Workflow Provenance: from Modeling to Reporting

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
Rayhan Ferdous

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

Workflow provenance is a crucial part of a workflow system as it enables data lineage analysis, error tracking, workflow monitoring, usage pattern discovery, and so on. Integrating provenance into a workflow system or modifying a workflow system to capture or analyze different provenance information is burdensome, requiring extensive development because provenance mechanisms rely heavily on the modelling, architecture, and design of the workflow system. Various tools and technologies exist for logging events in a software system. Unfortunately, logging tools and technologies are not designed for capturing and analyzing provenance information. Workflow provenance is not only about logging, but also about retrieving workflow related information from logs. In this work, we propose a taxonomy of provenance questions and guided by these questions, we created a workflow programming model 'ProvMod' with a supporting run-time library to provide automated provenance and log analysis for any workflow system. The design and provenance mechanism of ProvMod is based on recommendations from prominent research and is easy to integrate into any workflow system. ProvMod offers Neo4j graph database support to manage semi-structured heterogeneous JSON logs. The log structure is adaptable to any NoSQL technology. For each provenance question in our taxonomy, ProvMod provides the answer with data visualization using Neo4j and the ELK Stack. Besides analyzing performance from various angles, we demonstrate the ease of integration by integrating ProvMod with Apache Taverna and evaluate ProvMod usability by engaging users. Finally, we present two Software Engineering research cases (clone detection and architecture extraction) where our proposed model ProvMod and provenance questions taxonomy can be applied to discover meaningful insights.

Acknowledgements

First of all, I want to express my deep gratitude to my supervisors Dr. Chanchal K. Roy and Dr. Kevin A. Schneider for their research guidelines, direction, advice and above all for their inspiration, encouragement and continuous patience. The research work would not have been possible without their support.

I want to thank Dr. Banani Roy, for her direction at every single step of the work from the beginning. Being a newbie in the research domain, it would not be possible to frame the research work in a standard manner without her.

I want to thank all the Software Research Lab members for their cooperation, participation in the insightful discussions that gave me an opportunity to grow. In particular, I would like to thank Golam Mostaeen and Mahjabin Alam for their collaborative support.

I would like to thank all the summer research interns from the last two years and all the graduate students from other labs who took part in the user study as well as provided any collaborative support.

I am grateful to the Department of Computer Science at the University of Saskatchewan for providing me with financial support. I am thankful to any staff at the department for helping me to get rid of any formal procedure with ease, specifically Gwen Lancaster, Findlay Sophie, Heather Webb and Shakiba Jalal.

I am thankful to my parents Ferdous Alam and Rozina Parveen from the very deep of my heart, who provided me with inspirational and mental support from the very distance. Without their patience, I would not be able to solely focus on my work. My younger brother Arman Ferdous, whom I love the most after my parents, brings me meaning and comfort in my life. His presence and achievements also bring me happiness.

I also want to thank any of my friends whoever motivated me from abroad personally to continue my research and keep focus.

Finally, I want to express my special gratitude to Md Zahidul Islam, faculty of Computer Science and Engineering at Khulna University, Bangladesh who was my supervisor during my B.Sc. studies and always taught me how to focus on quality but not quantity. My research journey truly began by holding his hands. He is always a true inspiration in my life!

I want to dedicate this thesis to those creative people, who hold the courage to dream big, think different and pursue their dream with patience!

Contents

Permission to Use i

Abstract ii

Acknowledgements iii

Contents v

List of Tables viii

List of Figures ix

1 Introduction 1
- 1.1 Contribution . . . 3

2 Preliminaries 6
- 2.1 Terms and Definitions . . . 6
  - 2.1.1 Programming Model . . . 6
  - 2.1.2 Provenance . . . 6
  - 2.1.3 Workflow . . . 6
  - 2.1.4 Workflow System . . . 6
  - 2.1.5 Module . . . 6
  - 2.1.6 Data Item . . . 7
  - 2.1.7 Reporting . . . 7
- 2.2 Importance of Provenance . . . 7
- 2.3 Important Features of Provenance . . . 8

3 ProvMod: The Model 10
- 3.1 Design Goals . . . 10
- 3.2 Definitions . . . 11
  - 3.2.1 **Workflow W** . . . 11
  - 3.2.2 **Data D** . . . 11
  - 3.2.3 **Module M** . . . 11
  - 3.2.4 **Properties P** . . . 11
  - 3.2.5 **Type** . . . 11
  - 3.2.6 **Dataflow** $\rightarrow$ . . . 11
  - 3.2.7 **Module Invocation** . . . 11
- 3.3 ProvMod, Graph and Graph Database . . . 13
- 3.4 Querying . . . 14
  - 3.4.1 **Decide: The First Elementary Query** . . . 14
  - 3.4.2 **Extension of the First Elementary Query** . . . 14
  - 3.4.3 **Module Chain Regeneration with Decide** . . . 14
  - 3.4.4 **Property Mapping with Decide** . . . 14
  - 3.4.5 **Time Sequence Mapping: The Second Elementary Query** . . . 15
  - 3.4.6 **Data Sequence Mapping: The Third Elementary Query** . . . 15

4 Classifying Provenance Questions 16
- 4.1 Methodology of Deriving Taxonomy . . . 16
- 4.2 The Provenance Questions Taxonomy . . . 17
- 4.3 Graph Related Questions . . . 18

4.4 Monitoring Related Questions . . . 20
4.4.1 Real-time Aggregation . . . 20
4.4.2 Fixed-time Aggregation . . . 21
4.5 Statistical Analysis Related Questions . . . 21
4.5.1 Statistical Analysis Questions . . . 22

5 Implementation **23**
5.1 System Architecture . . . 23
5.2 Logging Mechanism . . . 23
5.2.1 Taxonomy of Workflow Systems by Cruz et al. . . . 25
5.2.2 Description of ProvMod Provenance Mechanism . . . 27
5.3 Workflow Components . . . 27
5.4 Logging Levels . . . 29
5.5 Services . . . 30
5.6 Log Schema . . . 32
5.7 Workflow Implementation . . . 32
5.8 Tool Development and Integration . . . 33

6 Experimental Analysis **37**
6.1 Sparse Workflow . . . 39
6.2 Large Workflow . . . 41
6.3 Deep Workflow . . . 43
6.4 Performance Analysis . . . 45

7 Reporting **50**
7.1 Visualization for Graph Related Questions . . . 50
7.2 Visualization for Time Series Monitoring Questions . . . 51
7.3 Visualization for Monitoring Metrics . . . 53
7.4 Visualizing Proportional Measurements . . . 53
7.5 Visualizing Nominal vs Nominal to Ordinal Monitoring . . . 54
7.6 Visualizing Statistical Analysis and Monitoring . . . 55

8 Integration and Performance Analysis **60**
8.1 Integration with Apache Taverna . . . 60
8.1.1 Fixed Routine for ProVerna . . . 60
8.1.2 Coding Template . . . 60
8.2 Benchmark for Comparison . . . 60
8.3 Result Analysis . . . 61
8.3.1 System Performance Analysis . . . 61
8.3.2 Query Performance Analysis . . . 62
8.3.3 Notes on Performance Overhead . . . 62

9 Comparisons **68**
9.1 Provenance Mechanism Comparison . . . 68
9.2 Completeness of Provenance Questions Taxonomy . . . 68
9.3 Comparison of Reporting Service . . . 71

10 User Study **72**
10.1 The Method . . . 72
10.1.1 Learning ProvMod . . . 72
10.1.2 Implementing ProvMod . . . 72
10.1.3 Inspecting Log Schema . . . 73
10.1.4 Learning Taverna . . . 73
10.1.5 Using ProVerna . . . 73
10.1.6 User Information and Task Load Assessment . . . 73

10.2 Evaluating Dashboards . . . 74
10.2.1 Assessing Graph Visualization . . . 74
10.2.2 Assessing Monitoring Visualizations . . . 75
10.3 User Information and Skill Level Demonstration . . . 77
10.4 NASA TLX Score . . . 79
10.5 Limitations of The Study and Future Goals . . . 81

11 End User Features **82**
11.1 Tool Development . . . 82
11.2 Workflow Implementation . . . 84
11.3 Logging Configuration . . . 84
11.4 Tuning Log Levels . . . 85
11.5 End User Reporting . . . 85
11.6 Integration . . . 85

12 Two Software Engineering Use Cases of ProvMod **86**
12.1 Clone Detection Meets Provenance . . . 86
12.1.1 Questions about Clone Analysis . . . 87
12.2 Architecture Extraction of Legacy Systems . . . 91
12.2.1 Questions about Architecture Extraction . . . 91

13 Related Works **95**
13.1 Surveys . . . 95
13.2 Automated Logging . . . 100
13.3 Data Provenance and Querying . . . 104
13.4 Workflow Provenance and Querying . . . 108
13.5 Analytic Provenance: Querying with Visualization . . . 111
13.6 Software Engineering . . . 113

14 Discussions **115**

15 Conclusion and Future Work **117**

References **119**

List of Tables

5.1 Provenance mechanism of ProvMod 34
5.2 User Properties 34
5.3 Object Properties 35
5.4 File Properties 35
5.5 Document Properties 36
5.6 Module Properties 36

9.1 Provenance mechanism comparison of Taverna, ProvMod and ProVerna 69

10.1 Average TLX score out of 10 from the user study. Scale 0 to 10 refers to low to high score. . 80
10.2 Median for TLX scores from the user study 80
10.3 Standard Deviation for the TLX scores from the user study 81

List of Figures

1.1 A layer based architecture for our proposed programming model . . . 3
1.2 System components services architecture . . . 4

3.1 Relation among components of ProvMod in a workflow . . . 12
3.2 Module behavior with conditional . . . 13

4.1 A classification of provenance questions in our work . . . 17

5.1 Implementation of ProvMod and relation between other components of the whole workflow system that we propose . . . 24
5.2 Lifecycle of workflow systems by Cruz et al. [1] . . . 25
5.3 Taxonomy presented by Cruz et al. [1] . . . 26
5.4 Description of ProvMod based on the taxonomy of Cruz et al. [1] . . . 28
5.5 ProvMod model components . . . 29
5.6 JSON log structure . . . 31
5.7 Implementing a workflow . . . 32
5.8 Tool development and integration template . . . 33

6.1 Experimental dataset overview . . . 38
6.2 First workflow for simulation . . . 38
6.3 Second workflow for simulation . . . 39
6.4 A portion of the provenance graph with 300 nodes generated by the simulation with sparse workflow. Green nodes are different modules with module name as label. Red nodes are intermediate Data Object items with data value as label. Blue nodes are Data File items with file path as label. The edges are labeled with IN or OUT. IN is directed from data to module and OUT is directed from module to data. . . . 40
6.5 The large workflow used in our experimental analysis . . . 41
6.6 A portion of the provenance graph with 100 nodes generated by simulation with large workflow. The red node is the Sharpen module labeled with tool name and all other green nodes are generated output data from Sharpen. Green data nodes are labeled with an ID. There is only one input to Sharpen in the full graph that is not highlighted here. The graph is expanded around the single Sharpen node only. . . . 42
6.7 The deep workflow used in our experimental analysis . . . 43
6.8 A portion of the provenance graph with 100 nodes generated by simulation with deep workflow. The red nodes are the different instances of Sharpen tool with same parameter setting. Green nodes labeled with IDs are either an input or output data items. One generated data item is used as input in the next module. . . . 44
6.9 Execution overhead of sparse workflow . . . 45
6.10 Query overhead of sparse workflow . . . 46
6.11 Comparison of ProvMod execution time with and without provenance . . . 47
6.12 Execution overhead of large workflow . . . 48
6.13 Query overhead of large workflow . . . 48
6.14 Execution overhead of deep workflow . . . 49
6.15 Query overhead of deep workflow . . . 49

7.1 A graph visualization to answer graph related question that does not return a single metric value, Objects are Blue, Modules are Red and Files are Yellow in color . . . 51
7.2 A time series visualization for representing real-time, time-series investigation of any property 52
7.3 A time series visualization for representing real-time, time-series investigation of multiple properties . . . 52

7.4 An average of CPU load for a fixed-time range for different modules . . . 53
7.5 An average of CPU load for a fixed-time range for different modules with a goal of 50% CPU load . . . 53
7.6 Metric view of two nominal property frequency . . . 54
7.7 Pie chart to provide a proportional view of a nominal property (here success and MRE or module run-time error) . . . 55
7.8 Pie chart to provide a proportional view of two nominal property frequencies (here success and MRE or module run-time error along with the module types) . . . 56
7.9 Heatmap to represent tool name vs error type to frequency of their occurrence . . . 57
7.10 Heatmap to represent tool name vs error type to CPU load . . . 58
7.11 Percentile distribution of CPU load grouped by tool names . . . 58
7.12 Standard deviation of CPU load for different tools . . . 59

8.1 A coding template for the developer that implements RemoveSpace function, to remove any space in the file content . . . 63
8.2 A real-world pipeline as the benchmark for ProVerna . . . 64
8.3 Comparing ProVerna and Taverna . . . 65
8.4 Execution time overhead based on the number of iterations . . . 65
8.5 Memory overhead based on the number of iterations . . . 66
8.6 Time taken to build graph during workflow run for two kinds of queries . . . 66
8.7 Full graph query time based on the number of nodes in the provenance graph . . . 67
8.8 Performance of elementary queries Decide, Time Sequence Mapping and Data Sequence Mapping 67

9.1 Query coverage of proposed taxonomy with respect to existing works . . . 70
9.2 Visualization feature comparison with respect to existing analytic provenance research works 71

10.1 Python programming fluency of different users . . . 77
10.2 Workflow experience . . . 77
10.3 Experience level with JSON . . . 78
10.4 Feedback on the question: Is the coding template reusable? . . . 78
10.5 Feedback on the question: Is the documentation simple, concise and to the point? . . . 79

11.1 Implementing existing tool FastQC in ProvMod . . . 82
11.2 Developing a tool from scratch . . . 83
11.3 A pipeline to implement the tool Entropy from ProvMod library with input file gene.txt, to produce a data item with the entropy value in it . . . 84

12.1 Specialized clone analysis approach with the elementary queries . . . 87
12.2 JHotDraw clone visualization. JHotDraw is a notable tool for its adherence to design principles. The nodes are function level code fragments labeled with an ID from the NiCad results. All clones are clustered in the graph. Any node connected to another means that they are both clones to each other. . . . 88
12.3 Bioblend - A Galaxy library's clone visualization . . . 89
12.4 Taverna Mobile clone visualization . . . 90
12.5 Specialized functional dependency extraction approach with the elementary queries . . . 91
12.6 A functional dependency graph for a specific usage scenario of CRHM. The system starting point is the yellow root node labeled with 'crhm'. Nodes system functions. Those are categorized and colored on the basis of their source file name. The nodes are connected with directed edge labeled 'calls'. So, if node A calls node B, there is a directed edge from A to B. . . . 93
12.7 A dependency graph for selected functions from the graph for a specific scenario . . . 94

1 Introduction

Workflow provenance maintains information about workflow processes, configurations, data propagation and lineage through logging. The importance of provenance has been pointed out by different researchers [1, 2, 3, 4]. For business process analysis, knowledge management, process re-engineering, and business intelligence, provenance is necessary [5]. Davidson et al. [6] clearly states, provenance is necessary for reproducibility and it enables data analysis and further exploration through collaboration. Besides, provenance is necessary for debugging, administration, error detection and fixes as well as security, data authentication, origin tracing, storage management and anomaly detection [7, 8, 9]. As scientific workflow systems have evolved to tackle Big Data in the cloud computing environment [7, 8], workflow provenance can make such systems efficient and traceable. Automated provenance is necessary to go beyond humanly written queries, data exploration, pattern mining, error detection, validation, diagnosis, collaboration to manage diverse or heterogeneous data sources [8]. For data-centric performance measurements, provenance is required [10]. Causal lineage capturing is important and can be done with provenance [8]. Real-time monitoring of provenance information makes it possible to use already allocated resources to be used again for the failed processes and then fine-tuning of a parallel process is possible [11]. Querying for identifying provenance graphs, overlaps among the graphs and similarities among them is discussed by Karvounarakis et al. [12] which is an important research topic. Above all, data visualization is also tightly coupled with any topic of provenance research [7, 8] to offer complex insight.

