly accelerate the data analysis process. Besides, the studies reveal several interesting findings towards further improvement of the concept of collaborative SWfMSs.

7.2 Future Work

We plan to investigate the following in future regarding collaborative SWfMSs.

7.2.1 Collaborative Provenance Models

Provenance management has become an important part in terms of SWfMSs [17, 114]. Provenance is about tracking the lineage of scientific data, such as the origin, derivation and context of a given dataset [21]. Recent study investigates several challenges and opportunities of provenance management in terms of collaborative SWfMSs [114]. The involvement of multiple users in terms of collaborative SWfMSs, adds a new dimension—the owner of data or workflow component to the overall provenance questions, such as the interaction and coordination among collaborators, origin and dependency information of a dataset and so on

[114]. We plan to work on such provenance model in future to answer different provenance questions from SciWorCS.

7.2.2 Studying More CSCW Techniques

We plan to investigate and study the impacts of more variants of CSCW tools and techniques for aiding the collaborative data analysis process using SWfMSs. We use our findings from our conducted user studies and collaborative systems of other domain for incorporating different CSCW tools and techniques, such as, annotation system on collaborative visualization framework, late join mechanism for the collaborators, adapting techniques for telepointer delay handling and so on. We plan to perform several empirical studies to better understand the effectiveness and usage patterns of the tools in terms of collaborative data analysis using collaborative SWfMSs.

7.2.3 Deadlock Awareness among Collaborators

The locking schemes [56, 199, 201, 165, 164, 167] on the workflow component facilitate the consistency management in the face of conflicting operation in collaborative SWfMSs. Similar to our proposed fine-grained locking scheme, the existing locking schemes use different database systems (e.g., hosted on a central server) to manage the lock requests on the workflow components. Hence, the locking schemes take advantages of the underlying database system, such as for concurrent transactions [201] (e.g., Atomicity, Consistency, Isolation and Durability). However, on the user interface level, the users might encounter the situation of deadlock, where two collaborators wait for one another for releasing the corresponding locked workflow components. As the situation occurs in collaborator levels, as a trivial solution, the deadlocks can be handled by discussion among the collaborators with the aid of provided CSCW tools for communication. However, we can use different CSCW techniques for facilitating the awareness of deadlock among collaborators to make the system even more interactive. As our future work, we plan to investigate different CSCW techniques to facilitate the deadlock awareness among collaborators in the context of collaborative SWfMSs.

7.2.4 Studying Collaborative Task Scheduling

In the context of data intensive scientific workflows, the processing and transferring of data sources among multiple collaborating groups can be an interesting scope of study in term of collaborative SWfMSs. An efficient collaborative task scheduling approach can significantly minimize the overall overhead in data transferring and processing.

7.2.5 Adaptation of SciWorCS in Source Code Repository Analysis

In the modern big data era software repositories have shown significant growth in complexity and volume comprising of millions of projects (e.g., 67 million+ projects on GitHub¹, 500,000+ projects in SourceForge² and so on) and enormous collection of information about software (e.g., 12.5 million+ active issues, 1.0 billion+ public commits alone in GitHub since September 2016). Like the bioinformatics domain, SWfMSs also show several potential towards textual analysis in the domain of source code repository analysis. Besides, the collaborative support of SciWorCS may reveal several interesting findings in such analysis process.

¹As per the data collected in July 2018 from: <https://octoverse.github.com>

²As per the data collected in July 2018 from: <https://sourceforge.net/about>

REFERENCES

- [1] Lemnatec. <http://www.lemnatec.com/products/>.
- [2] Plantcv. <http://plantcv.readthedocs.io/en/latest/>.
- [3] Enis Afgan, Dannon Baker, Marius Van den Beek, Daniel Blankenberg, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Carl Eberhard, et al. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research*, 44(W1):W3–W10, 2016.
- [4] Gail-Joon Ahn, Ravi Sandhu, Myong Kang, and Joon Park. Injecting rbac to secure a web-based workflow system. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 1–10. ACM, 2000.
- [5] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.*, 2015.
- [6] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, et al. Business process execution language for web services, 2003.
- [7] José Luis Araus and Jill E Cairns. Field high-throughput phenotyping: the new crop breeding frontier. *Trends in plant science*, 19(1):52–61, 2014.
- [8] Muhammad Asaduzzaman, Chanchal K Roy, Kevin A Schneider, and Daqing Hou. Csc: Simple, efficient, context sensitive code completion. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 71–80. IEEE, 2014.
- [9] Muhammad Asaduzzaman, Chanchal K Roy, Kevin A Schneider, and Daqing Hou. A simple, efficient, context-sensitive approach for code completion. *Journal of Software: Evolution and Process*, 28(7):512–541, 2016.
- [10] Brenda S Baker. A program for identifying duplicated code. *Computing Science and Statistics*, pages 49–49, 1993.
- [11] Brenda S Baker. On finding duplication and near-duplication in large software systems. In *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*, pages 86–95. IEEE, 1995.
- [12] Adam Barker and Jano Van Hemert. Scientific workflow: a survey and research directions. In *International Conference on Parallel Processing and Applied Mathematics*, pages 746–753. Springer, 2007.
- [13] Aaron Bauer and Zoran Popovic. Collaborative problem solving in an open-ended scientific discovery game. *PACMHCI*, 1(CSCW):22–1, 2017.
- [14] Boumediene Belkhouche, Anastasia Nix, and Johnette Hassell. Plagiarism detection in software designs. In *Proceedings of the 42nd annual Southeast regional conference*, pages 207–211. ACM, 2004.
- [15] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and evaluation of clone detection tools. *IEEE Transactions on software engineering*, 33(9), 2007.

- [16] Nicolas Bettenburg, Weyi Shang, Walid Ibrahim, Bram Adams, Ying Zou, and Ahmed E Hassan. An empirical study on inconsistent changes to code clones at release level. In *Reverse Engineering, 2009. WCRE'09. 16th Working Conference on*, pages 85–94. IEEE, 2009.
- [17] Fahima Bhuyan, Shiyong Lu, Robert Reynolds, Ishtiaq Ahmed, and Jia Zhang. Quality analysis for scientific workflow provenance access control policies. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 261–264. IEEE, 2018.
- [18] Christian Bird, David Pattison, Raissa D’Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35. ACM, 2008.
- [19] Jeremy Birnholtz and Steven Ibara. Tracking changes in collaborative writing: edits, visibility and group maintenance. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 809–818. ACM, 2012.
- [20] Kalina Bontcheva, Hamish Cunningham, Ian Roberts, Angus Roberts, Valentin Tablan, Niraj Aswani, and Genevieve Gorrell. Gate teamware: a web-based, collaborative text annotation framework. *Language Resources and Evaluation*, 47(4):1007–1029, 2013.
- [21] Rajendra Bose and James Frew. Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys (CSUR)*, 37(1):1–28, 2005.
- [22] Robert P Bostrom. Role conflict and ambiguity: Critical variables in the mis user-designer relationship. In *Proceedings of the seventeenth annual computer personnel research conference*, pages 88–115. ACM, 1980.
- [23] Gary Bradski and Adrian Kaehler. Opencv. *Dr. Dobbs journal of software tools*, 3, 2000.
- [24] Jeffrey L Brown, Clayton S Ferner, Thomas C Hudson, Ann E Stapleton, Ronald J Vetter, Tristan Carland, Andrew Martin, Jerry Martin, Allen Rawls, William J Shipman, et al. Gridnexus: A grid services scientific workflow system. *International Journal of Computer Information Science (IJCIS)*, 6(2):72–82, 2005.
- [25] Joseph Brown, Meg Pirrung, and Lee Ann McCue. Fqc dashboard: integrates fastqc results into a web-based, interactive, and extensible fastq quality control tool. *Bioinformatics*, 33(19):3137–3139, 2017.
- [26] Magiel Bruntink, Arie Van Deursen, Remco Van Engelen, and Tom Tourwe. On the use of clone detection for identifying crosscutting concern code. *IEEE Transactions on Software Engineering*, 31(10):804–818, 2005.
- [27] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. Using code normalization for fighting self-mutating malware. In *Proceedings of the International Symposium on Secure Software Engineering*, pages 37–44, 2006.
- [28] Joseph T Buck, Soonhoi Ha, Edward A Lee, and David G Messerschmitt. Ptolemy: A framework for simulating and prototyping heterogeneous systems. pages 1–34, 1994.
- [29] Elizabeth Burd and Malcolm Munro. Investigating the maintenance implications of the replication of code. In *Software Maintenance, 1997. Proceedings., International Conference on*, pages 322–329. IEEE, 1997.
- [30] Rajkumar Buyya and Srikumar Venugopal. The gridbus toolkit for service oriented grid and utility computing: An overview and status report. In *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, pages 19–66. IEEE, 2004.
- [31] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 745–747. ACM, 2006.