Even though provenance is a crucial part of any workflow system, provenance mechanisms differ from system to system [13, 14]. Cross-domain research and collaboration are now of paramount importance in data science and big data analytics. Prominent workflow systems and their features are best modularized to support modification and integration. Developers may need to modify or update the provenance mechanism of any existing workflow system to meet certain goals. The design and mechanism of provenance highly depend on the design, architecture, modelling, and orientation of the workflow systems. Thus modifying or updating the provenance part of a workflow system is not a common task with a standard approach. Building a new workflow system is also challenging and additional effort is necessary in order to integrate provenance into a new system. There are different tools and technologies for capturing and logging events (Log4j for Java, Logging for Python) but they were not built targeting provenance. So even though the developers use them they still need to design and implement a provenance model which will be able to answer certain provenance questions. Such development tasks are burdensome [15]. It is because provenance is not only about logging events in any software system but also about retrieving information about the workflows from

those logs.

To solve the stated problem, we propose a comprehensive taxonomy of workflow provenance questions influenced by existing research. Standing on the proposed taxonomy and state of the art of workflow research, we propose a workflow programming model 'ProvMod' which is capable of automated logging and oriented to answer a variety of provenance questions. Such automation is of high importance [15, 16, 17]. ProvMod is provided as a library and can also be integrated with any existing workflow system. Such logs hold a number of properties ranging from the workflow level to the OS level that is solely focused on answering provenance questions from the taxonomy. The logs follow a simple JSON structure. Thus it can be adapted to any NoSQL technology to fit into big data systems which is important [18]. Even the developers can add or reduce properties in the log schema to capture the desired amount of information.

ProvMod is developed using an OOP (Object Oriented Programming) and thus easy to implement with less learning curve. With a layered architecture (illustrated in Figure 1.1), every component stands separately as a distinct service. Such architecture is inspired by existing work [18, 19]. On the OOP layer, there stands the model. The tools stand on the model and there could be a DSL in the middle for domain specific tasks. Logging configuration is also kept separate for custom logging with different logging levels. Such customization is highly inspired [7]. The core services are illustrated in Figure 1.2. We implement the model in Python. ProvMod leverages the power of Neo4j graph database to handle semi-structured heterogeneous log data. A highly coupled graph database support is better for provenance [20]. It can be also be directly used to represent workflows DAGs (Directed Acyclic Graphs). Above all, the end user also gets real-time monitoring features in ELK Stack which is crucial for auditing a running system [5, 11]. Elasticsearch is used in ProvMod for real-time log parsing and Kibana is used for monitoring, data visualization, and statistical analysis.

Finally, in order to show that our model is adaptable to existing workflow systems and configurable, ProvMod is integrated with Apache Taverna Workflow System [21] which is done with our provided simple tool development template. It generates a new workflow system that we name ProVerna. ProVerna not only has the Taverna front end and features for building workflows but also all the facilities of ProvMod. We also compare Taverna, ProVerna, and variations of ProVerna to evaluate the overhead of the model from different angles.

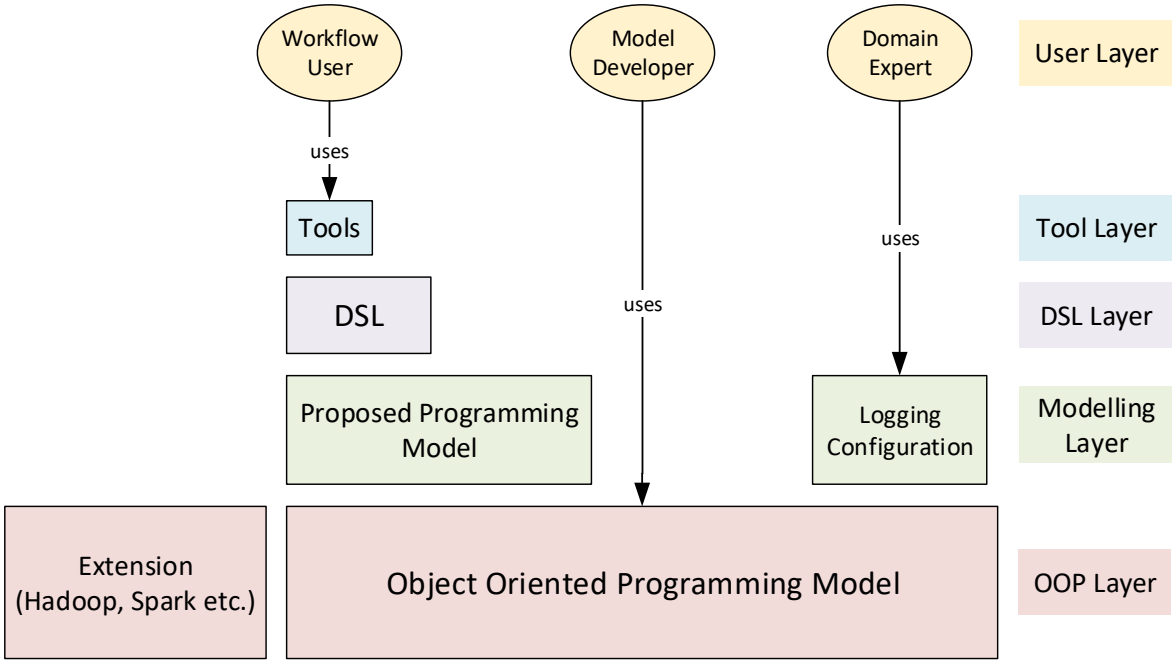

Figure 1.1: A layer based architecture for our proposed programming model

1.1 Contribution

We try to touch every parts and phase of the workflow systems and executions. So, we have a number of contributions that are distinct. We made the following contributions in this work:

1. We propose a provenance question taxonomy, that covers a number of provenance questions. The taxonomy can even be extended.

2. We propose a workflow programming model named ProvMod, that is oriented to answer workflow provenance questions.

3. ProvMod follows a layered architecture and SoC design principle to keep the workflow, provenance, provenance database, tool database and visualization services of the model separate and independent.

4. We define a graph model for ProvMod that is highly adaptable to Graph Databases.

5. ProvMod is adaptable to existing tools. When adapted, all the features of ProvMod and the tools will be available in the newly adapted tool.

6. Developing new tools from scratch is also supported in ProvMod in an easy fashion and minimal OOP knowledge is required.

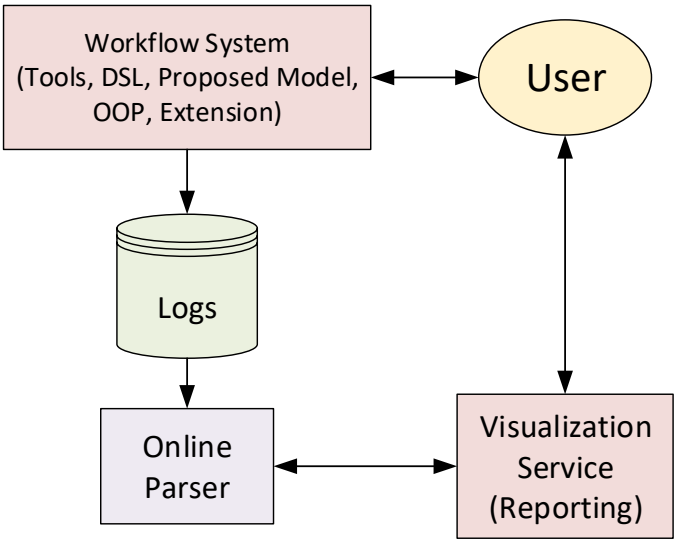

Figure 1.2: System components services architecture

7. Workflow implementation with ProvMod is designed with care and during the workflow implementation developers do not need to focus on anything other than the pipeline.
8. Implemented workflows and tools can be reused or shared with other developers.
9. Workflows in ProvMod can support conditional module execution as well as multiple distinct inputs and outputs.
10. Due to layered architecture, the user can create another custom layer between the workflow and ProvMod to offer DSL (Domain Specific Language).
11. ProvMod captures workflow level provenance automatically as well as the user can generate their own custom log at any step of the implementation at will.
12. Automated logging has small overhead and does not affect the data analysis.
13. The automated logging can be disabled or even modified based on the target domain.
14. Logging and the workflow modelling is completely separate in ProvMod. So, domain experts can modify the log schema/configuration/model and developers/analysts can focus on only implementing the workflows.
15. ProvMod is integrated with Neo4j Graph Database and the workflow provenance graphs support any Cypher query for analysis.
16. Data visualization is the primary way of answering provenance questions in ProvMod. We offer a number of real-time data visualizations that can answer a number of provenance questions.

17. We use several modern technologies to ship the model such as, Python programming language for workflow implementation, RESTful Web Service for cloud integration and separation of components as services and security, JSON for logging and adaptation with NoSQL databases like Cassandra and CouchDB, Elasticsearch for automated log management, data streaming and security, Kibana and D3 for data visualization, Neo4j for workflow graph management and analysis with Cypher query language.

18. We show how complete our provenance questions taxonomy is based on the state of the art research works. We also describe the mechanism of ProvMod provenance and compare with other provenance enabled systems. Again, we also compare the enhanced visualization features of ProvMod with existing analytic provenance systems.

19. For ProvMod we show several use cases to present its application and advantages in other software engineering topics.

20. We analyze our model with three different kinds of workflows that we define, sparse, large and deep to observe the model performance behaviour from different angles. For these purposes, we implement Bioinformatics and Phenotyping image analysis workflows.

21. We compare ProvMod with Apache Taverna Workflow System [21] for evaluating ProvMod using a real-world Bioinformatics benchmark and evaluate the model.

22. We also integrate our model with Apache Taverna to offer a new model named ProVerna that offers end user with the Taverna GUI and all the ProvMod features in the background.

23. We perform two separate user studies on implementing the programming model and the visualization supports of ProvMod.

2 Preliminaries

We present the definitions and overview of the research of several concepts for the readers' simplicity here. It will help them to get an overview and idea about the importance of this research.

2.1 Terms and Definitions

We present some basic definitions for several terms that we use throughout this thesis frequently.

2.1.1 Programming Model

A programming model is based on a theoretical concept that helps the developer to implement the theoretical concept without implementing it from scratch.

2.1.2 Provenance

Provenance in computing means to save the relevant information whenever a data is processed and transformed so that later on, using the saved logs the process details can be regenerated.

2.1.3 Workflow

A workflow is basically a directed graph that represents the data flow and propagation through different data analysis tools. The graph contains different types of nodes to represent data and tools. The directed edges represent the data flow direction. Workflows may or may not contain cycles that represent loops. Any workflow that never has loop turns into a directed acyclic graph (DAG).

2.1.4 Workflow System

A workflow system is a system that offers various features and tools to implement pipelines and workflows to implement data-flow and run them without building them from scratch. Workflow systems are more formally known as workflow management systems. Workflow systems can be made for scientific or business domains. Scientific workflow management systems are sometimes referred as SWFMS in short.

2.1.5 Module

In this thesis, with the term module, we refer to any data tool in a workflow system.

2.1.6 Data Item

By data item, in this thesis, we refer to any data object in a workflow system.

2.1.7 Reporting

Reporting refers to present any final result using various forms such as data visualization and interactivity.

2.2 Importance of Provenance

The importance of provenance was strongly mentioned by different researchers. In this section, we present a number of critical points that got attention from researchers in recent years. The following statements are clearly mentioned in the existing works that provide motivation to our research.

1. For business process analysis, knowledge management, process re-engineering and business intelligence provenance is necessary [5].

2. Davidson et al. [6] in their work states the importance of provenance. They also discuss clearly that, it is necessary for reproducibility. It enables data analysis and further exploration. Data mining and visualization are also tightly coupled with provenance research. Provenance enables collaboration.

3. Workflow has already entered into the domain of cloud computing thus entered provenance. For data-intensive systems it is important [7, 8]. Provenance is necessary for debugging, administration, error detection and fixation as well as security, data authentication, origin tracing, storage management, anomaly detection [7, 8, 9].

4. The integration of provenance in Big Data is burdensome and thus it should already be there when a data-intensive system is ported [15]. Automated provenance is necessary to go beyond humanly written queries, data exploration, pattern mining, error detection, validation, diagnosis, collaboration to manage diverse data sources.

5. For data-centric performance measurements of any system, provenance is required [10].

6. Provenance can also be used to analyze legacy systems [22].

7. For heterogeneous data management, provenance is required in cloud computing and data-intensive systems [8].

8. Causal lineage capturing can be done with provenance [8].

9. Real-time monitoring makes it possible to use already allocated resources to be used again for the failed processes [11] Fine tuning of parallel processes also becomes possible then. It also enables elasticity.

10. Provenance can be shipped in two ways, lazy or eager and which way is better in which case is a matter of evaluation [23, 10].

11. Provenance components from workflow systems were identified by Hartig et al. [24].

12. Querying for identifying provenance graphs overlaps among the graphs and similarities among them is discussed by Karvounarakis et al. [12].

13. Workflow provenance and data provenance concepts are clearly distinguished and the importance of workflow provenance can be found in the work of Amsterdamer et al. [25].

2.3 Important Features of Provenance

Various features of provenance were also mentioned by the recent researches that they emphasize for any provenance enabled systems. Now we present a number of crucial features that is very important for any provenance enabled systems. The following features, when present in a provenance enabled system, can make it complete.

1. VanderAalst et al. and Costa et al. [5, 11] mention the importance of real-time monitoring in their work as data-intensive processes takes time to finish and processes need to be investigated in the middle of execution. Larger workflow management systems contain more noise due to human or system related errors and thus it becomes more complex to mine provenance from such bigger systems [5].

2. The feature of conditional workflow is mentioned by Bahsi et al. [26].

3. Freire et al. [20] mentions that tightly coupled provenance facilities are better for workflow management systems.

4. Davidson et al. [6] state, the need for simplicity and usability is very important for provenance. Debugging system problem through visualization is also emphasized by their work.

5. Adaptability and custom configuration are necessary by provenance [7] since configuring logs require domain knowledge and that is time-consuming as well as error prone.

6. Various level of user access is important for ensuring security and protection of data and the system [7].

7. Overhead or performance is an issue to consider in provenance for data-intensive systems [15, 8, 18].

8. Security is another issue to consider in provenance [15, 8, 18].

9. Automation in provenance is required [15, 17, 16, 17].

10. A common log structure is necessary that is stated by Glavic et al. [10].

11. User-oriented provenance model is emphasized in existing works [27]. User-based custom annotation is preferred [18].

12. Provenance in an open world system rather than a closed one is preferred for heterogeneous data based systems [22]. Heterogeneous data modelling should be supported [18].

13. Legacy system's provenance capturing is also important [22].

14. The need of elasticity or adaptability of system tools is necessary for provenance [8].

15. We need APIs for application-specific provenance capturing to ensure extendability [8, 19].

16. Atomicity (the most elementary information) of provenance data is important for workflow systems [8].

17. Causal lineage ordering is necessary for provenance [8].

18. A layer-based architecture is preferred by researchers for provenance enabled workflow systems [18, 19].

19. Scalability is also necessary for provenance in data-intensive systems [18].

20. Data visualization and further data analysis features are important for provenance [18].

21. Graph database could be used in provenance for analyzing semi-structured or unstructured data [18].

22. Temporal and time-series data analysis is emphasized in provenance analysis to get a better understanding of the system and process changes [18, 23].

23. A need for a general standard in provenance research was mentioned by Buneman et al. [28].

24. Different levels of provenance information with properties were emphasized by researchers [9].

25. Provenance facilitated system should have query features [9, 19].

26. Fault tolerant database management is necessary for provenance [29].

3 ProvMod: The Model

This section presents the theoretical definition of our model including different components. We define the workflow, data, module, their properties, data flow, invocation and type. The model is oriented to answer provenance questions. Different provenance questions with a proposed taxonomy are described in Chapter 4. Although the model is oriented to answer provenance questions, it is not fully dependent on the question types and queries. It may create problem and limitation in the long run. To overcome such issues, we introduce properties in the model. Any model component holds certain properties that are relevant to the system domain. More properties can be added if necessary. So, in summary, any provenance question our model answers, the answer is derived from the properties. For supporting different workflow systems, we introduce the module and data items. Clearly, any further kind of workflow artifacts can be introduced.

3.1 Design Goals

When designing the model, we follow some simple design goals. They are:

1. The model is oriented to answer provenance questions.
2. Any workflow artifact will hold specific properties. Any information about the artifact has to be retrieved from the properties of the artifact only.
3. Since, the artifacts' information is solely retrieved from the properties, new properties can be introduced in a new system.
4. The model has to be as simple as possible with respect to workflow implementation. So, it will offer some familiar operations that are relevant to any workflow implementation.
5. The model will follow SoC to interact with other system components.
6. The model will offer automated logging.

3.2 Definitions

In this section, the theoretical definitions of the model components are presented.

3.2.1 Workflow W

A workflow $W = (V, E)$ is a Directed Acyclic Graph (DAG) $V = \{v : v \in D \cup M\}, E = \{e : e \in F\}$ where D denotes Data, M denotes Module and F denotes Dataflow.

3.2.2 Data D

Data is a workflow element from $W = (V, E)$ where, $D = \{d : d \in V, d = (p_i)_{i=1}^{n} = (p_1, p_2, ..., p_n), p_i \in P\}$ and P denotes an ordered set to represent a number of properties of D.

3.2.3 Module M

Module is a workflow element from $W = (V, E)$ where, $M = \{m : m \in V, m = (p_i)_{i=1}^{n} = (p_1, p_2, ..., p_n), p_i \in P\}$ and P is an ordered set to represent a number of properties of M.

3.2.4 Properties P

Properties $P = (p_i)_{i=1}^{n} = (p_1, p_2, ..., p_n)$ is an ordered set associated with any data element $d \in D$ or module item $m \in M$ from a workflow W where $Type(p) \in \{Null, Any\}$.

3.2.5 Type

Type is a function that returns the data type of any property item p from the workflow when applied upon it. The data type could be $Null$ or Any. Any is a set that contains all possible data types from a particular implementation of the workflow W.

3.2.6 Dataflow $\rightarrow$

A dataflow F is an ordered tuple containing two workflow elements to represent a directed edge in a workflow W where, $F = (d, m)|(m, d), d \in D, m \in M$**.** (d, m) and (m, d) are correspondingly considered as input dataflow (written as $d \rightarrow m$) and output dataflow (written as $m \rightarrow d$).

3.2.7 Module Invocation

A module invocation K is an ordered tuple containing two dataflows where, $K = \{(f_1, f_2) : f_1 = (d_i, m), f_2 = (m, d_o)\}$.

There are several features that are supported by our model based on the model components' definitions. They are,

1. The users can be considered as a property any workflow components.
2. Properties can be modified for different components as well as systems.
3. Modules can not only have multiple inputs but also multiple outputs.
4. Multiple inputs or output data items can be different.
5. Modules can be incorporated with certain logical conditions and can generate default values when not executed.
6. The model can be incorporated with any types from a certain implementation.
7. Module abstraction and invocation are conceptually separated from each other.

How the components are related to each other can be understood from the Figures 3.1. How a module behaves with a condition is described in Figure 3.2.

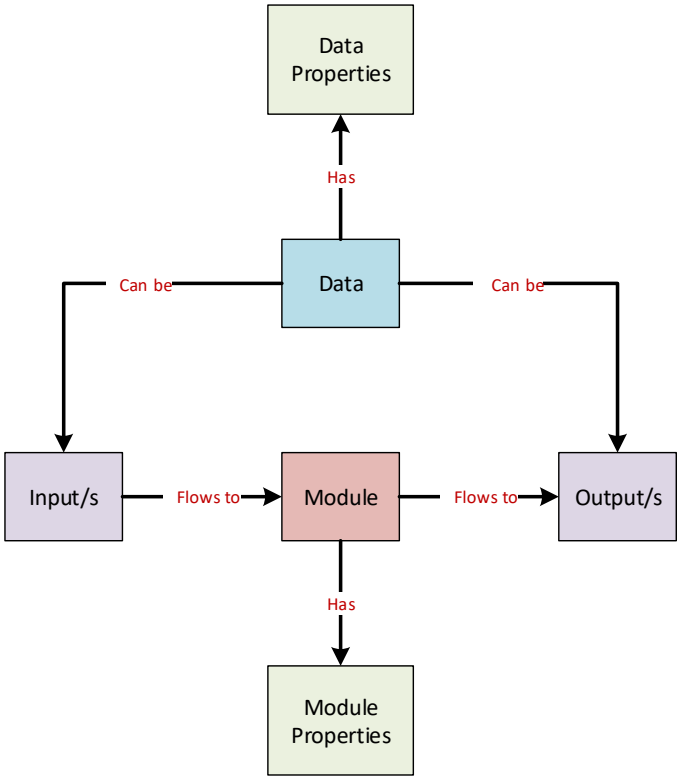

Figure 3.1: Relation among components of ProvMod in a workflow

The module components can also support conditionals that are already mentioned. When a module condition is false, a default data flow can be generated in ProvMod.

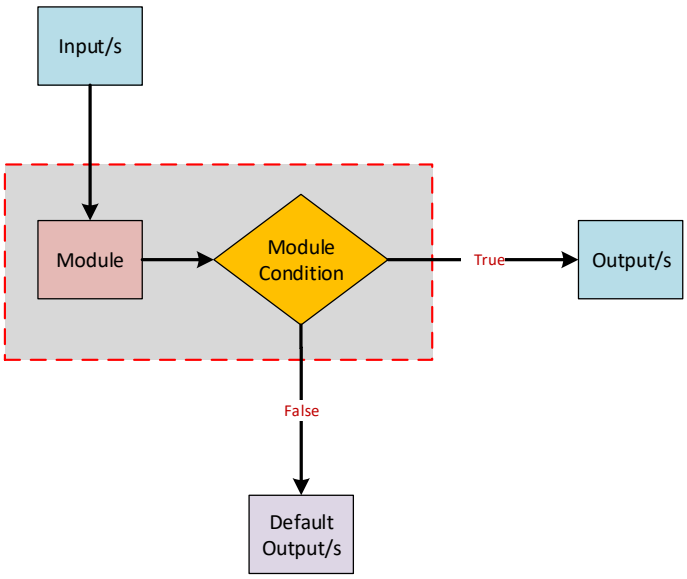

Figure 3.2: Module behavior with conditional

3.3 ProvMod, Graph and Graph Database

The very basic relational schema between data and module items (Figure 3.1) is indeed a graph. This graph can be considered the most elementary workflow for our model. The graph is more specifically a DAG that contains one data as input to a module and one data output from that module. From this observation, we become encouraged to acquire graph database technology in our model ProvMod. A graph database is one of the modern database technologies that is getting huge interest in the Big Data domain. Even many relational database technologies are being replaced by NoSQL databases [30]. For example, Google's BigTable [31] and Facebook's Cassandra [32] are coming into light. Graph database such as Neo4j offers a trivial way to implement such graph data structure quickly [33]. Surely, such structure can be obtained and represented through relational database tables, but that will add up another overhead of implementation to the model. Graph database such as Neo4j also offers great performance and is lightweight [30]. Our model also being property-centric, Neo4j is of great advantage to us. Neo4j also has other APIs available to work with for different purposes. Thus the graph database can later be extended as we obtained Neo4j.

Based on the motivations presented, design goals and future goals we move forward to model the querying features of our model ProvMod.

3.4 Querying

We propose a set of 3 queries that can cover a number of provenance queries from the existing literature and also can be used to derive more. The last two are derived from the first one. So, we consider them as elementary queries and the first one to be an atomic query.

3.4.1 Decide: The First Elementary Query

We propose the first elementary query and name it Decide. It is named that way because this query is used to retrieve the smallest information from the provenance graph. Such information could be for example, the value of a target property of a target node.

The elementary query Q is a function of the workflow W that returns a collection of property p that satisfies certain condition(s) C. So,

$Q = f(W, C)$ where $p \in P$ and C is a Boolean condition. In Cypher the query format should be,

MATCH (n:Type) WHERE Condition RETURN n;

3.4.2 Extension of the First Elementary Query

The first query Decide can be extended to generate workflow and pipeline module chains and data tables as property to property mapping.

3.4.3 Module Chain Regeneration with Decide

A workflow could be regenerated using the elementary query Decide with a sequence of dataflows.

For example, $Q_1 = A \rightarrow B, Q_2 = B \rightarrow C, Q_3 = C \rightarrow D$ which gives $A \rightarrow B \rightarrow C \rightarrow D$.

3.4.4 Property Mapping with Decide

A dataset could be generated using the first elementary query Decide.

For example, $Q_1 = (x_1, x_2, ..., x_n)$ and $Q_2 = (y_1, y_2, ..., y_n)$ thus $Q_1 \leftrightarrow Q_2$ is an one-to-one mapping between the ordered sets that gives $((x_1, y_1), (x_2, y_2), (x_3, y_3))$.

The two extensions can be used to create two other important queries that we consider as elementary but derived from the first query Decide. Note, every node in the graph is also uniquely identified with an

identifier property. So, querying that unique property is indeed getting nodes from the graph.

3.4.5 Time Sequence Mapping: The Second Elementary Query

Time-series analysis is an important issue in provenance and thus we consider it as an elementary query so that it can be featured as a native facility in ProvMod and used for further analysis in a portable manner. Using the extensions described above, the Time Sequence Mapping query structure in Cypher will be,

MATCH (n:Type) WHERE Condition RETURN n.p ORDER BY n.time;

We name this elementary query that way because it is used to retrieve a time-series data. Since time-series is a very frequently used data analysis topic, we consider it as an elementary query.

3.4.6 Data Sequence Mapping: The Third Elementary Query

If we want to map between two different properties from the nodes of the graph, we can use this query to derive a dataset for further analysis in general. The Data Sequence Mapping query structure in Cypher is,

MATCH (n1:Type1), (n2:Type2) WHERE Condition AND n1.p == n2.p RETURN n1.p1, n2.p2;

Data to data mapping also being important and very frequently used data analysis topic, we also consider this query as elementary and name it that way.

4 Classifying Provenance Questions

In this chapter we demonstrate our proposed provenance questions taxonomy to classify them. For each category from the classification, we mention a number of derived questions in the following sections. For all the questions, we also provide the reader with a template so that they can implement it in a graph database technology. We focus on Cypher query language primarily, though we present the queries in a possible generic format. To get a clear idea for each query, the queries are accompanied with examples.

4.1 Methodology of Deriving Taxonomy

We follow a theoretical approach to derive such a taxonomy illustrated in Figure 4.1. The following matters are considered when we create the taxonomy.

1. Whenever we ask a workflow provenance question, its answer can be directly retrieved from the provenance graph.

2. We have already proposed a model that is based on three provenance queries. Those three queries are related to the provenance graph.

3. Clearly, the question can be either about a particular data value (Decide query), or a time-series of data values (Time-Sequence mapping), or a mapping between two kinds of data values (Data-Sequence Mapping).

4. These three queries can be combined with each other to derive any other types of data mapping holding multiple data items.

5. The graph related provenance questions are indeed to derive a desired dataset for further analysis. It is because, if we have the dataset, we can answer relevant questions. So, we propose monitoring related questions that works with aggregation.

6. We propose three basic aggregation types, Time-series, Proportional and Metric.

7. Finally, the question may involve multiple aggregation, that we call Statistical Analysis questions. This kind of questions involves applying different tools and techniques for prediction, clustering and so on.

8. We classify Statistical Analysis type into further generic categories, Graph analysis, regression or clustering and so on.

Since we consider real-time monitoring, we also include real-time or fixed-time category in our taxonomy. Again, if we find any general question type in future that is not already in the taxonomy, it can be added in the relevant class to extend the taxonomy. Workflow provenance questions were mainly derived from the graph related questions, but we also include analytic questions. Any aggregation is covered by Monitoring questions and any tool application is covered by Statistical questions.

4.2 The Provenance Questions Taxonomy

Provenance questions and their answers are our main goal to be discovered in this work. So we discuss this topic in a separate chapter. We categorize the question into three parts. A brief picture is in Figure 4.1. They are as follows:

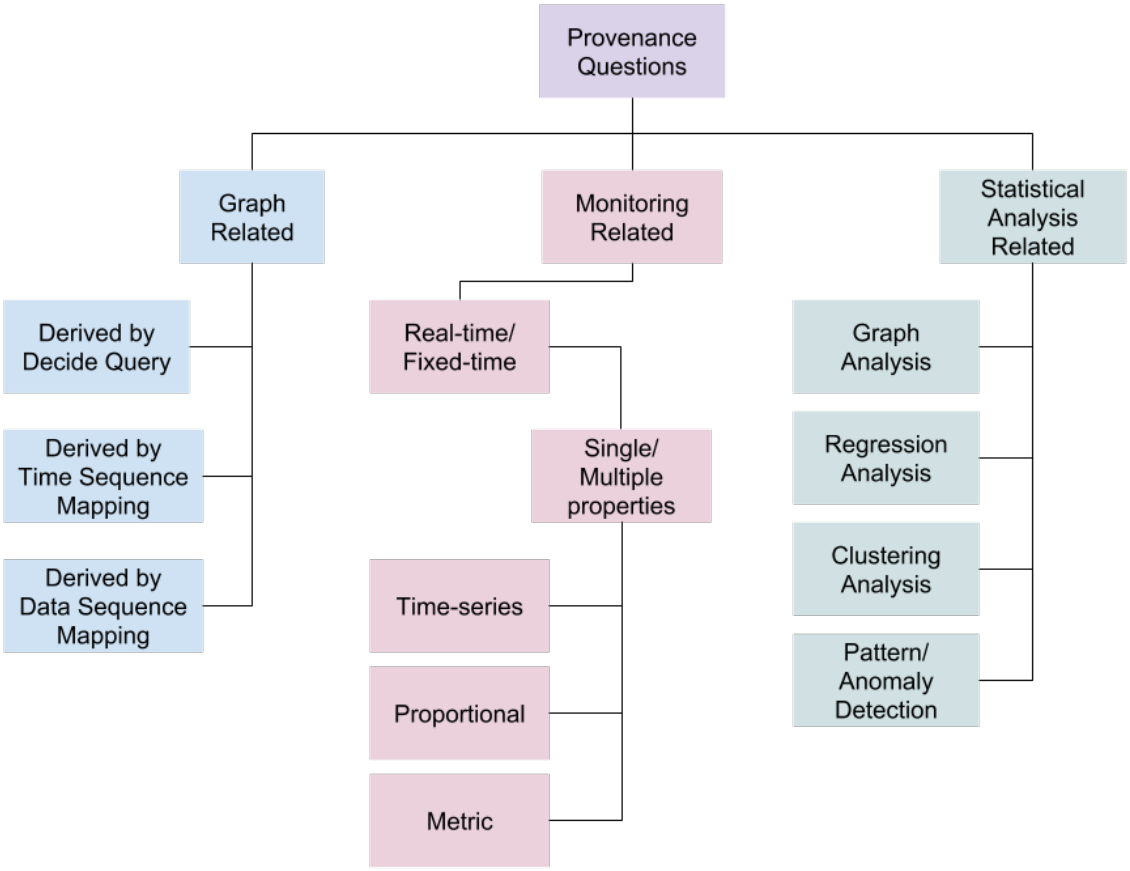

Figure 4.1: A classification of provenance questions in our work

1. **Graph related questions**: Graph related questions involve the workflow graph analysis at any time. It is an offline mode of analysis and not real-time.