- [32] Leonardo Candela, Donatella Castelli, and Pasquale Pagano. Virtual research environments: an overview and a research agenda. *Data Science Journal*, 12:GRDI75–GRDI81, 2013.
- [33] Esther Care, Patrick Griffin, Claire Scoular, Nafisa Awwal, and Nathan Zoanetti. Collaborative problem solving tasks. In *Assessment and teaching of 21st century skills*, pages 85–104. Springer, 2015.
- [34] Alan Charpentier, Jean-Rémy Falleri, David Lo, and Laurent Réveillère. An empirical assessment of bellon’s clone benchmark. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, pages 20:1–20:10. ACM, 2015.
- [35] Gary Chastek, Patrick Donohoe, Kyo Chul Kang, and Steffen Thiel. Product line analysis: a practical introduction. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 2001.
- [36] Artem Chebotko, Shiyong Lu, Seunghan Chang, Farshad Fotouhi, and Ping Yang. Secure abstraction views for scientific workflow provenance querying. *IEEE Transactions on Services Computing*, (4):322–337, 2010.
- [37] Yuan Cheng, Fazhi He, Yiqi Wu, and Dejun Zhang. Meta-operation conflict resolution for human–human interaction in collaborative feature-based cad systems. *Cluster Computing*, 19(1):237–253, 2016.
- [38] Yuan Cheng, Fazhi He, Bin Xu, Soonhung Han, Xiantao Cai, and Yilin Chen. A multi-user selective undo/redo approach for collaborative cad systems. *Journal of Computational Design and Engineering*, 1(2):103–115, 2014.
- [39] James R Cordy, Charles D Halpern-Hamu, and Eric Promislow. Txl: A rapid prototyping system for programming language dialects. *Computer Languages*, 16(1):97–107, 1991.
- [40] James R Cordy and Chanchal K Roy. The NICAD clone detector. In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*, pages 219–220. IEEE, 2011.
- [41] Brian Corrie and Todd Zimmerman. Build it: Will they come? In *Media Space 20+ Years of Mediated Life*, pages 393–413. Springer, 2009.
- [42] Rafael Ferreira da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. A characterization of workflow management systems for extreme-scale applications. *Future Generation Computer Systems*, 75:228–238, 2017.
- [43] Neil Davey, Paul Barson, Simon Field, Ray Frank, and D Tansley. The development of a software clone detector. *International Journal of Applied Software Technology*, 1995.
- [44] David De, Roure Carole, and Goble Robert Stevens. The design and realisation of the myexperiment virtual research environment for social sharing of workflows. 2008.
- [45] Cleidson RB de Souza, David Redmiles, Li-Te Cheng, David Millen, and John Patterson. Sometimes you need to see through walls: a field study of application programming interfaces. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 63–71. ACM, 2004.
- [46] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future generation computer systems*, 25(5):528–540, 2009.
- [47] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.
- [48] Joanna DeFranco-Tommarello and F Deek. Collaborative software development: a discussion of problem solving models and groupware technologies. In *hicss*, page 41. IEEE, 2002.
- [49] Wanchun Dou, Jinjun Chen, Shaokun Fan, and SC Chueng. A context-and role-driven scientific workflow development pattern. *Concurrency and Computation: Practice and Experience*, 20(15):1741–1757, 2008.

- [50] Ekwa Duala-Ekoko and Martin P Robillard. Tracking code clones in evolving software. In *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, pages 158–167. IEEE, 2007.
- [51] Noah Fahlgren, Maximilian Feldman, Malia A Gehan, Melinda S Wilson, Christine Shyu, Douglas W Bryant, Steven T Hill, Colton J McEntee, Sankalpi N Warnasooriya, Indrajit Kumar, et al. A versatile phenotyping system and analytics platform reveals diverse temporal responses to water availability in setaria. *Molecular plant*, 8(10):1520–1535, 2015.
- [52] Noah Fahlgren, Malia A Gehan, and Ivan Baxter. Lights, camera, action: high-throughput plant phenotyping is ready for a close-up. *Current opinion in plant biology*, 24:93–99, 2015.
- [53] Thomas Fahringer, Radu Prodan, Rubing Duan, Jüurgen Hofer, Farrukh Nadeem, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, et al. Askalon: A development and grid computing environment for scientific workflows. *Workflows for e-Science*, pages 450–471, 2007.
- [54] FastQC. A quality control tool for high throughput sequence data. <http://www.bioinformatics.babraham.ac.uk/projects/fastqc/>.
- [55] Xubo Fei and Shiyong Lu. A dataflow-based scientific workflow composition framework. *IEEE Transactions on Services Computing*, 5(1):45–58, 2012.
- [56] Xubo Fei, Shiyong Lu, and Jia Zhang. A granular concurrency control for collaborative scientific workflow composition. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 410–417. IEEE, 2011.
- [57] Fabio Fiorani and Ulrich Schurr. Future scenarios for plant phenotyping. *Annual review of plant biology*, 64:267–291, 2013.
- [58] Stephen M Fiore and Travis J Wiltshire. Technology as teammate: Examining the role of external cognition in support of team cognitive processes. *Frontiers in psychology*, 7:1531, 2016.
- [59] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T Silva. Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3), 2008.
- [60] Liping Gao, Fangyu Yu, Qingkui Chen, and Naixue Xiong. Consistency maintenance of do and undo/redo operations in real-time collaborative bitmap editing systems. *Cluster Computing*, 19(1):255–267, 2016.
- [61] Ritu Garg and Awadhesh Kumar Singh. Multi-objective workflow grid scheduling using ϵ -fuzzy dominance sort based discrete particle swarm optimization. *The Journal of Supercomputing*, 68(2):709–732, 2014.
- [62] Ritu Garg and Awadhesh Kumar Singh. Adaptive workflow scheduling in grid computing based on dynamic resource availability. *Engineering Science and Technology, an International Journal*, 18(2):256–269, 2015.
- [63] Nils Göde and Rainer Koschke. Incremental clone detection. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pages 219–228. IEEE, 2009.
- [64] Nils Göde and Rainer Koschke. Frequency and risks of changes to clones. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 311–320. ACM, 2011.
- [65] Michael W Godfrey and Lijie Zou. Using origin analysis to detect merging and splitting of source code entities. *IEEE Transactions on Software Engineering*, 31(2):166–181, 2005.
- [66] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, 11(8):R86, 2010.
- [67] GoJS. Interactive JavaScript Diagrams in HTML. <https://gojs.net/latest/index.html>.

- [68] Ian Goldin. *World wide research: Reshaping the sciences and humanities*. MIT Press, 2010.
- [69] Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova, Frank Leymann, and Michael Reiter. Conventional workflow technology for scientific simulation. In *Guide to e-Science*, pages 323–352. Springer, 2011.
- [70] Saul Greenberg, Carl Gutwin, and Mark Roseman. Semantic telepointers for groupware. In *Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on*, pages 54–61. IEEE, 1996.
- [71] Ambient Software Evoluton Group. IJaDataset 2.0. <http://secold.org/projects/seclone>.
- [72] Jonathan Grudin. Why csw applications fail: problems in the design and evaluation of organizational interfaces. In *Proceedings of the 1988 ACM conference on Computer-supported cooperative work*, pages 85–93. ACM, 1988.
- [73] Carl Gutwin and Saul Greenberg. Support for group awareness in real-time desktop conferences. 1995.
- [74] Mark Harman. Search based software engineering for program comprehension. In *Program Comprehension, 2007. ICPC'07. 15th IEEE International Conference on*, pages 3–13. IEEE, 2007.
- [75] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.
- [76] Anja Hartmann, Tobias Czauderna, Roberto Hoffmann, Nils Stein, and Falk Schreiber. Htpheno: an image analysis pipeline for high-throughput plant phenotyping. *BMC bioinformatics*, 12(1):148, 2011.
- [77] Mark Hartswood, Rob Procter, Mark Rouncefield, and Roger Slack. Making a case in medical work: implications for the electronic medical record. *Computer Supported Cooperative Work (CSCW)*, 12(3):241–266, 2003.
- [78] Markus Held and Wolfgang Blochinger. Structured collaborative workflow design. *Future Generation Computer Systems*, 25(6):638–653, 2009.
- [79] Austin Z Henley, Kİvanç Muçlu, Maria Christakis, Scott D Fleming, and Christian Bird. Cfar: A tool to increase communication, productivity, and review quality in collaborative code reviews. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 157–170. ACM, 2018.
- [80] Robert J Henry. *Plant genotyping: the DNA fingerprinting of plants*. CABI, 2001.
- [81] Yoshiki Higo, Yasushi Ueda, Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. On software maintenance process improvement based on code clone analysis. *Product Focused Software Process Improvement*, pages 185–197, 2002.
- [82] Daniel Hoffman and Paul Strooper. Tools and techniques for java api testing. In *Software Engineering Conference, 2000. Proceedings. 2000 Australian*, pages 235–245. IEEE, 2000.
- [83] David Hollingsworth and UK Hampshire. Workflow management coalition: The workflow reference model. *Document Number TC00-1003*, 19:16, 1995.
- [84] James J Hunt and Walter F Tichy. Extensible language-aware merging. In *Software Maintenance, 2002. Proceedings. International Conference on*, pages 511–520. IEEE, 2002.
- [85] Nancy Ide, James Pustejovsky, Christopher Cieri, Eric Nyberg, Denise DiPersio, Chunqi Shi, Keith Suderman, Marc Verhagen, Di Wang, and Jonathan Wright. The language application grid. In *International Workshop on Worldwide Language Service Infrastructure*, pages 51–70. Springer, 2015.
- [86] Patricia Jablonski and Daqing Hou. Cren: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the ide. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, pages 16–20. ACM, 2007.

- [87] Hugo T. Jankowitz. Detecting plagiarism in student pascal programs. *The Computer Journal*, 31(1):1–8, 1988.
- [88] Lingxiao Jiang, Ghassan Mishherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th international conference on Software Engineering*, pages 96–105. IEEE Computer Society, 2007.
- [89] Lingxiao Jiang, Zhendong Su, and Edwin Chiu. Context-based detection of clone-related bugs. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 55–64. ACM, 2007.
- [90] Zhen Ming Jiang and Ahmed E Hassan. A framework for studying clones in large software systems. In *Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on*, pages 203–212. IEEE, 2007.
- [91] Marina Jirotko, Charlotte P Lee, and Gary M Olson. Supporting scientific collaboration: Methods, tools and concepts. *Computer Supported Cooperative Work (CSCW)*, 22(4-6):667–715, 2013.
- [92] Marina Jirotko, Rob Procter, Mark Hartswood, Roger Slack, Andrew Simpson, Catelijne Coopmans, Chris Hinds, and Alex Voss. Collaboration and trust in healthcare innovation: The ediamond case study. *Computer Supported Cooperative Work (CSCW)*, 14(4):369–398, 2005.
- [93] J Howard Johnson. Identifying redundancy in source code using fingerprints. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering-Volume 1*, pages 171–183. IBM Press, 1993.
- [94] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter? In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 485–495. IEEE, 2009.
- [95] Kaggle. Titanic: Machine Learning from Disaster. <https://www.kaggle.com/c/titanic/data>.
- [96] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002.
- [97] Cory Kapser and Michael W Godfrey. Improved tool support for the investigation of duplication in software. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 305–314. IEEE, 2005.
- [98] Cory Kapser and Michael W Godfrey. "cloning considered harmful" considered harmful. In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*, pages 19–28. IEEE, 2006.
- [99] Cory J Kapser and Michael W Godfrey. Supporting the analysis of clones in software systems. *Journal of Software: Evolution and Process*, 18(2):61–82, 2006.
- [100] Nickolaos Kavantzas. Web services choreography description language (ws-cdf) version 1.0. <http://www.w3.org/TR/ws-cdl-10/>, 2004.
- [101] VR Kavitha and N Suresh Kumar. A method for identifying loops in a workflow using petri nets. *Life Science Journal*, 10(3), 2013.
- [102] Iman Keivanloo, Feng Zhang, and Ying Zou. Threshold-free code clone detection for a large-scale heterogeneous java repository. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 201–210. IEEE, 2015.
- [103] Andy Kellens, Kim Mens, and Paolo Tonella. A survey of automated code-level aspect mining techniques. In *Transactions on aspect-oriented software development IV*, pages 143–162. Springer, 2007.
- [104] Rainer Koschke, Raimar Falke, and Pierre Frenzel. Clone detection using abstract syntax suffix trees. In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*, pages 253–262. IEEE, 2006.

- [105] Christos Kozanitis and David A. Patterson. Genap: a distributed sql interface for genomic data. In *BMC Bioinformatics*, 2016.
- [106] Ashraf Labib and Martin Read. Not just rearranging the deckchairs on the titanic: Learning from failures through risk and reliability analysis. *Safety science*, 51(1):397–413, 2013.
- [107] Min Li, Shuming Gao, and Charlie C Wang. Real-time collaborative design with heterogeneous cad systems based on neutral modeling commands. *Journal of Computing and Information Science in Engineering*, 7(2):113–125, 2007.
- [108] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on software Engineering*, 32(3):176–192, 2006.
- [109] Cui Lin, Shiyong Lu, Xubo Fei, Artem Chebotko, Darshan Pai, Zhaoqiang Lai, Farshad Fotouhi, and Jing Hua. A reference architecture for scientific workflow management systems and the view soa solution. *IEEE Transactions on Services Computing*, 2(1):79–92, 2009.
- [110] M. Lipp. The Danet Workflow Component. <http://www.wfmopen.sourceforge.net/>, 2007.
- [111] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015.
- [112] Salvatore Loreto and Simon Pietro Romano. *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser*. ” O’Reilly Media, Inc.”, 2014.
- [113] LSST. Large Synoptic Survey Telescope. <http://www.lsst.org/lsst/science>, 2009.
- [114] Shiyong Lu and Jia Zhang. Collaborative scientific workflows. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 527–534. IEEE, 2009.
- [115] Bertram Ludäscher, 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(10):1039–1065, 2006.
- [116] Paul Luff, Jon Hindmarsh, and Christian Heath. *Workplace studies: Recovering work practice and informing system design*. Cambridge university press, 2000.
- [117] Ruiqi Luo, Ping Yang, Shiyong Lu, and Mikhail Gofman. Analysis of scientific workflow provenance access control policies. In *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, pages 266–273. IEEE, 2012.
- [118] D.H. Honemann M. Robert, W.J. Evans and T.J. Balch. Robert’s Rules of Order. Newly Revised, 10th Edition. Perseus Publishing Company, 2000.
- [119] Udi Manber et al. Finding similar files in a large file system. In *Usenix Winter*, volume 94, pages 1–10, 1994.
- [120] Marta Mattoso, Claudia Werner, Guilherme Horta Travassos, Vanessa Braganholo, Eduardo Ogasawara, Daniel Oliveira, Sergio Cruz, Wallace Martinho, and Leonardo Murta. Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management*, 5(1):79–92, 2010.
- [121] Nirav Merchant, Eric Lyons, Stephen Goff, Matthew Vaughn, Doreen Ware, David Micklos, and Parker Antin. The iplant collaborative: Cyberinfrastructure for enabling data to discovery for the life sciences. *PLoS Biol*, 2016.
- [122] Ana I Molina, Miguel A Redondo, Manuel Ortega, and Ulrich Hoppe. Ciam: A methodology for the development of groupware user interfaces. *J. UCS*, 14(9):1435–1446, 2008.