2. **Monitoring related questions**: Monitoring questions involve any workflow graph/node related metric (properties) analysis that is also real-time. It is related to simple aggregation on the provenance graph data.

3. **Statistical analysis related questions**: Statistical analysis related question involve any kind of complex analysis to find any statistical measurement of the workflow graph/node related metrics (properties). These questions involve the application of different statistical or machine learning based tools on the provenance data. Such kind of analysis largely depends on the approach and available data.

4.3 Graph Related Questions

In this section, we present a number of provenance questions that we focus to answer in our work that is related to workflow graph analysis in offline mode. The questions are from very basic to advanced level for provenance and some of them were already discussed in recent works from different angles. Even some of the existing works only focused on several of them. We categorize the queries into different classes and each of them contains many questions that fall under provenance.

Also, since these questions are for graph analysis, we present the corresponding Cypher queries for their retrieval in this section. The Cypher queries mentioned here are in abstract form to show the general structure of them. For more details, the reader may refer to the Implementation section.

The very first question of workflow provenance is:

- **What is the complete workflow DAG G?**
  Match (n) return n;
  For example, this query can be used to retrieve the complete provenance graph collection until now from the beginning of time the system was being used.

The first question can be used to derive several other provenance questions. They are as followed:

- **What is the workflow DAG G containing node n1, n2, ... , nn?** (Using Decide)
  Match (n1), (n2), ... , (nn) Where n1.p = n_1, n2.p = n_2, ... , nn.p = n_n return n1, n2, ... , nn;
  For example, this query will retrieve a subgraph that has only some particular nodes identified by a property. The property itself can be an unique identifier. The subgraph will also hold any edges if there is any among the target nodes.

- **What are the inputs of a module M in workflow DAG G?** (Using Decide)
  Match (i), (n) where (i)-[:INPUT]->(n) return i, n;
  For example, this query will retrieve only the input and the target module nodes as a subgraph from the full provenance graph. INPUT is a ProvMod keyword.

- **What are the outputs of a module M in workflow DAG G?** (Using Decide)
  Match (O), (n) where (n)-[:OUTPUT]->(O) return n, O;
  Similar to the previous query this query will retrieve the outputs and the target module node. OUTPUT is a ProvMod keyword.

- **What are the outputs of a module M in workflow DAG G?** (Using Decide)
  Match (i), (n), (O) where (i)-[:INPUT]->(n)-[:OUTPUT]->(O) return i, n, O;
  This query will retrieve the input and output nodes and the target module node as a pipeline from the full provenance graph.

- **Does a node hold a property P in a workflow DAG G?** (Using Decide)
  Match (n:Type) where n.P1 <> Null return "P";
  This query will check whether a property is held by a certain type of node.

- **What is the value of property P of node N in a workflow DAG G?** (Using Decide)
  Match(n) return n.p;
  For example, this query will retrieve how much CPU load was used for a module run which is saved as one of its property.

- **What is the collection of all workflow DAGs** $G_1, G_2, ..., G_n$**?** (Using Decide)
  Match(n) return n;
  This query is the same as the very first query that we have already mentioned.

- **What is the time ordered list of property P from workflow DAG collection** $G_1, G_2, ..., G_n$**?**
  (Using Time Sequence Mapping) *Match(n) return n.P order by n.time;*
  For example, this query will retrieve all the module names from the full provenance graph with ascending order of their creation time, so that we can find out, the serial order of modules that were executed one after another.

- **What is the property K ordered list of property P from workflow DAG collection** $G_1, G_2, ..., G_n$**?**
  (Using Data Sequence Mapping)
  Match(n) return n.P order by n.K;
  This query can, for example, retrieve the module names in ascending order by their CPU load, so that we get an idea of- which module uses the most or least CPU during execution. Such a query can also be used to create datasets from the full provenance graph. The previous query is also a derivation of this query in a simple manner.

- **What is the workflow DAG G holding a certain condition C?** (Using Decide)
  Match(n) where C return n;
  To retrieve any subgraph from the full provenance we can use this structure of Cypher query.

The graph related provenance questions and corresponding Cypher query can also be found in the Appendix.

4.4 Monitoring Related Questions

Provenance monitoring involves real-time observation of provenance-related properties such as system resources, error occurrences and so on. With these questions, the user can get an idea of what is going on in the workflow system. Here we present the question in a generic format that can be adapted to any workflow model. Monitoring is highly related to data visualization in our work. The questions are also classified into several categories.

4.4.1 Real-time Aggregation

Real-time Aggregation on Single Property

For any aggregation A on any property P, from time T, for workflow node group G:

- **What is the real-time, time-series representation?**
  For example, we want to monitor the CPU load of a workflow system integrated with ProvMod. This can be a line chart, area chart or even a bar chart. The X-axis is the time and the Y-axis is the CPU load in this case. The time series chart may hold several CPU load indexes for several tools. They can be distinguished by their colour. Since real-time, the chart changes over time.

- **What is the real-time, proportional representation?**
  This kind of representation can be a pie chart or a donut chart for example. It is good for single nominal properties such as module names and how many times they occurred as well as their proportional frequency or occurrence.

- **What is the real-time metric representation?**
  Based on the aggregation, this kind of representation provides the user with a value found from the aggregation function applied.

- **What is the real-time metric representation for nominal properties P1 and P2?**
  This kind of representation can be useful for comparing between two nominal properties such as tool names and error types. They can be placed on a heat map and the heatmap colour can represent a selected property CPU load for example.

Real-time Aggregation on Dual Property

For any aggregation A on two properties P1 and P2, from time T, for workflow node group G:

- **What is the real-time, time-series representation?**
  As explained before, this time-series can hold several properties with different line colours for comparison.
- **What is the real-time, proportional representation?**
  Two pie or donut charts can be bonded together to get an overall proportional measurement of the selected properties based on the applied aggregation.
- **What is the real-time metric representation?**
  Different metrics can be represented together beside each other.

4.4.2 Fixed-time Aggregation

Fixed-time Aggregation on Single Property

For any aggregation A on any property P, from time T1 to T2, for workflow node group G:

- **What is the time-series representation?**
  This kind of representation is similar to the time series representation discussed above but is not real-time and static based on the selected time range.
- **What is the proportional representation?**
- **What is the metric representation?**
- **What is the metric representation for nominal properties P1 and P2?**

Fixed-time Aggregation on Dual Property

For any aggregation A on two properties P1 and P2, from time T, for workflow node group G:

- **What is the time-series representation?**
- **What is the proportional representation?**
- **What is the metric representation?**

4.5 Statistical Analysis Related Questions

Statistical analysis related questions involve analyzing the provenance data in offline mode for getting certain insight. It actually depends largely on the goal, system, domain and so on. It mainly covers the data analysis that incorporates mining data using certain statistical, machine learning or deep learning tools for finding insight. Here we present several statistical analysis questions that can be important for provenance research. Please note, they may also fall under the fixed-time analysis criteria or not.

4.5.1 Statistical Analysis Questions

For a time range T1 to T2:

- **What is the similarity metric for a subgraph G1 and subgraph G2 from the complete provenance graph collection G?**
  Such queries can be used to find similar subgraphs from the full provenance graph. It can lead to finding the insight about what are the core pipelines that are being executed again and again over the time.

- **What is the cluster of nodes holding a certain condition of interest from the full provenance graph?**
  For example, based on particular analysis criteria, we may try to find a cluster of nodes. Say, those nodes are around a particular node, or similar in values and so on. This can lead to mining insight about the usage scenario and patterns.

- **What is the centrality of a workflow DAG G?**
  Such statistical measurement of graphs can be used for mining provenance graph.

- **What is the correlation between two ordinal workflow properties P1 and P2?**
  The properties of different nodes from the full provenance graph can be used to prepare a mapped dataset and find the correlation between them to discover further simple insights.

- **What is the prediction of a workflow property P in a future time point F?**
  Based on the insights, for example, found from the previous correlation question, we may predict the future CPU loads if a regular pattern is discovered. Another example could be, to predict the usage time of certain users or how much resources they might be using so that the workflow system resource management can be done in a smarter manner.

- **What is the pattern of a workflow property P?**
  Such question insight can be used to discover anomalies in any system components, unexpected behaviour or even user or security breaches.

At the moment we do not put much emphasis on this type of questions because they involve different goal and analysis that depends on the domain and problems goals. Our main goal is to create a generalized workflow programming model for featuring and enhancing provenance related features but not any particular data mining or analysis.

5 Implementation

This chapter describes the architecture of ProvMod. It also describes the logging mechanism of the model standing on a prominent research work of Cruz et al. [1]. Several technical model features are described, such as- logging levels, services, log schema that we used along with their details. Finally how the model was implemented and how it can be used for integration is also described.

5.1 System Architecture

The implementation of ProvMod is provided in Figure 5.1. The user uses a workflow through the user interface. Through the user interface, they may use a DSL to implement their workflows that stands on the ProvMod model. ProvMod stands on Python Interpreter at the core. The logging configuration can be customized by a different user who is an expert in the domain. External tools can be integrated with Provmod that can even be developed by other users. ProvMod also offers a number of tools as Library Tools. External tools and Library Tools may have their own database facilities. The ProvMod, leveraging the power of Python Programming Language and Logging Configuration, saves the logs in a Graph Database that we implemented with Neo4j [33]. Through the user interface, the user may later submit a query through a query engine. The query engine uses Cypher Query Language to parse logs from the Graph Database using Elasticsearch [34]. The query result is provided with Kibana [35] or D3 in a web browser. All the communication between each pair of components is done through RESTful Web Services [36]. We also emphasize using NoSQL database such as Cassandra [32] for Library Tools.

5.2 Logging Mechanism

Standing on the excellent workflow taxonomy from the work of Cruz et al. [1] we describe the provenance mechanism of ProvMod. It is presented in Figure 5.4. To know more about the taxonomy of Cruz et al. the reader may refer to 5.3. The taxonomy is described in the followings in details.

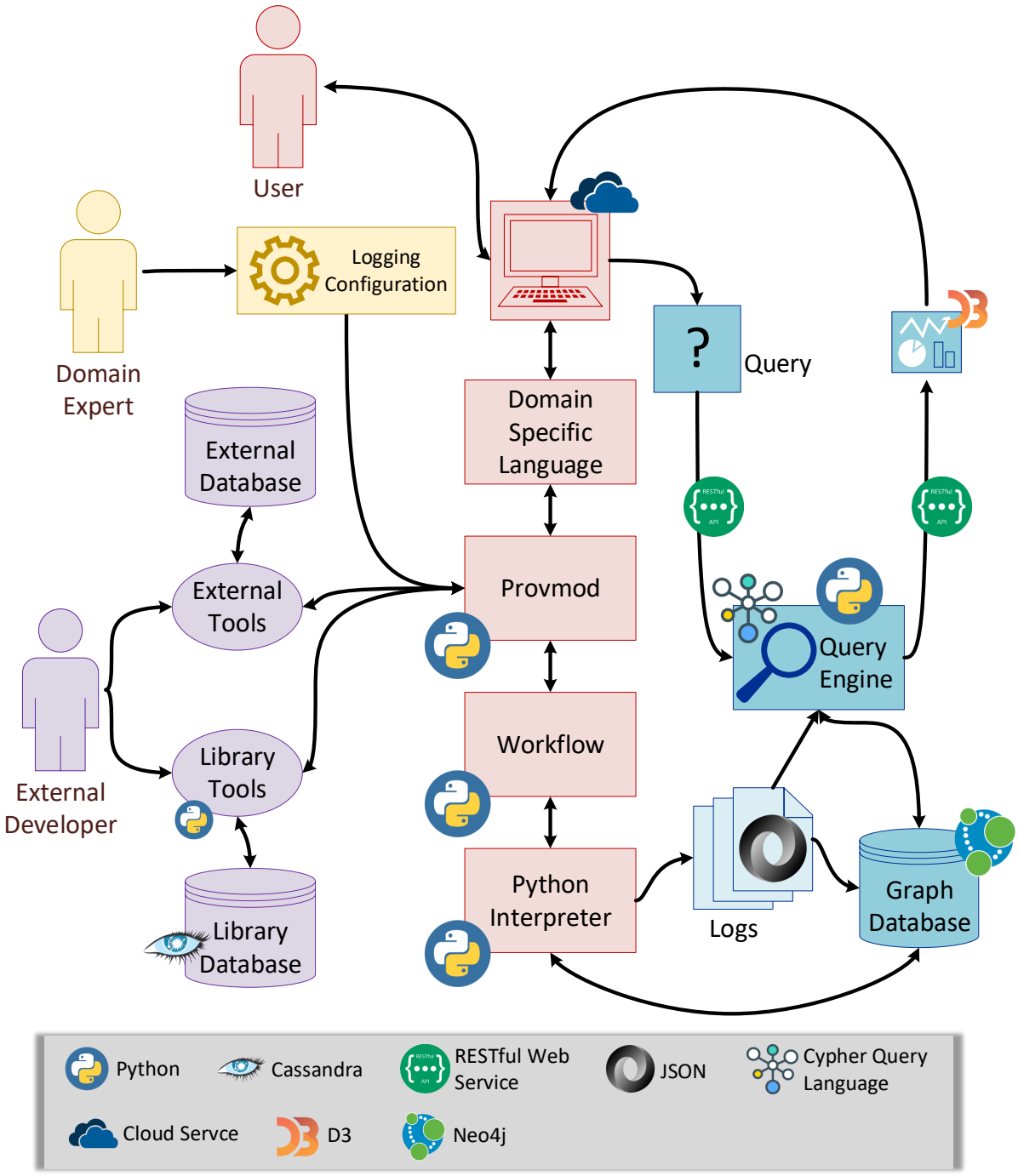

Figure 5.1: Implementation of ProvMod and relation between other components of the whole workflow system that we propose

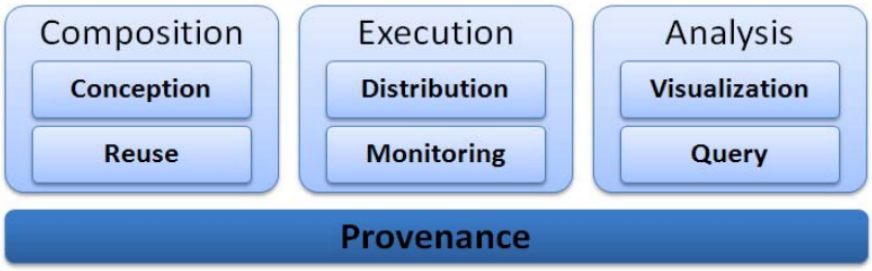

Figure 5.2: Lifecycle of workflow systems by Cruz et al. [1]

5.2.1 Taxonomy of Workflow Systems by Cruz et al.

Although the taxonomy is for workflow systems, it is oriented to classify the provenance mechanisms. There are different criteria that Cruz et al. considered. Such as capture, access, subject and storage of provenance. They are described in brief here.