- [123] Ana Isabel Molina, Miguel Angel Redondo, and Manuel Ortega. A conceptual and methodological framework for modeling interactive groupware applications. In *International Conference on Collaboration and Technology*, pages 413–420. Springer, 2006.
- [124] Ana Isabel Molina, Miguel Ángel Redondo, and Manuel Ortega. A methodological approach for user interface development of collaborative applications: A case study. *Science of Computer Programming*, 74(9):754–776, 2009.
- [125] Manishankar Mondal. *On the stability of software clones: A genealogy-based empirical study*. PhD thesis, University of Saskatchewan Saskatoon, 2013.
- [126] Manishankar Mondal, Md Saidur Rahman, Chanchal K Roy, and Kevin A Schneider. Is cloned code really stable? *Empirical Software Engineering*, pages 1–78, 2017.
- [127] Manishankar Mondal, Chanchal K Roy, and Kevin A Schneider. Bug propagation through code cloning: An empirical study. In *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*, pages 227–237. IEEE, 2017.
- [128] Manishankar Mondal, Chanchal K Roy, and Kevin A Schneider. Does cloned code increase maintenance effort? In *Software Clones (IWSC), 2017 IEEE 11th International Workshop on*, pages 1–7. IEEE, 2017.
- [129] G. Mostaeen, Jeffrey Svajlenko, Banani Roy, Chanchal K. Roy, and K. Schneider. On the use of machine learning techniques towards the design of cloud based automatic code clone validation tools. In *Source Code Analysis and Manipulation, 2018. SCAM 2018. 18th IEEE International Working Conference on*. IEEE, 2018.
- [130] Golam Mostaeen, Banani Roy, Chanchal K. Roy, and Kevin A. Schneider. Fine-grained attribute level locking scheme for collaborative scientific workflow development. In *Services Computing (SCC), 2018 IEEE International Conference on*, pages 273–277. IEEE, 2018.
- [131] Jonathan Munson and Prasun Dewan. A concurrency control framework for collaborative systems. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 278–287. ACM, 1996.
- [132] K. Murphy. Machine learning: a probabilistic perspective, 2012.
- [133] NCI. Cancer Biomedical Informatics Grid (caBIG). <https://cabig.nci.nih.gov/>.
- [134] Davide Nicolini. *Practice theory, work, and organization: An introduction*. OUP Oxford, 2012.
- [135] Sylvie Noël and Jean-Marc Robert. How the web is used to support collaborative writing. *Behaviour & Information Technology*, 22(4):245–262, 2003.
- [136] Sylvie Noël and Jean-Marc Robert. Empirical study on collaborative writing: What do co-authors do, use, and like? *Computer Supported Cooperative Work (CSCW)*, 13(1):63–89, 2004.
- [137] John T Nosek. Augmenting the social construction of knowledge and artifacts. Technical report, Temple Univ. Philadelphia, Dept. Of Computer And Information Sciences, 1998.
- [138] Eduardo Ogasawara, Jonas Dias, Vitor Silva, Fernando Chirigati, Daniel Oliveira, Fabio Porto, Patrick Valdúriez, and Marta Mattoso. Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience*, 25(16):2327–2341, 2013.
- [139] T Oinn. Xscuff language reference. *Internet: Available: www.ebi.ac.uk/tmo/mygrid/XScuffSpecification.html [October 14, 2009]*, 2004.
- [140] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

- [141] Tom Oinn, Mark Greenwood, Matthew Addis, M Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, et al. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, 2006.
- [142] Judith S Olson, Dakuo Wang, Gary M Olson, and Jingwen Zhang. How people write together now: Beginning the investigation with advanced undergraduates in a project course. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 24(1):4, 2017.
- [143] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- [144] Jeffrey M Perkel. Scientific writing: the online cooperative: collaborative browser-based tools aim to change the way researchers write and publish their papers. *Nature*, 514(7520):127–129, 2014.
- [145] Matic Perovšek, Janez Kranjc, Tomaž Erjavec, Bojan Cestnik, and Nada Lavrač. Textflows: A visual programming platform for text mining and natural language processing. *Science of Computer Programming*, 121:128–152, 2016.
- [146] Denys Poshyvanyk and Andrian Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In *null*, pages 37–48. IEEE, 2007.
- [147] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. Finding plagiarisms among a set of programs with jplag. *J. UCS*, 8(11):1016, 2002.
- [148] Radu Prodan and Thomas Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694. ACM, 2005.
- [149] Mohammad Masudur Rahman and Chanchal K Roy. Improving ir-based bug localization with context-aware query reformulation. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 621–632. ACM, 2018.
- [150] David Randall, Richard Harper, and Mark Rouncefield. *Fieldwork for design: theory and practice*. Springer Science & Business Media, 2007.
- [151] Paolo Romano, Rosalba Giugno, and Alfredo Pulvirenti. Tools and collaborative environments for bioinformatics research. *Briefings in bioinformatics*, 12(6):549–561, 2011.
- [152] Anthony Rowe, Dimitrios Kalaitzopoulos, Michelle Osmond, Moustafa Ghanem, and Yike Guo. The discovery net system for high throughput bioinformatics. *Bioinformatics*, 19(suppl_1):i225–i231, 2003.
- [153] Banani Roy, Amit Kumar Mondal, Kawser Wazed, Chanchal K. Roy, and Kevin A. Schneider. Towards a reference architecture for cloud-based plant genotyping and phenotyping analysis frameworks. In *Proc. of International Conference on Software Architecture*, 2017, accepted.
- [154] Chanchal K Roy. Detection and analysis of near-miss software clones. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 447–450. IEEE, 2009.
- [155] Chanchal K. Roy and James R. Cordy. A survey on software clone detection research. *Queen’s School of Computing TR*, 541(115):64–68, 2007.
- [156] Chanchal K. Roy and James R. Cordy. An empirical study of function clones in open source software. In *Reverse Engineering, 2008. WCRE’08. 15th Working Conference on*, pages 81–90. IEEE, 2008.
- [157] Chanchal K Roy and James R Cordy. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 172–181. IEEE, 2008.

- [158] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K Roy, and Cristina V Lopes. Sourcerercc: Scaling code clone detection to big-code. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 1157–1168. IEEE, 2016.
- [159] André Schaaff, L Verdes-Montenegro, J Ruiz, and J Santander Vela. Scientific workflows in astronomy. *Proceeding of Astronomical Data Analysis Software and Systems*, 2012.
- [160] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85. ACM, 2003.
- [161] Ricky J Sethi and Yolanda Gil. Scientific workflows in data analysis: Bridging expertise across multiple domains. *Future Generation Computer Systems*, 75:256–270, 2017.
- [162] Ricky J Sethi, Hyunjoon Jo, and Yolanda Gil. Re-using workflow fragments across multiple data domains. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 90–99. IEEE, 2012.
- [163] Adam C. Siepel, Andrew N. Tolopko, Andrew D. Farmer, Peter A. Steadman, Faye D. Schilkey, B. Dawn Perry, and William D. Beavis. An integration platform for heterogeneous bioinformatics software components. *IBM Systems Journal*, 40(2):570–591, 2001.
- [164] Gergely Sipos. Protecting the consistency of workflow applications in collaborative development environments. *Future Generation Computer Systems*, 28(3):500–512, 2012.
- [165] Gergely Sipos and Péter Kacsuk. Maintaining consistency properties of grid workflows in collaborative editing systems. In *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*, pages 168–175. IEEE, 2009.
- [166] Gergely Sipos and Péter Kacsuk. Efficient graph partitioning algorithms for collaborative grid workflow developer environments. *Euro-Par 2010-Parallel Processing*, pages 50–61, 2010.
- [167] Gergely Sipos and Peter K Kacsuk. Collaborative workflow editing in the p-grade portal. 2005.
- [168] Gergely Sipos, Gareth Lewis, Péter Kacsuk, and Vassil Alexandrov. Workflow-oriented collaborative grid portals. *Advances in Grid Computing-EGC 2005*, pages 64–69, 2005.
- [169] Jung Soh, Xiaoli Dong, Sean M. Caffrey, Gerrit Voordouw, and Christoph W. Sensen. Phoenix 2: A locally installable large-scale 16s rrna gene sequence analysis pipeline with web interface. *Journal of Biotechnology*, 167(4):393 – 403, 2013.
- [170] Apache Spark. Apache Spark Lightning-fast cluster computing. <https://spark.apache.org/>.
- [171] Vipin T Sreedharan, Sebastian J Schultheiss, Géraldine Jean, André Kahles, Regina Bohnert, Philipp Drewe, Pramod Mudrakarta, Nico Görnitz, Georg Zeller, and Gunnar Rätsch. Oqtans: the rna-seq workbench in the cloud for complete and reproducible quantitative transcriptome analysis. *Bioinformatics*, 30(9):1300–1301, 2014.
- [172] Chengzheng Sun. Optional and responsive fine-grain locking in internet-based collaborative systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):994–1008, 2002.
- [173] Chengzheng Sun and David Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 9(1):1–41, 2002.
- [174] David Sun and Chengzheng Sun. Operation context and context-based operational transformation. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 279–288. ACM, 2006.
- [175] Jeff Thomas Svajlenko et al. *Large-Scale Clone Detection and Benchmarking*. PhD thesis, University of Saskatchewan, 2018.