Capture

Capture refers to the fashion of capturing any provenance event. They are also measured from several angles:

1. **Tracing**: The provenance may be traced in an eager or lazy fashion. The eager way is to capture the provenance immediately as the event occurs. The lazy way is to capture the provenance data only when necessary.

2. **Levels**: There are different levels of provenance information. Workflow level provenance holds information only about the workflow which is retrieved from the workflow layer. Activity provenance is about the internal information of any process. OS level provenance more information about the system being used.

3. **Mechanism**: The provenance mechanism may be externally included to any system, or it may be internally included.

4. **Technique**: Technique is classified into annotation and inversion. Annotation involves containing important information with the data items, either by the user or the system. Inversion is to recreate the data products which are expensive or inaccessible. The inversion method is used to regenerate the data items.

Access

There are several ways to access provenance. They are:

1. **Visual**: Provenance can be represented visually which is very popular now.

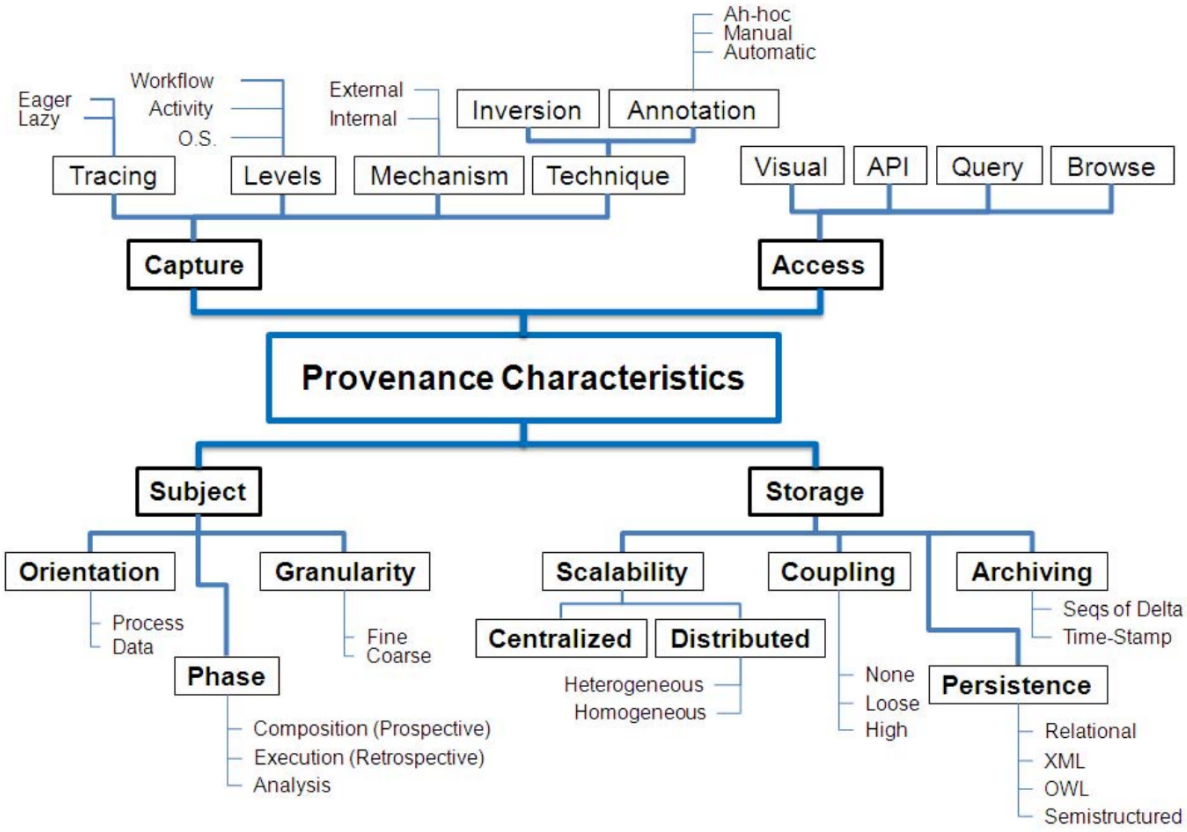

Figure 5.3: Taxonomy presented by Cruz et al. [1]

2. **API**: A system specific API can be used to access provenance data.

3. **Query**: Query can be performed over the provenance database. Also, the system can provide a way to perform queries over the database.

4. **Browse**: Provenance information can be browsed for meeting user needs through such supporting interface.

Subject

What information is being saved in provenance is a matter of concern. Based on this there are different subjects of provenance:

1. **Orientation**: Provenance can be captured for process or it can also be captured for data items.

2. **Phase**: The provenance can be specifically captured in a particular phase of a workflow lifecycle state. There are three phases of a workflow lifecycle (5.2). Composition phase refers to building any workflow and even reuse it further. Execution phase is about executing the workflow and monitoring it. Analysis phase involves visualization and querying to analyze provenance data.

3. **Granularity**: Provenance information has two granularity levels. One is fine-grained or data provenance and the other is coarse-grained or workflow provenance.

Storage

How provenance data is stored is another measurement for modern workflow systems. They are measured with the following features:

1. **Scalability**: The provenance storage can be centralized or distributed. For distributed storage support, it may support homogeneous data of a single structure or heterogeneous data of various structures.
2. **Coupling**: Provenance can be highly coupled or loosely coupled with any workflow systems.
3. **Persistence**: It is about the conceptual modelling of the system that it uses to manage workflows. Although different standards were proposed, still there is no well-accepted standard for workflow modelling. Based on the modelling, the provenance data can be expressed in various ways. They can be relational, XML based, OWL based or semi-structured.
4. **Archiving**: How provenance data is being stored is the matter of this measurement. Provenance data can be captured immediately as the original log data which is known as the time-stamping approach. Another approach is the sequence of Delta. This approach captures the changes and version information between two consecutive provenance data states. So, later on, the difference between any two states can be used to derive the target provenance information.

Based on this taxonomy, we describe the provenance mechanism of ProvMod in the following section.

5.2.2 Description of ProvMod Provenance Mechanism

Standing on the taxonomy of Cruz et al. [1], we describe the taxonomy of our proposed programming model in Figure 5.4. For a quick look, a brief description of each mechanism category of ProvMod can be found in Table 5.1.

5.3 Workflow Components

There are several component types (Figure 5.5) that are already implemented and supported by the ProvMod while implementing any pipelines. The components are:

- **Data**: Data is a superclass type that has some basic features to represent any data item in the workflow.
- **Object**: An Object is a class of data that is inherited from the Data class. Object class is used to represent any data item in the workflow using any direct value. For example, an integer, string, number, a list, tuple and so on that comes with the programming language the developer is using to

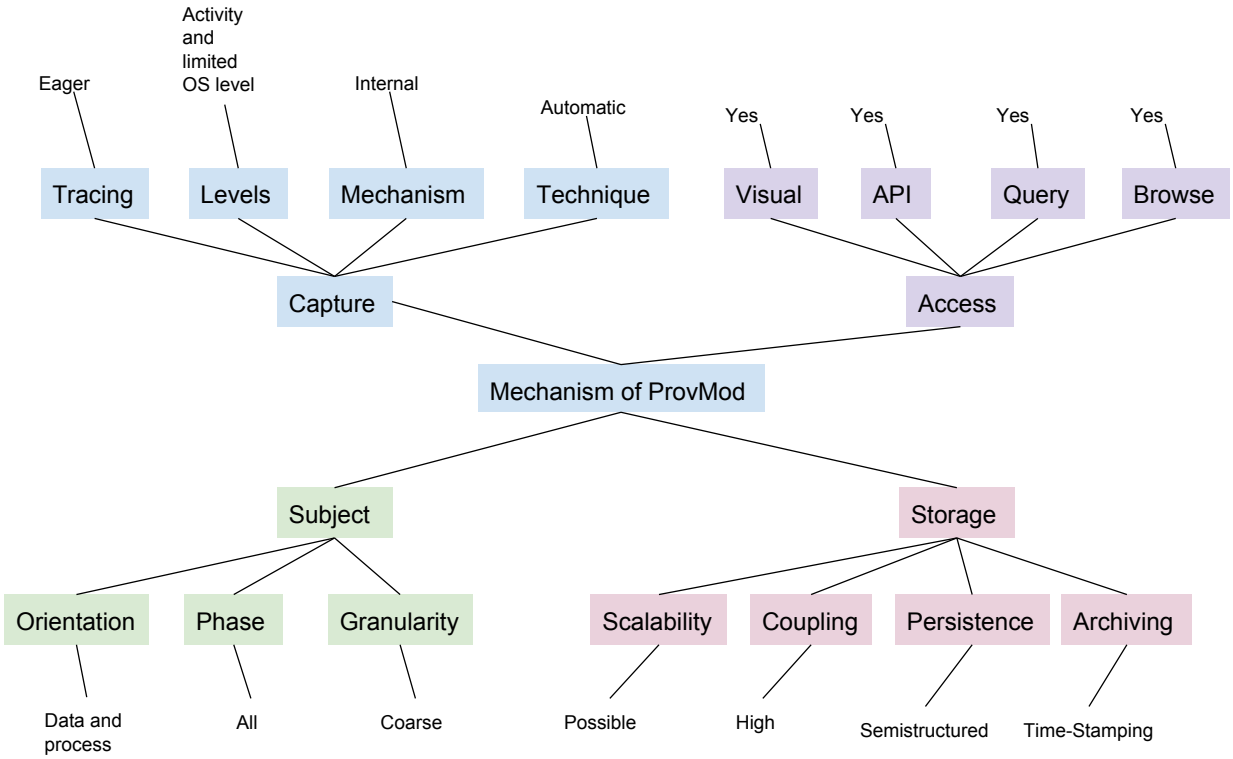

Figure 5.4: Description of ProvMod based on the taxonomy of Cruz et al. [1]

implement ProvMod. Primarily we use Python and it can work with any data types from Python programming language. This class is very useful for representing intermediate data items that are created in the middle of workflow runs and removed from the memory when the system exits the workflow environment.

- **File**: A File class is used to represent a file object in the workflow. File class does not hold a file itself but a file lookup reference to read the file or to write a new file in that reference. It is useful for representing data that are saved in the disk.

- **Document**: Any NoSQL data item is represented with a Document class. It is similar to File class but has a lookup reference to read data from a database system or to write in it.

- **Module**: Module is a class to represent any tool that takes Data items as inputs and creates Data items as outputs.

- **Edges**: There are two types of workflow edges to represent the data flow. INPUT and OUTPUT. INPUT represents a data flow from any data item into a Module and OUTPUT represents a data flow

from any Module to new data items.

- **Properties**: There are several properties that are incorporated with the workflow components. They are considered as workflow node properties and can be used for graph query to real-time monitoring. The list and meaning of each property can be found in Table 5.2, 5.3, 5.4, 5.5, 5.6.
- **User**: User is a class in the workflow model ProvMod but is not directly used by the developer but is automatically generated when the model is used.

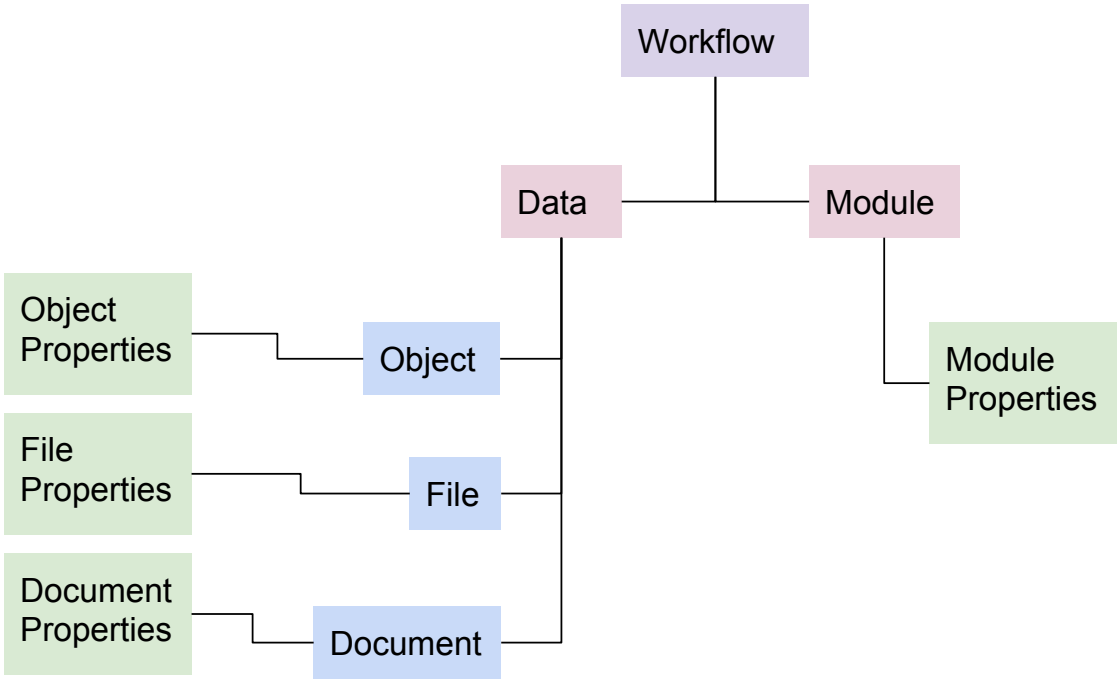

Figure 5.5: ProvMod model components

5.4 Logging Levels

Different logging levels are supported in ProvMod that can also be tuned and even disabled or enabled according to the will of the user. The logging levels are dependent on the base programming language that is used to implement ProvMod. Here we used Python as thus have the following logging levels.

1. **INFO**: Info is a coarse level logging level. With this level, we will be analyzing any function or main module (a python script that is directly run) but will not be looking into their internal details. So, any

function or main scope is considered as a black box in this log level. We will get information about inputs, parameters and outputs. Look at the figure that makes clear about the points where to use Info.

2. **DEBUG**: Debug is more detailed than Info. We will look into more details like, how the parameter values are affecting, how the internal states are being changed etc. So, any log level that is deeper than Info will be of Debug level.

3. **WARNING**: If the developers think there might be a problem in any place of the program but that is not creating an issue, then they could provide a warning levelled log with an appropriate message.

4. **ERROR**: Error level log is for that kind of error that properly cared with exception handler by the programmer. So, it occurs often times but does not make the program to abort.

5. **FATAL**: Fatal errors are those errors that make the program to abort immediately.

5.5 Services

The ProvMod model, use different services to accomplish certain tasks. Different components are kept separate. The components communicate with each other using RESTful Web Service.

The automated log generation is done with the configuration that is also a separate service. The model makes use of it during any workflow execution.

External tools, any related databases involving them are considered as independent and separate services. Tools that ship with the model are also separated by design and can be considered as distinct services.

The provenance database is not to be confused with the tool library database or any external tool database. They are important for their operation where the provenance database is responsible for holding provenance and workflow history. That is built on Neo4j Graph Database. So JSON is easily obtained here. The service must be ready before the workflow run and any feature that comes with Neo4j can be used to analyze the graph data later on without even using the model.