- [176] Jeffrey Svajlenko and Chanchal K Roy. Efficiently measuring an accurate and generalized clone detection precision using clone clustering. In *SEKE*, pages 426–433, 2016.
- [177] Robert Tairas and Jeff Gray. Phoenix-based clone detection using suffix trees. In *Proceedings of the 44th annual Southeast regional conference*, pages 679–684. ACM, 2006.
- [178] Robert Tairas and Jeff Gray. An information retrieval process to aid in the analysis of code clones. *Empirical Software Engineering*, 14(1):33–56, 2009.
- [179] Yla R Tausczik, Aniket Kittur, and Robert E Kraut. Collaborative problem solving: A study of mathoverflow. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 355–367. ACM, 2014.
- [180] Francis Eng Hock Tay and Avijit Roy. Cybercad: a collaborative approach in 3d-cad technology in a multimedia-supported environment. *Computers in Industry*, 52(2):127–145, 2003.
- [181] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The triana workflow environment: Architecture and applications. *Workflows for e-Science*, pages 320–339, 2007.
- [182] David Tilman, Christian Balzer, Jason Hill, and Belinda L Befort. Global food demand and the sustainable intensification of agriculture. *Proceedings of the National Academy of Sciences*, 108(50):20260–20264, 2011.
- [183] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.
- [184] Qiang Tu et al. Evolution in open source software: A case study. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 131–142. IEEE, 2000.
- [185] Qiang Tu and Michael W Godfrey. An integrated approach for studying architectural evolution. In *Program Comprehension, 2002. Proceedings. 10th International Workshop on*, pages 127–136. IEEE, 2002.
- [186] useGalaxy. An open source, web-based platform for data intensive biomedical research. <https://usegalaxy.org/>.
- [187] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. Yawl: yet another workflow language. *Information systems*, 30(4):245–275, 2005.
- [188] Filip Van Rysselberghe and Serge Demeyer. Reconstruction of successful software evolution using clone detection. In *null*, page 126. IEEE, 2003.
- [189] Kunal Vyas, Zeshi Zheng, and Lin Li. Titanic-machine learning from disaster. *Machine Learning Final Project, UMass Lowell*, pages 1–7, 2015.
- [190] Andrew Walenstein and Arun Lakhotia. The software similarity problem in malware analysis. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.
- [191] Christopher Walton and A Barker. An agent-based e-science experiment builder. In *Proceedings of the 1st International Workshop on Semantic Intelligent Middleware for the Web and the Grid*, pages 247–264, 2004.
- [192] Weka. Open Source Machine Learning Tools Collection. <https://www.cs.waikato.ac.nz/ml/weka/>.
- [193] James A Whittaker and Michael G Thomason. A markov chain model for statistical software testing. *IEEE Transactions on Software engineering*, 20(10):812–824, 1994.
- [194] James Wilsdon et al. *Knowledge, networks and nations: Global scientific collaboration in the 21st century*. The Royal Society, 2011.

- [195] Stefan Wuchty, Benjamin F Jones, and Brian Uzzi. The increasing dominance of teams in production of knowledge. *Science*, 316(5827):1036–1039, 2007.
- [196] Maik Wurdel, Daniel Sinnig, and Peter Forbrig. Ctml: Domain and task modeling for collaborative environments. *J. UCS*, 14(19):3188–3201, 2008.
- [197] Jiachen Yang, Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki, and Shinji Kusumoto. Classification model for code clones based on machine learning. *Empirical Software Engineering*, 20(4):1095–1125, 2015.
- [198] Yunwen Ye, Yasuhiro Yamamoto, and Kumiyo Nakakoji. A socio-technical framework for supporting programmers. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 351–360. ACM, 2007.
- [199] Jia Zhang. Co-taverna: a tool supporting collaborative scientific workflows. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 41–48. IEEE, 2010.
- [200] Jia Zhang, Qihao Bao, Xiaoyi Duan, Shiyong Lu, Lijun Xue, Runyu Shi, and Pingbo Tang. Collaborative scientific workflow composition as a service: An infrastructure supporting collaborative data analytics workflow design and management. In *Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on*, pages 219–228. IEEE, 2016.
- [201] Jia Zhang, Daniel Kuc, and Shiyong Lu. Confucius: A tool supporting collaborative scientific workflow composition. *IEEE Transactions on Services Computing*, 7(1):2–17, 2014.
- [202] Haibin Zhu and MengChu Zhou. Role-based collaboration and its kernel mechanisms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 36(4):578–589, 2006.

APPENDIX A

CONSISTENCY MANAGEMENT SIMULATION STUDY

A.1 Code Snippets

```
1 //node construct
2 function Node(data) {
3   this.data = data;
4   this.parent = null;
5   this.isLocked = false;
6   this.currentOwner = "NONE";
7   this.children = [];
8 }
9
10 //tree construct
11 function Tree(data) {
12   var node = new Node(data);
13   this._root = node;
14 }
15
16 //traverse the tree by df default starting from the root of the tree
17 Tree.prototype.traverseDF = function(callback) {
18
19   // this is a recurse and immediately-invoking function
20   (function recurse(currentNode) {
21     // step 2
22     for (var i = 0, length = currentNode.children.length; i < length; i++) {
23       // step 3
24       recurse(currentNode.children[i]);
25     }
26
27     // step 4
28     callback(currentNode);
29
30     // step 1
31   })(this._root);
32
33 };
34
35 //traverse by depth first search from a specified start node (parent)
36 Tree.prototype.traverseDF_FromNode = function(startNode, callback) {
37
38   // this is a recurse and immediately-invoking function
39   (function recurse(currentNode) {
40     // step 2
41     for (var i = 0, length = currentNode.children.length; i < length; i++) {
42       // step 3
43       recurse(currentNode.children[i]);
44     }
45
46     // step 4
47     callback(currentNode);
48
49     // step 1
50   })(startNode);
51
52 };
53
54 //scans through all the nodes of the tree
55 Tree.prototype.contains = function(callback, traversal) {
56   traversal.call(this, callback);
57 }
```

```

58 };
59
60 //add a new node to a specific parent of the tree
61 Tree.prototype.add = function(data, toData, traversal) {
62     var child = new Node(data),
63         parent = null,
64         callback = function(node) {
65             if (node.data === toData) {
66                 parent = node;
67             }
68         };
69
70     this.contains(callback, traversal);
71
72     if (parent) {
73         parent.children.push(child);
74         child.parent = parent;
75     } else {
76         throw new Error('Cannot add node to a non-existent parent.');
```

```

126     throw new Error('The node did not have any previous parent!');
127   }
128 };
129 };
130
131 //removes a particular node from its parent.
132 Tree.prototype.remove = function(data, fromData, traversal) {
133   var tree = this,
134       parent = null,
135       childToRemove = null,
136       index;
137
138   var callback = function(node) {
139     if (node.data === fromData) {
140       parent = node;
141     }
142   };
143
144   this.contains(callback, traversal);
145
146   if (parent) {
147     index = findIndex(parent.children, data);
148
149     if (index === undefined) {
150       throw new Error('Node to remove does not exist.');
```

```

194
195     return nodeFloorAvailability;
196 }
197 }
198
199 //someone has got the access to this node, so lock it and all its descendants
200 Tree.prototype.lockThisNodeAndDescendants = function(newOwner, nodeData, traversal) {
201     var theNode = this.getNode(nodeData, traversal);
202     this.traverseDF.FromNode(theNode, function(node) {
203         //use helper function to load this node for the corresponding user
204         lockNode(node, newOwner);
205     });
206 }
207
208 //someone has released the access to this node, so UNLOCK it and all its descendants
209 Tree.prototype.unlockThisNodeAndDescendants = function(nodeData, traversal) {
210     var theNode = this.getNode(nodeData, traversal);
211     this.traverseDF.FromNode(theNode, function(node) {
212         //use the helper function to unlock the node.
213         unlockNode(node);
214     });
215 }
216
217
218 //HELPER FUNCTION: child index
219 function findIndex(arr, data) {
220     var index;
221
222     for (var i = 0; i < arr.length; i++) {
223         if (arr[i].data === data) {
224             index = i;
225         }
226     }
227
228     return index;
229 }
230
231 //HELPER FUNCTION: lock a given node with corresponding owner name
232 function lockNode(node, nodeOwner) {
233     node.isLocked = true;
234     node.currentOwner = nodeOwner;
235 }
236
237 //HELPER FUNCTION: unlock a node
238 function unlockNode(node) {
239     node.isLocked = false;
240     node.currentOwner = "NONE";
241 }

```

Listing A.1: Log Snippet as Recorded for a User-Study Session.

APPENDIX B

USER STUDY: USER MANUAL AND STUDY DESIGN

B.1 SciWorCS

SciWorCS is a Collaborative **Scientific Workflow** management **System**. SciWorCS follows a plugin-based architecture for the scientific computational modules. It provides collaborative environment for scientific data analysis using efficient attribute level locking scheme. In addition to the collaborative data analysis, SciWorCS also provides real-time monitoring of the computation steps.

B.2 Introduction to SciWorCS Editor

Fig. B.1, shows a screenshot of the collaborative scientific workflow composition system. Besides, logging in to a SciWorCS¹ cloud instance would be better to follow along the descriptions.

The panel labeled as ‘A’, contains all the workflow components such as, *Toolbox* (i.e., set of workflow modules), *Saved Workflows* and *Shared Workflows* with other users. The set of modules are classified in different Toolbox based on the general data analysis purposes of the computational modules. Some examples of such Toolboxes are: *Bioinformatics*, *Machine Learning*, *Source Analysis* and so on as illustrated in the figure.

The collaborative composition of the workflow is done on panel ‘B’. For the intended data analysis task, the required modules are selected (e.g., via *Left-Mouse Click*) from the corresponding Toolbox to appear in the composition panel. The selected modules are then connected together (via., *Left-Mouse Click & Drag*) on the corresponding input/output ports for defining the datalink relation among the modules. Please note that, the linking requires a matching data types (e.g., a *.txt* output port connects only with another *.txt* input port) between the input/output ports. The modules can be configured with corresponding attributes from panel ‘C’. Panel ‘D’ shows a list of collaborators and their current online/offline status. The list of the workflow outputs are shown in panel ‘E’. New dataset can be browsed and uploaded to the central server for analysis from the panel ‘F’.

B.3 SciWorCS Workflow Composition

B.3.1 Prerequisites

Basic understanding of SciWorCS editor and locking schemes for collaborative workflow composition. Compose the following reference workflow with the modules available in ‘Galaxy Modules’ toolbox. The workflow need to compose using both of the locking schemes separately - locking scheme one² and locking scheme two³. Each participants are required to add at least one of the computational modules to the workflow.

B.3.2 Task Description

This task is about collabortive composition of scientific workflows. The reference of the workflows to compose are taken from *myExperiment* - a shared social space for scientific artifacts sharing among researchers. Collaborators need to compose the workflows using two different locking schemes.

At the end of the study, **SAVE the generated console logs** as *participant_yourID_task_1a.log* and *participant_yourID_task_1b.log* for the two systems respectively.

¹Example SciWorCS Cloud Instance:: sample cloud instance

²Link for Locking Scheme 1

³Link for Locking Scheme 2

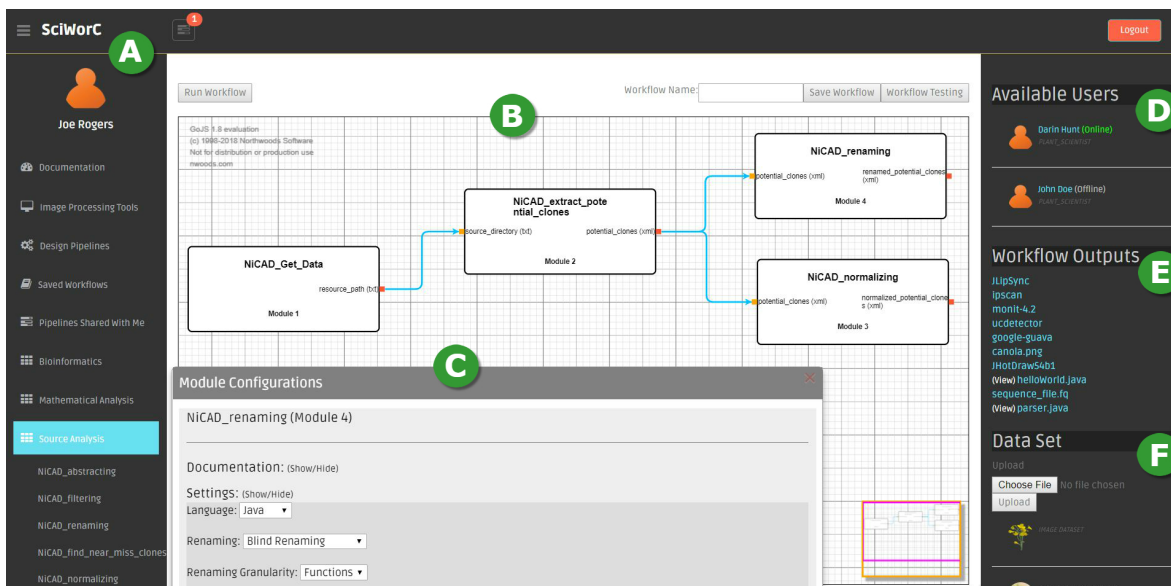


Figure B.1: SciWorCS Editor for Collaborative Scientific Workflow Composition

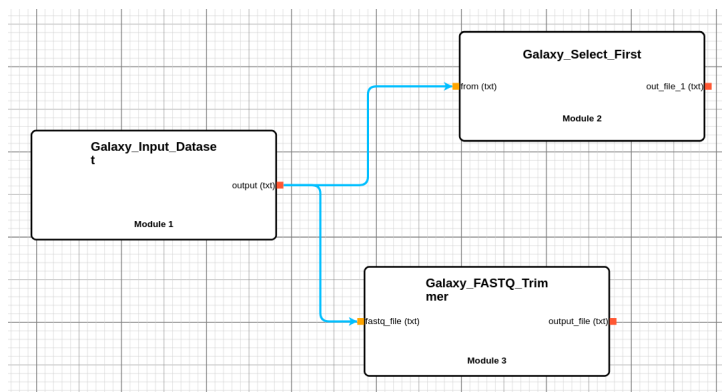


Figure B.2: Sample workflow for collaborative composition practice

Workflows for *system1* locking scheme

Fig. 1.2 and Fig. 1.3 are the two reference workflows to compose using *system1* locking scheme. At the end of *system1* locking scheme, please, **ANSWER NASA-TLX** questionnaires with your participant id and Task id as **1a** here.

Workflows for *system2* locking scheme

Fig. 1.4 and Fig. 1.5 are the two reference workflows to compose using *system1* locking scheme. At the end of *system1* locking scheme, please, **ANSWER NASA-TLX** questionnaires with your participant id and Task id as **1b** here.

B.3.3 Toolbox

The required tools are available in the toolbox named as *Galaxy Modules*. The modules are prefixed with *Galaxy*. There are 19 such computational modules in the toolbox for this task.

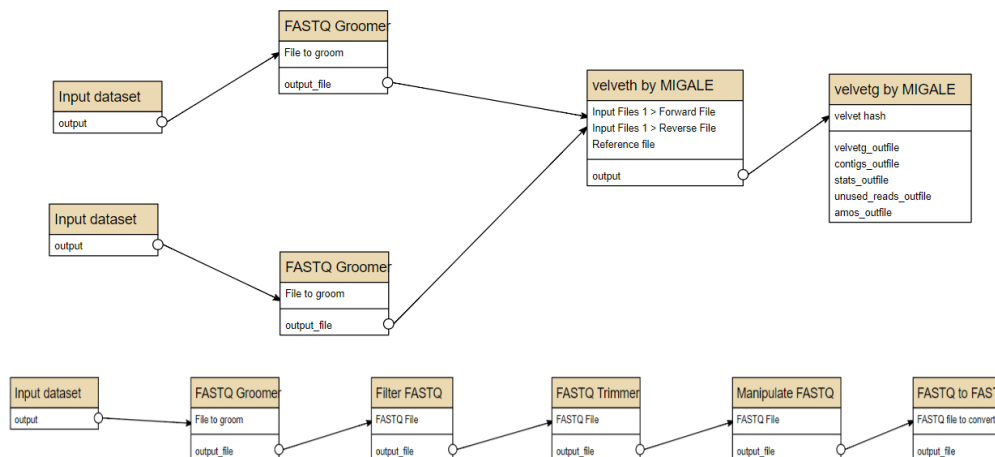


Figure B.3: Paired-end reads assembly after FastQ groomer using a Migale modified version of Velvet tool.

B.4 Collaborative Problem Solving

B.4.1 Prerequisites

Basic understanding of executing and monitoring the workflow. Using the ‘Mathematical Analysis’ Toolbox collaboratively develop a workflow that adds two given numbers and returns the obtained result.