Log parsing and real-time monitoring is done by Elasticsearch. Elasticsearch has its own way of log parsing and management in a cluster system. In our work, we use the default user mode of Elasticsearch and do not focus on any distributed data management rather than the model. Easily, Elasistacsearch is configurable in an HPC environment, and thus such HPC features will be shipped to the model if the developer wants. Elasticsearch being a separate service, it must be ready before any workflow runs for log parsing. Note, logs are parsed both to Neo4j during workflow run as provenance is highly coupled in our model as well as to Elasticsearch index. The reason behind this is to enable real-time monitoring. On the other side, Kibana is another front-end service for the user that communicates with Elasticsearch and offers real-time monitoring. Kibana should also be ready before workflow execution.

```
{
  "time": "event time",
  "file": "executed file name",
  "func": "executed function name",
  "level": "logging level",
  "line": "executed line during logging",
  "module": "script name",
  "msecs": "milisecond part of time",

  "msg":{
          "event": "predefined event category",
          "id": "event id",
          "name": "user name",
          "time": "time of event",
          "type": "data type of data node",
          "value": "value of object node",
          "user": "user id",
          "memory": "memory load",
          "cpu": "cpu load",
          "label": "node type",
          "error": "compiler error message",
          "source": "source of file node",
          "address": "database document address",

          "p@": ["list of parameter ids"],

          "memory_init": "initial memory load",
          "cpu_init": "initial cpu load",
          "duration_init": "module initialization duration",
          "cpu_run": "cpu load of module execution",
          "memory_run" "memory load of module execution",
          "duration_run": "module execution duration",

          "o@": ["list of module output ids"],

          "time_run": "module execution start time"
        }

  "logger": "logger class name",
  "path": "executed script full path",
  "procID": "process id",
  "proc": "process name",
  "threadID": "thread id",
  "threadName": "thread name"

}
```

Figure 5.6: JSON log structure

5.6 Log Schema

The log is in JSON format follows a JSON nested data structure.

Here we present the JSON structure in Figure 5.6. Note, the JSON schema is presented here is for file and not exactly same as used in Elasticsearch. In Elasticsearch we use even a reduced version of it and only the necessary properties for analysis and experiments.

5.7 Workflow Implementation

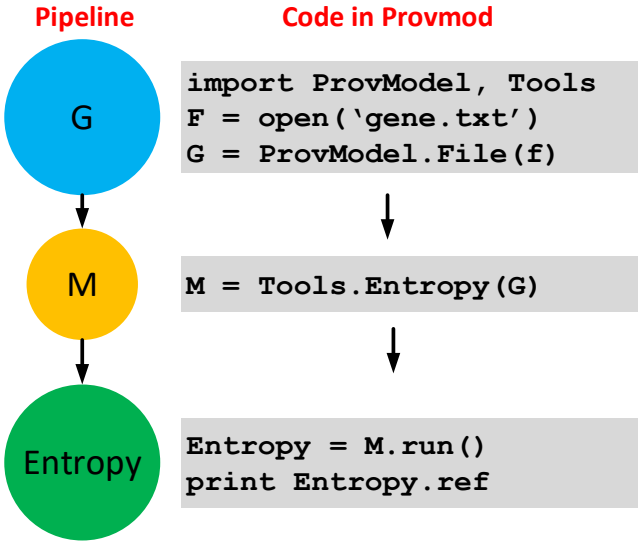

Figure 5.7: Implementing a workflow

Although not a direct workflow system, ProvMod can be used to implement workflows very easily. The technique is illustrated in Figure 5.7.

Simply we import the model as a Python library. The tools are also imported from the library. For any file from the disk or other sources, we create an instance of File class from the model. Any tool that we will be using, is taken from the tool library as a class. An instance of the target tool class is created to use the tool. During tool instance creation, you initialize it with the specified input items. The tool invocation is made with the ***run()*** command. Any tool invocation can return you with a number of objects that are inherited from Data class in the model. Any data item's data value can be read with the ***ref*** variable in our model. There are other log properties that are automatically handled.

5.8 Tool Development and Integration

To develop a tool the reader may refer to the detailed documentation in `https://github.com/rayhan-ferdous/Provmod/wiki`. Here we present a simple and reusable template that can be used to implement and integrate tools in workflow systems enabling ProvMod (Figure 5.8). An example of the template usage is discussed in Chapter 11. There, the reader can also get examples of implementing existing tools in ProvMod.

<TOOL_NAME>.py

```
def <FUNCTION_NAME>(in1):
    f = open(in1)
    out = open(str(<OUTPUT_FILE_NAME>), 'w')
    <FUNCTION_LOGIC>
    return out.name

class <TOOL_CLASS_NAME>(Module):
    def body(self):
        res = <FUNCTION_NAME>(self.P[0].ref)
        return Object(res)

d1 = Object(sys.argv[1])
d2 = <TOOL_CLASS_NAME>(d1).run()
print d2.ref
```

Figure 5.8: Tool development and integration template

Table 5.1: Provenance mechanism of ProvMod

Category	**Mechanism**	**ProvMod Behavior**	**Explanation**
Capture	Capture	Eager	Provenance is logged immediately
	Levels	Activity and limited OS level	ProvMod can trace OS level activities through permission and custom configuration
	Mechanism	Internal	Logging is internally fused with ProvMod
	Technique	Annotation	Capture is done in annotation fashion for each data items
Access	Visual	Yes	Visualization supported
	API	Yes	API development and integration supported
	Query	Yes	Querying through Cypher supported
	Browse	Yes	Browsing logs is supported through Elasticsearch frontend
Subject	Orientation	Data and Process	Both data and workflow steps are captured
	Phase	All	Provenance is captured in all phases of provenance lifecycle
	Granularity	Coarse	Coarse granularity rather than fine grained
Storage	Scalability	Distributed through the support of relevant tools	Distributed systems can be integrated as external tools to support scalability
	Coupling	High	Provenance is highly coupled with ProvMod
	Persistence	Semistructured	Graph Database support
	Archiving	Time-stamping	Logs are accompanied by their own timestamps

Table 5.2: User Properties

Component	Property	Information
User	id	Holds an unique ID for identification
	name	Holds the user name
	event	Holds the user invocation message USR-INVK
	time	Holds the time in UTC when the user was created

Table 5.3: Object Properties

Component	Property	Information
Object	id	Holds an unique ID for identification
	user	Holds the user ID
	event	Holds the user invocation message USR-INVK
	time	Holds the time in UTC when the user was created
	value	Holds the data item value
	memory	Holds amount of RAM used during creation in MB
	CPU	Holds amount of CPU used during creation in %
	label	holds the message "object"
	error	Holds either "success" or the error message OCE

Table 5.4: File Properties

Component	Property	Information
File	id	Holds an unique ID for identification
	user	Holds the user ID
	event	Holds the file creation message FIL-CRTN
	time	Holds the time in UTC when the user was created
	value	Holds the data item value
	memory	Holds amount of RAM used during creation in MB
	CPU	Holds amount of CPU used during creation in %
	label	holds the message "File"
	error	Holds either "success" or the error message FCE
	type	Holds the type of file that should be "file"
	source	Holds the path of the file as a string

Table 5.5: Document Properties

Component	Property	Information
Document	id	Holds an unique ID for identification
	user	Holds the user ID
	event	Holds the document creation message DOC-CRTN
	time	Holds the time in UTC when the user was created
	memory	Holds amount of RAM used during creation in MB
	CPU	Holds amount of CPU used during creation in %
	label	holds the message "document"
	error	Holds either "success" or the error message FCE
	type	Holds the type of the document, should be "document"
	address	Holds the reference address of the document as a string

Table 5.6: Module Properties

Component	Property	Information
Module	id	Holds an unique ID for identification
	user	Holds the user ID
	event	Holds the module creation message MOD-CRTN or run message MOD-RUN
	time	Holds the time in UTC when the user was created
	memory_init	Holds amount of RAM used during creation in MB
	cpu_init	Holds amount of CPU used during creation in %
	label	holds the message "module"
	error	Holds either "success" or the error message MIE or MRE
	p@	Holds the parameter list
	name	Holds the source name
	memory_run	Holds the RAM usage for Module run
	cpu_run	Holds the CPU usage for Module run in %
	duration_init	Holds the duration of time to create the Module
	duration_run	Holds the duration of time to run the Module
	o@	Holds the outputs as a list

6 Experimental Analysis

In this chapter, we analyze the model for three different kinds of workflows. The first one is a sparse workflow, then a large workflow and finally a deep workflow. Here we introduce them briefly:

1. **Sparse Workflow**: By sparse workflow, we refer such workflow that has different smaller workflows in the workflow space. The different workflows are relatively very small (for example having at most 10 nodes) and not necessarily connected to each other. So, simply sparse workflows are a collection of small disconnected DAGs in a workflow graph space.

2. **Large Workflow**: By large workflow, we refer to such workflow that is very large in a graph level. That means a node may have thousands of child nodes which expands the full workflow but still keeps the workflow's depth shallow.

3. **Deep Workflow**: By deep workflow, we refer to such workflow that is deep according to the graph level. That means the workflow has continuous trailing child nodes one after another. This way, the workflow becomes a very long linear DAG.

For each kind of workflows we just mentioned, we implement and run them iteratively to increase the provenance graph size. The sparse workflow is used to understand the performance behaviour of a random usage scenario of the model for workflow implementation. Large workflow is implemented to specifically see, what are the effects of graph level expansion on the model. Finally, the deep workflow is implemented to see the effects of graph depth on the model.

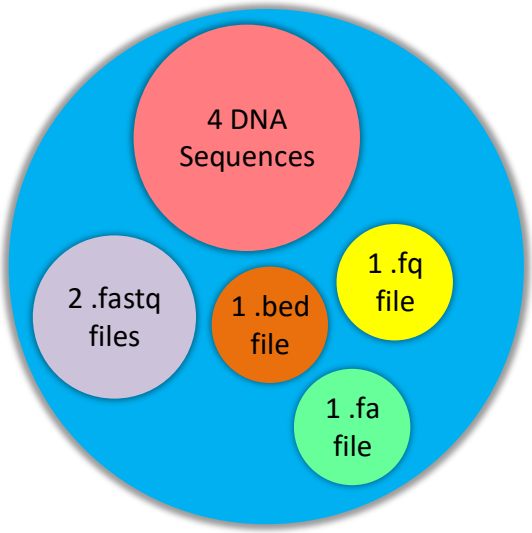

Figure 6.1: Experimental dataset overview

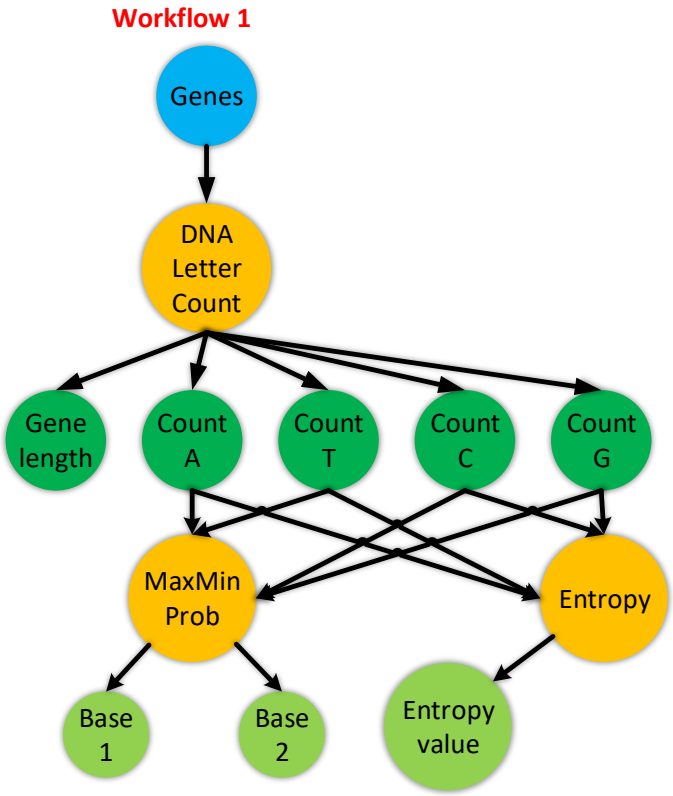

Figure 6.2: First workflow for simulation

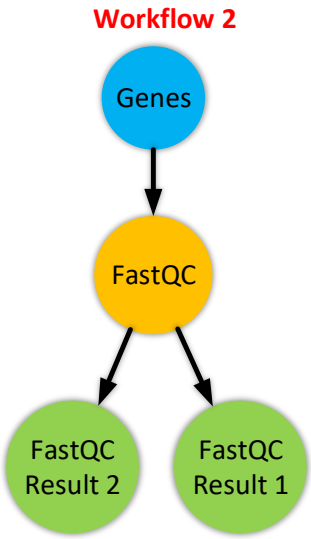

Figure 6.3: Second workflow for simulation

6.1 Sparse Workflow

We run a simulation that has several tools, data items and also such items that can generate errors in some of the times during the simulation. It helps us to populate log data and analyze the monitoring facility in real time. We also use this simulation for user study. The purpose of this simulation is to create a simulated usage pattern quickly and analyze the logs for the study.

We implement two different workflows from Bioinformatics. The first workflow is about counting DNA letters from a genetic dataset. We also count the length of the gene. Based on the nucleotide base counts, we can calculate the Entropy of the gene sequence. We also find out, which base is having the most and least probability of occurrence over the full sequence to get an understanding of the full sequence along with its entropy. In the second workflow, we run FastQC [37] over the datasets to generate FastQC results. Note, in the simulation, a collection of genetic data is used that is described in Figure 6.1. The FastQC files are only valid inputs for FastQC and generate error otherwise. In the simulation, to generate a user-oriented usage scenario, we choose a data randomly from the dataset to input through the workflows. Also, from the first workflow, only DNALetterCount is executed or DNALetterCount with Entropy is executed or DNALetterCount with Entropy and MaxMinProv tools are executed. Otherwise, FastQC is executed with random inputs. They are selected randomly. Between the executions, there is a random gap of 1 to 7 seconds. From the simulation, we create a provenance graph of around 20,000 nodes that contains logs about FastQC errors too. A portion with 300 nodes from the provenance graph is shown in Figure 6.4.