B.4.2 Task Description

For the problem solving tasks, we used simple mathematical modules which are responsible for elementary arithmetic operations, such as *Addition*, *Subtraction*, *Division*, *Multiplication* and *Power*. The integrated mathematical modules accept one or more numeric values and outputs a single resultant value as corresponding data files. As tasks the collaborators get a set of sample numeric input and output to predict and design the corresponding workflow that generates the similar output as sample dataset. Note that, these five simple mathematical modules can be used multiple times with complex datalink relations among them to yet solve much complicated numeric patterns. However, the prediction of such complex workflows can be often be non-trivial making the solution ambiguous. Hence, in order to keep the problem simpler, we limit the required design of the mathematical workflow to a simpler and fixed structure. For this study the used arithmetic equation structure is, $z = Ax^C \mathcal{O} By^D$, where x, y are sample inputs, z is the corresponding sample output, \mathcal{O} is any operator from the available mathematical modules and A, B, C, D are some integer constants. For a given of sets of sample inputs and outputs (i.e., x, y and z), participants need to predict the constants, A, B, C, D and the operator, \mathcal{O} . Hence, the designed workflow requires maximum seven mathematical modules -two for taking inputs (i.e., x and y), two for multiplications (i.e., Ax and Bx), two for powers (i.e., x^C and y^D) and one for unknown mathematical operator, \mathcal{O} . Please see Fig 2.1 for an example reference workflow of the task.

Note that, some of the modules can be omitted given the calculated corresponding constant value is 1. For example, if $A = 1$ then the multiplication module can be omitted. In that case, the workflow can have a different structure than the provided reference structure.

At the end of the study, **SAVE the generated console logs** as *participant_yourID_task_2.log*. Also please, **ANSWER NASA-TLX** questionnaires with your participant id and Task id as **2** here.

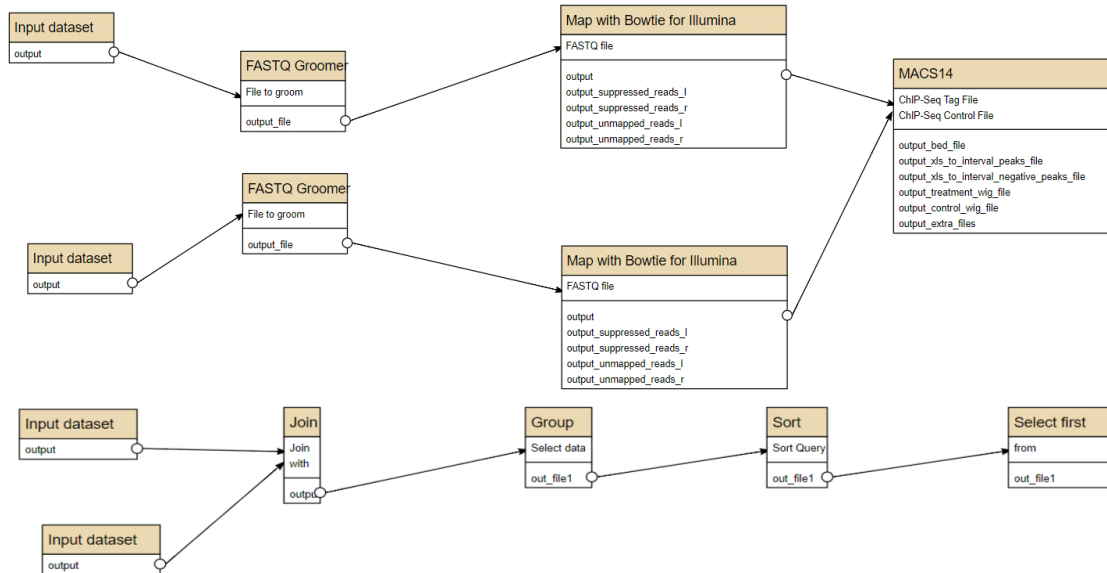


Figure B.4: Workflow used when applying the CPB2012 Basic Protocol 3; Peaks for ChIP-seq data using MACS14.

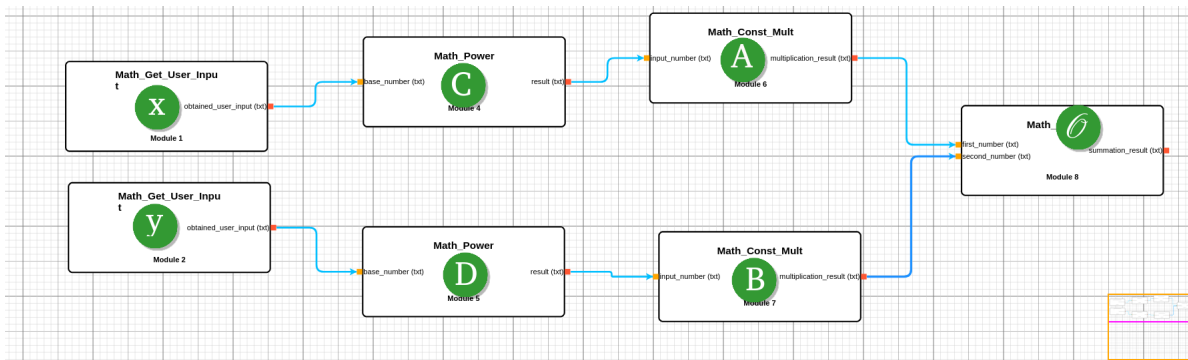


Figure B.5: Reference Workflow for the Task

Sample Input Output for the Task

Participants need to collaboratively solve and build the required workflows for the sample input output in Table 2.2, 2.3 and 2.4. Participants collaboratively work towards the building of the three corresponding workflows. The target is maximizing the number of solved problems within a time range of 10 minutes.

B.4.3 Toolbox

The available modules for the task are as follows:

1. Math.Get_User.Input: Gets the user input (i.e., x and y).
2. Math.Const.Mult: Multiplies a variable by a constant (i.e., A in Ax).
3. Math.Power: Raise a power on a given variable (i.e., C in x^C).
4. Arithmetic Operators: Applies mathematical operator on two given variable and returns resultant. Available mathematical operators are: Addition, Subtraction, Multiplication, Division.

Table B.1: First Math Task (1 of 2)

x	y	z
1	1	2
2	2	8
3	3	18
4	4	32

Table B.2: Second Math Task (2 of 2)

x	y	z
4	6	26
3	4	18
4	2	14
9	2	24

B.5 Collaborative Data Analysis using Machine Learning Classifiers and Statistical Tools

B.5.1 Task Description

We select a publicly available dataset - ‘*Titanic: Machine Learning from Disaster*’ for the machine learning based classification task. Table B.3, present the dataset definition as collected from *Kaggle* - a community of data scientists and machine learners. The dataset is about the *RMS Titanic* passengers, where the task is to build a machine learning model to predict which sort of people were more likely to survive the sinking of Titanic. The machine learning model for the binary classification (i.e., whether *Survived* or not) is to be build by using features, such as ticket class, age, sex and so on of the corresponding passengers. Note that, there exists multiple solution paths and strategies in selecting the feature set, selecting the ‘correct’ machine learning model, configuring the selected machine learning model for training and so on. Collaborators need to iteratively converge on plans (i.e., feature set selection, machine model selection and so on) via discussion, statistical analysis of the dataset and reviewing the obtained performance of the trained model. Collaborators’ discussion process is assisted via different statistical computational modules and collaborative visualization tools of SciWorCS. For example, for the given classification task visualizing the data distribution for survival in terms of age, sex, ticket class and so on can be important for selecting the contributing features or the machine learning model.

At the end of the study, **SAVE the generated console logs** as *participant_yourID_task_3.log*. Also please, **ANSWER NASA-TLX** questionnaires with your participant id and Task id as **3** here.

B.5.2 Dataset

The supplied dataset for the study contains information of 892 passengers of Titanic in *.csv* format, with the feature set as specified in Table B.3. The dataset can be viewed from SciWorCS. Note that, like any other data analysis task, the provided dataset requires pre-processing for handling missing data (such as, missing Age information of some passengers and so on), transforming to categorical features (such as, Sex can be represented as 0/1 instead of textual - ‘male’/‘female’).

B.5.3 Toolbox

The available modules for the task are as follows:

1. `Stats.Load.Dataset`: Selects and loads the available dataset for the analysis.
2. `Stats.MissingValues`: Calculates and shows the statistics of missing values in a given dataset in *.txt* format.

Table B.3: Publicly Available Dataset - ‘Titanic: Machine Learning from Disaster’ as Collected from *Kaggle*.