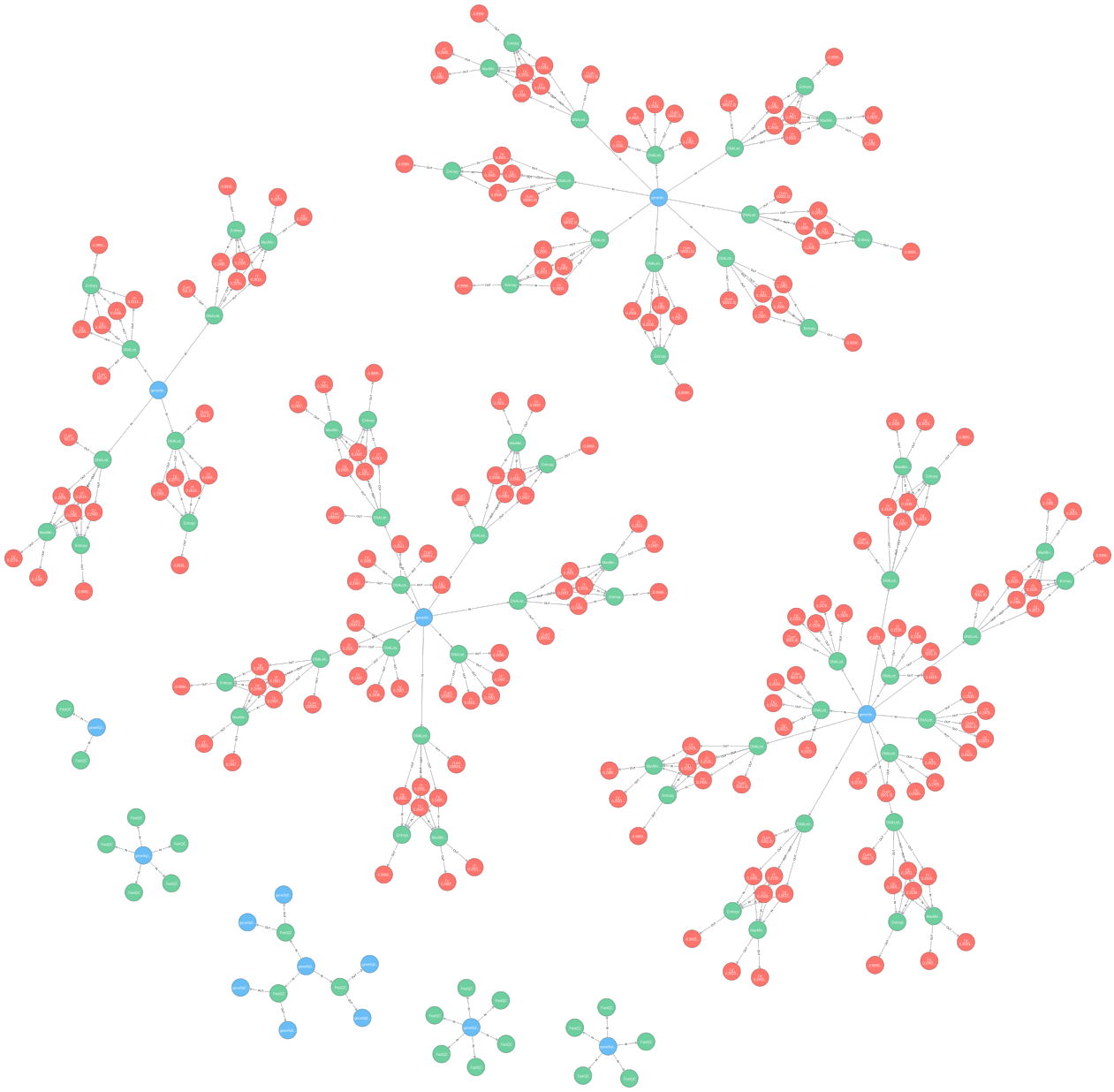

Figure 6.4: A portion of the provenance graph with 300 nodes generated by the simulation with sparse workflow. Green nodes are different modules with module name as label. Red nodes are intermediate Data Object items with data value as label. Blue nodes are Data File items with file path as label. The edges are labeled with IN or OUT. IN is directed from data to module and OUT is directed from module to data.

6.2 Large Workflow

As mentioned earlier, we implement a workflow in this phase to see the effect of workflow graph level expansion on the model. We select a plant phenotyping image analysis workflow. This workflow has a single module named Sharpen. Sharpen module takes a phenotyping image as input and applies an image processing filter to sharpen the image. This creates another image as output. So, simply the workflow has one input, one module and one output to sharpen an image. The Sharpen module has a filter level parameter that specifies how strongly the filter will be applied. The large workflow is shown in Figure 6.5.

Iteratively, we use the same input image data item and only one instance of Sharpen module with a range of filter level parameter value. This lets us use only one input image and one module in a workflow to generate a number of output images for different settings. Such a scenario is very useful to investigate the distinct results of different settings and also to save them separately. The results can be used to select the best workflow setting for such scenarios. Note, in our model, we are applying the same module again and again here but that is still not creating any cycle or loop in the workflow. It is because ProvMod creates the provenance graphs as DAGs and separate execution information is logged as property values. The workflow is executed for thousands of times to create a reasonable number of nodes in the provenance graph. A snapshot of the provenance graph after simulating this scenario with large workflow is in Figure 6.6.

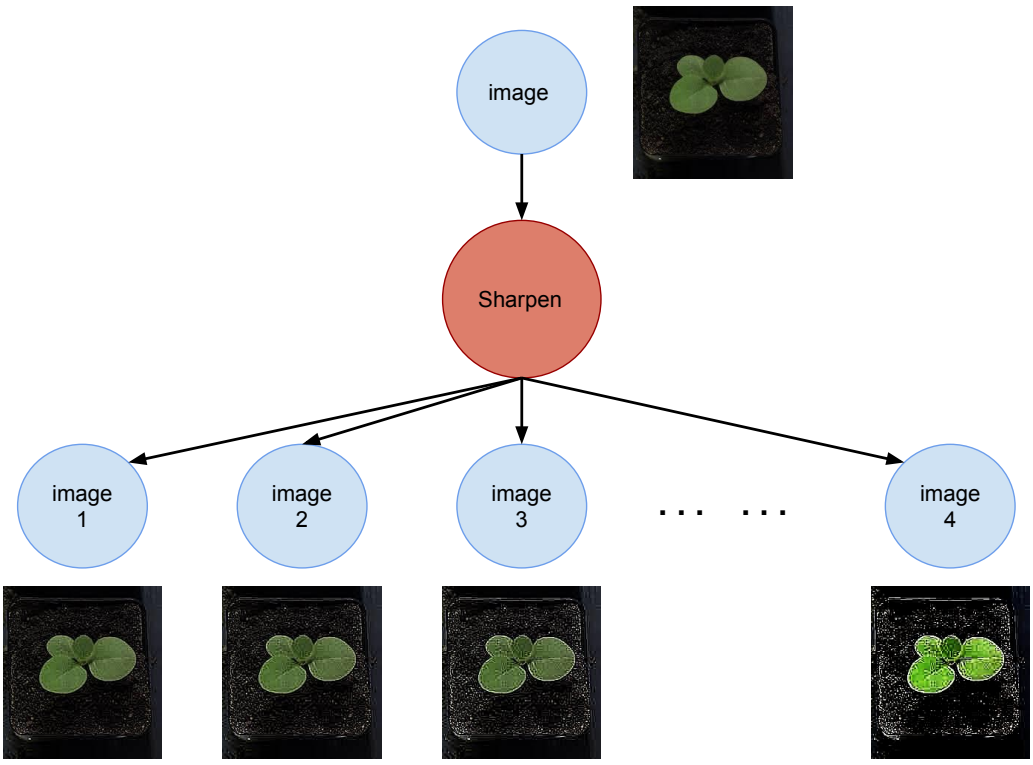

Figure 6.5: The large workflow used in our experimental analysis

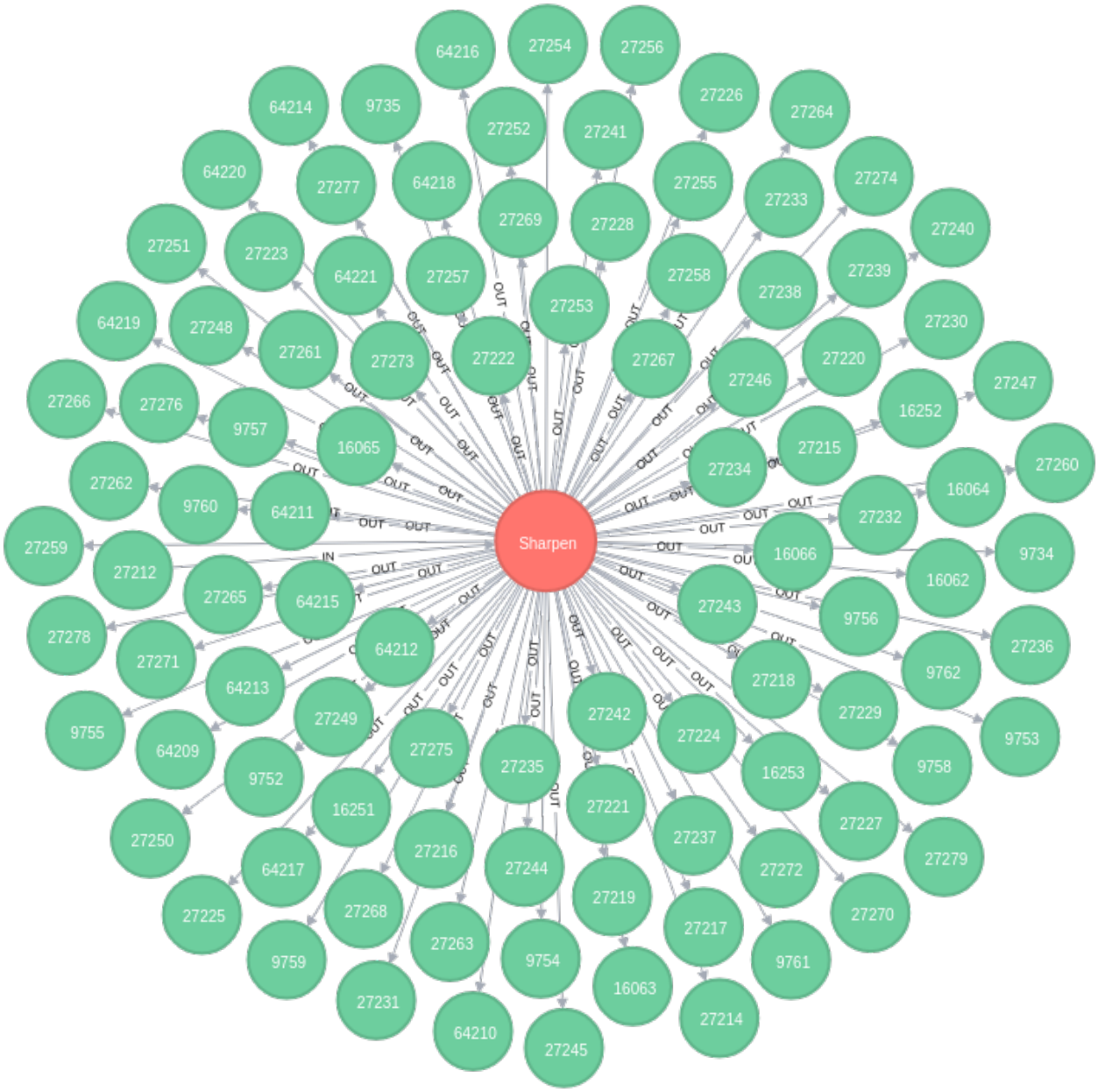

Figure 6.6: A portion of the provenance graph with 100 nodes generated by simulation with large workflow. The red node is the Sharpen module labeled with tool name and all other green nodes are generated output data from Sharpen. Green data nodes are labeled with an ID. There is only one input to Sharpen in the full graph that is not highlighted here. The graph is expanded around the single Sharpen node only.

6.3 Deep Workflow

The deep workflow we implement consists of the same basic data item and module Sharpen from the previous section. In this case, we use a module instance only once with the same filter level parameter setting. The only difference is, the output of the previous run is used for the next run. This creates a linear pipeline with repeated module invocations. The workflow is shown in Figure 6.7.

This kind of workflow is useful when an analysis needs multiple application and the complete analysis run takes much time. So, the user can create all the intermediate results. Any intermediate result until the last result or an error can be used anytime. The module setting remains fixed in this case, that can be investigated to observe data propagation. This workflow is also executed thousands of times to create a large number of nodes in the provenance graph.

In figure 6.8, we present a provenance graph portion generated by simulation with deep workflow.

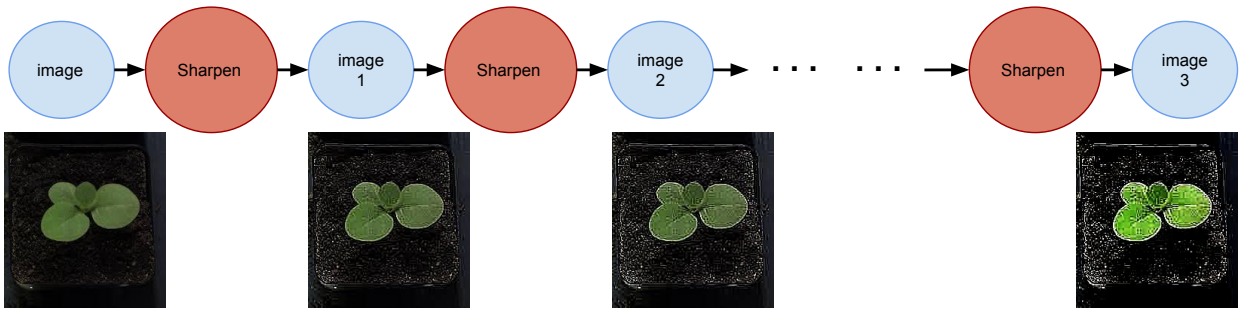

Figure 6.7: The deep workflow used in our experimental analysis

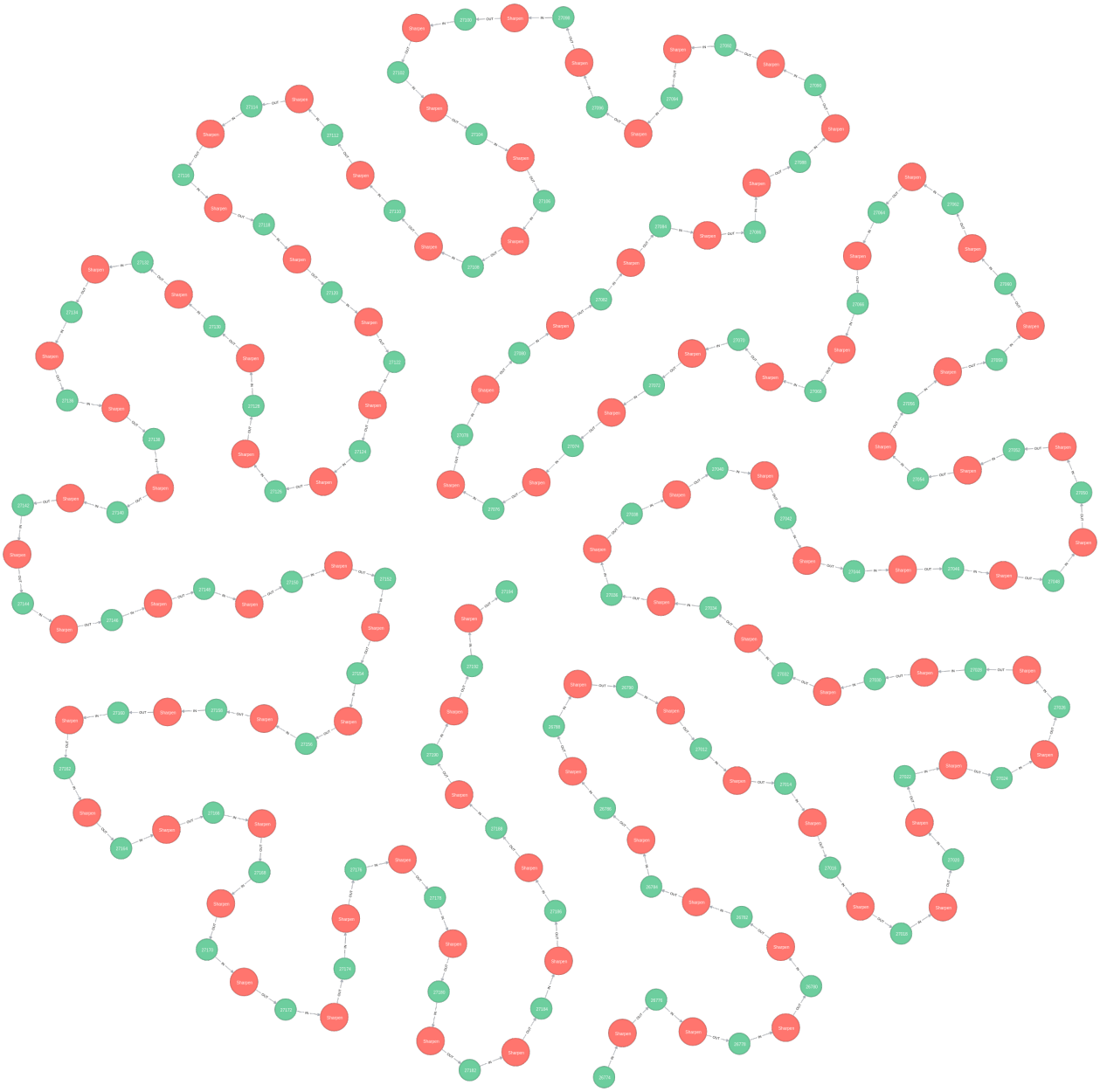

Figure 6.8: A portion of the provenance graph with 100 nodes generated by simulation with deep workflow. The red nodes are the different instances of Sharpen tool with same parameter setting. Green nodes labeled with IDs are either an input or output data items. One generated data item is used as input in the next module.