Variable	Definition
Survived	Survival of the Passenger (0 = No, 1 = Yes)
Pclass	Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
Name	Name of the Passenger
Sex	Sex
Age	Age in years
Sibsp	Number of siblings/spouses aboard
Parch	Number of parents/children aboard
Ticket	Ticket number
Fare	Passenger fare
Cabin	Cabin number
Embarked	Port of Embarkation

3. *Stats.FillMissingByMedian*: The missing values are filled with median value of the available values. For example, the missing Age are filled median values of the available Age of other passengers.
4. *Stats.Feature.Distribution*: Graph plots the selected features based on class labels. Note that, the selected feature should **NOT** contain any missing values (such as Age). To handle such missing values *Stats.FillMissingByMedian* module might be used prior to visualizing the feature distribution.
5. *Stats.FeatureCategories.ByLabel*: Graph plots the feature categories by label. For example, how many of the survived passengers were male or female (e.g., dataset categorization by Sex).
6. *Stats.CreateCategoricalVariable*: Creates numerical categorical variables. For example, the Sex field contains either ‘male’ or ‘female’. The categorization creates two additional features namely - ‘Sex_male’ and ‘Sex_female’ and filled with numeric 0/1 accordingly.
7. *Stats.Drop_Variable*: Drops the selected variable which is not used further for classification.
8. *Machine Learning Models*: There are five machine learning based classification models in the Tool-box. The classification models are: *Logistic Regression*, *SVM*, *kNN*, *Random Forest* and *Naive Bayes* classifiers.

B.6 NASA-TLX Questionnaires

The NASA-TLX Questionnaires [75]—as presented to the participants at the end of every sessions of the user study. Participants were asked to provide their responses for the questionnaires within a range of 1 to 10 (e.g., 1: Very Low and 10: Very High):

1. How mentally demanding was the task?
2. How much physical activity was required (e.g. pushing, pulling, turning, controlling, activating, etc)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?
3. How much time pressure did you feel due to the rate of pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?
4. How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)?
5. How hard did you have to work to accomplish your level of performance?
6. How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, relaxed and complacent did you feel during the task?

B.7 Log Snippet

Listing B.1 is a Log snippet as recorded for a user study sessions. An entry of the log is generated for a corresponding event by a participant during a user-study session. The stored log contains different information, such as *Timestamp* of the event, *Source Script* responsible for the event trigger, *User ID*, *Event Type* and *Log Information* -representing the details information of the generated event. The recorded logs were then parsed for different analysis.

```
1 TIMESTAMP SOURCE_SCRIPT USER_ID EVENT_TYPE LOG_INFO
2 15:35:33.994 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:Hi this time you can do the first seven module
3 15:35:43.120 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:I can do the rest
4 15:37:23.158 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:Yeah sure
5 15:37:54.191 telepointer_module_lockingGO.js:2801 john@gmail.com =>MODULE_ADDED=>module_id:2
6 15:37:56.588 telepointer_module_lockingGO.js:1108 john@gmail.com =>MODULE_MOVED=>key:
   module_id_2*x:-116*y:-172
7 15:38:04.216 telepointer_module_lockingGO.js:2801 john@gmail.com =>MODULE_ADDED=>module_id:4
8 15:38:06.898 telepointer_module_lockingGO.js:1108 john@gmail.com =>MODULE_MOVED=>key:
   module_id_4*x:588*y:-171
9 15:38:28.663 telepointer_module_lockingGO.js:2801 john@gmail.com =>MODULE_ADDED=>module_id:6
10 15:38:30.831 telepointer_module_lockingGO.js:1108 john@gmail.com =>MODULE_MOVED=>key:
   module_id_6*x:-95*y:45
11 15:39:45.478 telepointer_module_lockingGO.js:2801 john@gmail.com =>MODULE_ADDED=>module_id:8
12 15:39:47.968 telepointer_module_lockingGO.js:1108 john@gmail.com =>MODULE_MOVED=>key:
   module_id_8*x:-492*y:202
13 15:40:09.228 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:from scratch?
14 15:40:37.564 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:yeah makes sense
15 15:40:43.619 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:lets start over
16 15:41:39.655 telepointer_module_lockingGO.js:1028 john@gmail.com =>MODULE_DELETED=>module_id
   :module_id_8
17 15:41:40.469 telepointer_module_lockingGO.js:1028 john@gmail.com =>MODULE_DELETED=>module_id
   :module_id_6
18 15:41:41.349 telepointer_module_lockingGO.js:1028 john@gmail.com =>MODULE_DELETED=>module_id
   :module_id_2
19 15:41:43.093 telepointer_module_lockingGO.js:1028 john@gmail.com =>MODULE_DELETED=>module_id
   :module_id_4
20 15:43:46.225 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:are you going the whole thing?
21 15:44:18.681 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:dont do the last one
22 15:44:33.104 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:I know how to use your module
23 15:45:28.720 telepointer_module_lockingGO.js:3126 john@gmail.com =>P2P_CHAT_SENT=>to:
   darin_gmail_com*text:okay
24 15:45:55.435 telepointer_module_lockingGO.js:1131 john@gmail.com =>
   SUB_WORKFLOW_LOCK_REQUESTED=>rootNode:module_id_15
25 15:46:12.826 telepointer_module_lockingGO.js:1131 john@gmail.com =>
   SUB_WORKFLOW_LOCK_REQUESTED=>rootNode:module_id_18
26 15:46:24.682 telepointer_module_lockingGO.js:1131 john@gmail.com =>
   SUB_WORKFLOW_LOCK_REQUESTED=>rootNode:module_id_19
27 15:46:28.130 telepointer_module_lockingGO.js:1131 john@gmail.com =>
   SUB_WORKFLOW_LOCK_REQUESTED=>rootNode:module_id_20
28 15:46:36.889 telepointer_module_lockingGO.js:2688 john@gmail.com =>SUB_WORKFLOW_LOG_GRANTED
   =>rootNode:module_id_18
29 15:46:38.526 telepointer_module_lockingGO.js:2688 john@gmail.com =>SUB_WORKFLOW_LOG_GRANTED
   =>rootNode:module_id_19
30 15:46:45.872 telepointer_module_lockingGO.js:2688 john@gmail.com =>SUB_WORKFLOW_LOG_GRANTED
   =>rootNode:module_id_20
31 15:47:20.661 telepointer_module_lockingGO.js:2688 john@gmail.com =>SUB_WORKFLOW_LOG_GRANTED
   =>rootNode:module_id_15
32 15:47:31.030 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
   moduleID:module_id_18
```

```

33 15:47:31.030 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_15*frompid: summation_result.txt*to: module_id_18*topid: base_number.txt
34 15:47:54.459 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
    moduleID: module_id_19
35 15:47:54.460 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_18*frompid: result.txt*to: module_id_19*topid: first_number.txt
36 15:48:46.282 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
    moduleID: module_id_19
37 15:48:46.283 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_16*frompid: multiplication_result.txt*to: module_id_19*topid: second_number.txt
38 15:49:06.816 telepointer_module_lockingGO.js:2688 john@gmail.com =>SUB_WORKFLOW_LOG_GRANTED
=>rootNode: module_id_21
39 15:49:06.816 telepointer_module_lockingGO.js:1131 john@gmail.com =>
    SUB_WORKFLOW_LOCK_REQUESTED=>rootNode: module_id_21
40 15:49:18.330 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
    moduleID: module_id_21
41 15:49:18.330 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_19*frompid: summation_result.txt*to: module_id_21*topid: first_number.txt
42 15:49:23.538 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
    moduleID: module_id_21
43 15:49:23.538 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_20*frompid: summation_result.txt*to: module_id_21*topid: second_number.txt
44 15:49:28.639 telepointer_module_lockingGO.js:2688 john@gmail.com =>SUB_WORKFLOW_LOG_GRANTED
=>rootNode: module_id_22
45 15:49:28.640 telepointer_module_lockingGO.js:1131 john@gmail.com =>
    SUB_WORKFLOW_LOCK_REQUESTED=>rootNode: module_id_22
46 15:49:31.331 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
    moduleID: module_id_22
47 15:49:31.332 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_21*frompid: subtraction_result.txt*to: module_id_22*topid: base_number.txt
48 15:49:40.564 telepointer_module_lockingGO.js:2801 john@gmail.com =>MODULE_ADDED=>module_id
:25
49 15:49:45.695 telepointer_module_lockingGO.js:1108 john@gmail.com =>MODULE_MOVED=>key :
    module_id_25*x:867.8714625000002*y:587.0895187500003
50 15:50:00.073 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
    moduleID: module_id_25
51 15:50:00.074 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_24*frompid: subtraction_result.txt*to: module_id_25*topid: second_number.txt
52 15:50:04.401 telepointer_module_lockingGO.js:1767 john@gmail.com =>MODULE_CONFIG_CHANGE=>
    moduleID: module_id_25
53 15:50:04.401 telepointer_module_lockingGO.js:1089 john@gmail.com =>DATALINK_ADDED=>from :
    module_id_22*frompid: result.txt*to: module_id_25*topid: first_number.txt
54 ...
55 ...
56 ...
57 ...

```

Listing B.1: Log Snippet as Recorded for a User-Study Session.