6.4 Performance Analysis

For the analysis, we use a system with Ubuntu Linux 16.04 LTS, 16 GB DDR4 memory, Intel Core i7-7700HQ CPU at 2.80GHz x 8 CPU, 256 GB SSD as secondary memory.

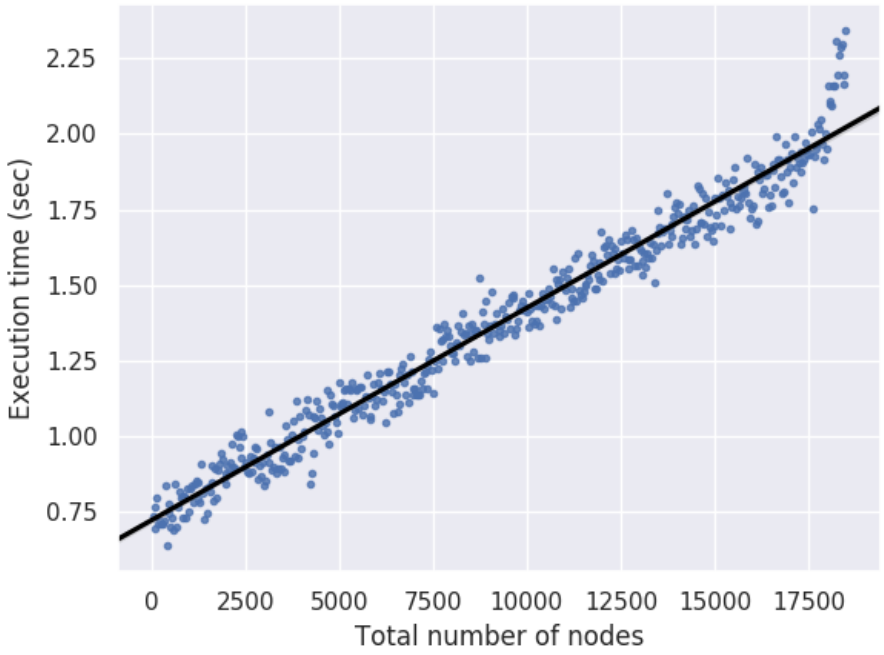

Figure 6.9: Execution overhead of sparse workflow

We analyze several things about our ProvMod model. The analyses are described below with the found insights:

1. As our model is fused with the Graph Database, querying and adding nodes is always happening throughout the execution. It can be easily turned off, but we are eager to see how much performance overhead is occurring for that. In Figure 6.9, we can see, when there are around 20,000 nodes in the simulated provenance graph, the tool execution time is around 2 seconds. In real-world workflows, a workflow may contain around 10 nodes or so, but never thousands of nodes for a single user. So, our ProvMod model shows good performance.

2. The query time will also increase as the graph is increased (the sparse workflow). We search the full graph during the simulation at the end of every single simulation step and capture the time to collect the full graph. We find that the full graph search is returned within 1.5 seconds when there are around 20,000 nodes in the provenance graph in Figure 6.10. This clarifies that our ProvMod model query is not time-consuming and fast.

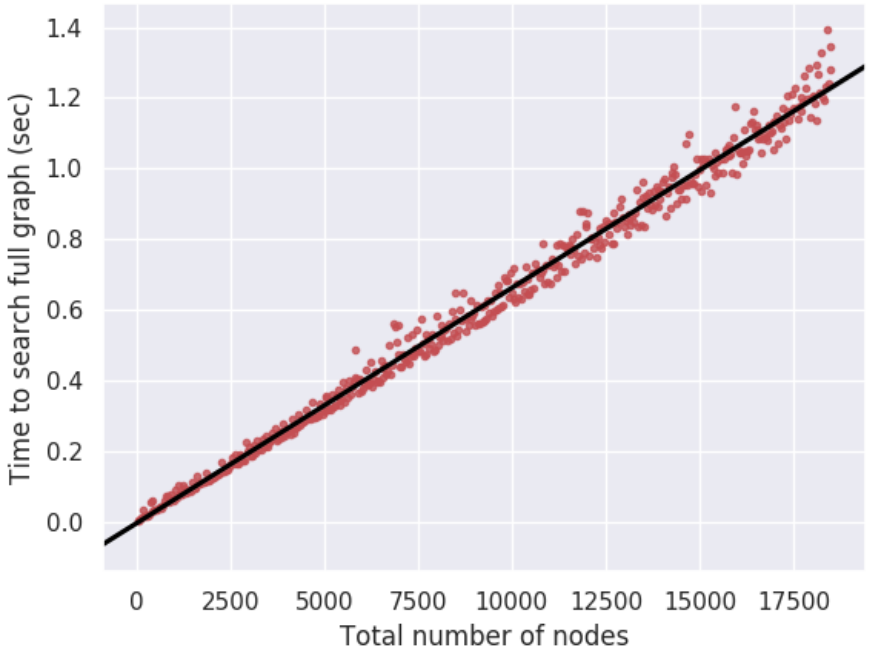

Figure 6.10: Query overhead of sparse workflow

3. Finally, we compare the whole simulation with and without provenance and measure the performance overhead the ProvMod logging is actually creating that we present in Figure 6.11. It becomes clear that for such a huge graph with 20,000 nodes the provenance creates an overhead of around 0.5 seconds on average.

4. From the found insights and analysis results from the sparse workflow, we apply the same technique on the large workflow and deep workflow. The large workflow shows whether there is any effect on ProvMod performance for any provenance graph level expansion. On the other hand, the deep workflow shows whether there is any effect of provenance graph depth on the model. We can also compare the results between them.

5. Figure 6.12 and 6.14 shows the execution overhead due to number of nodes. We can see, there is a little execution time difference for the same number of nodes in them. But that both are having more execution overhead than the sparse workflow (Figure 6.9). From this observation, we imply that the execution time could be increased due to the number of connected nodes because the sparse workflow has more disconnected nodes. Though this implication is not very strong because the sparse workflow has error generating nodes in it too.

6. Figure 6.13 and 6.15 clearly shows that the query overhead is heavily increased in large workflow and deep workflow than sparse workflow (Figure 6.10). Here, the query time only depends on the number

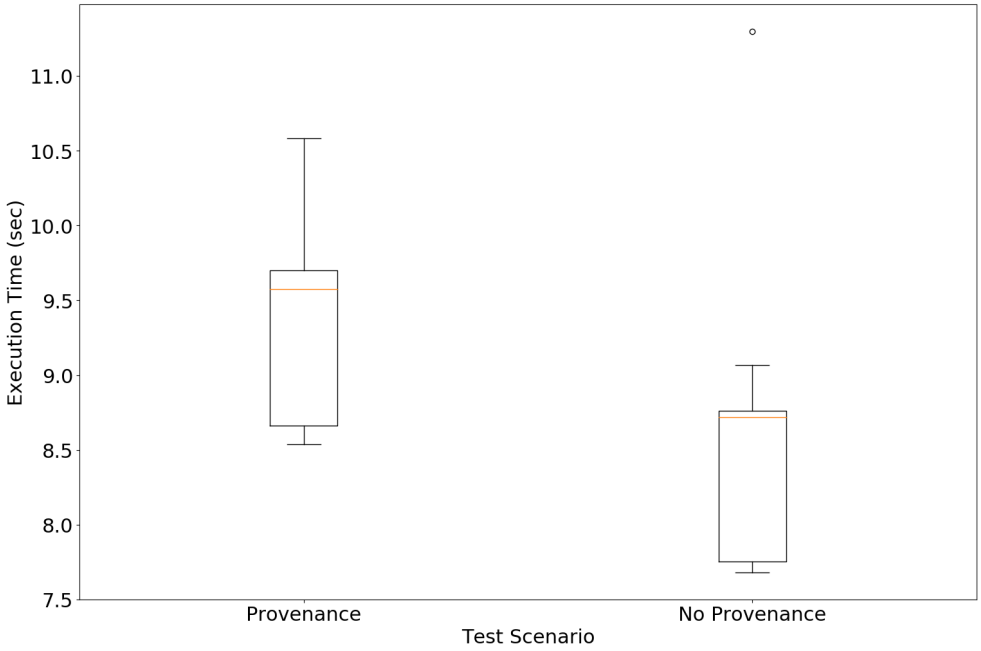

Figure 6.11: Comparison of ProvMod execution time with and without provenance

of nodes in the provenance graph. So from this observation, we strongly imply that the query overhead increases due to the number of connected nodes. Although, we do not see much difference between the query overhead of large workflow and deep workflow.

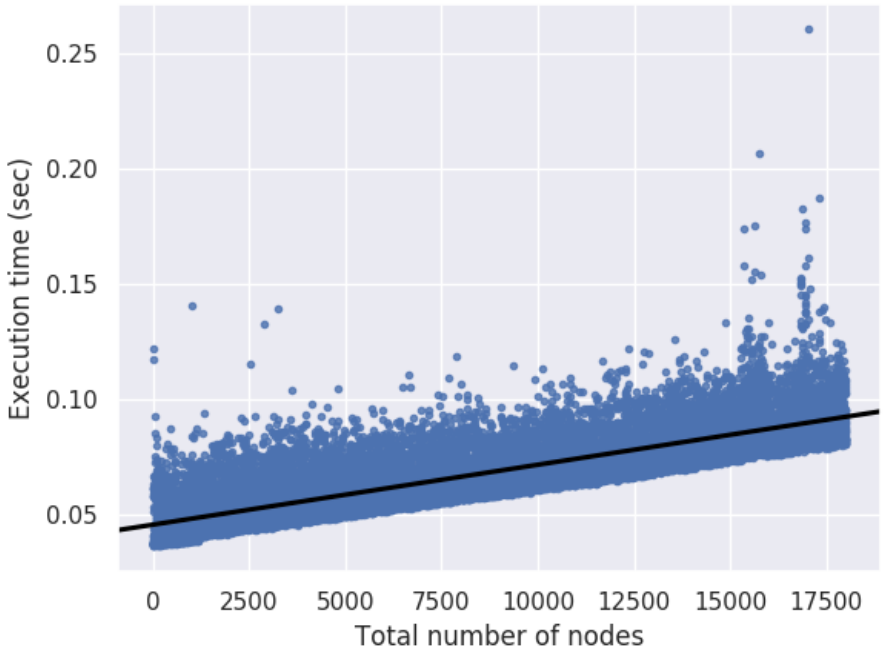

Figure 6.12: Execution overhead of large workflow

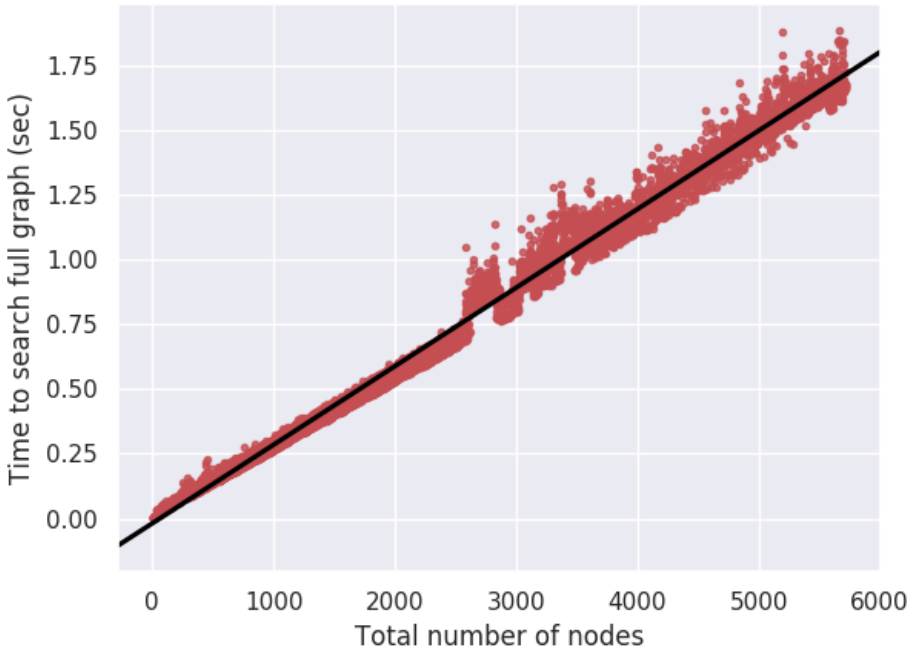

Figure 6.13: Query overhead of large workflow

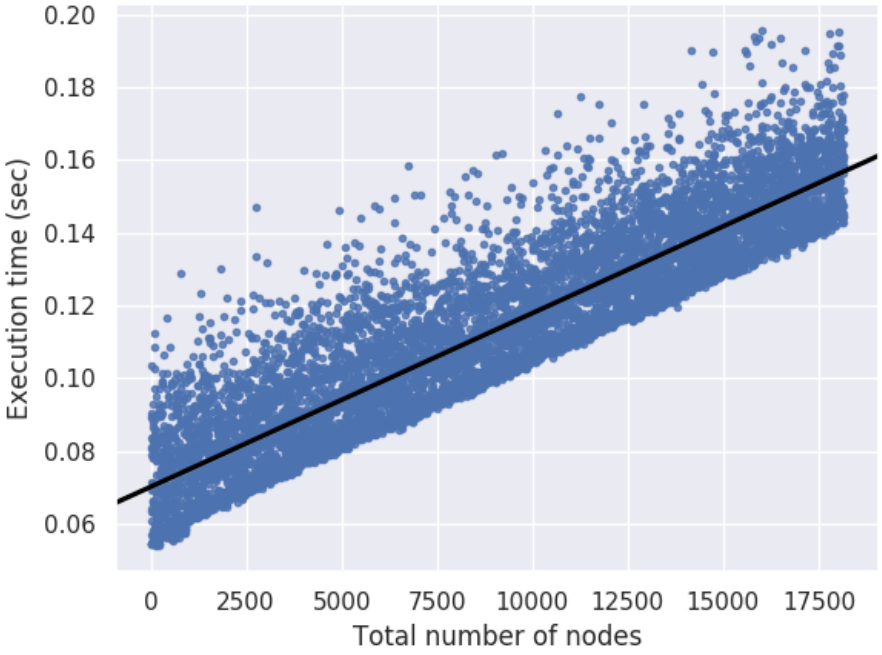

Figure 6.14: Execution overhead of deep workflow

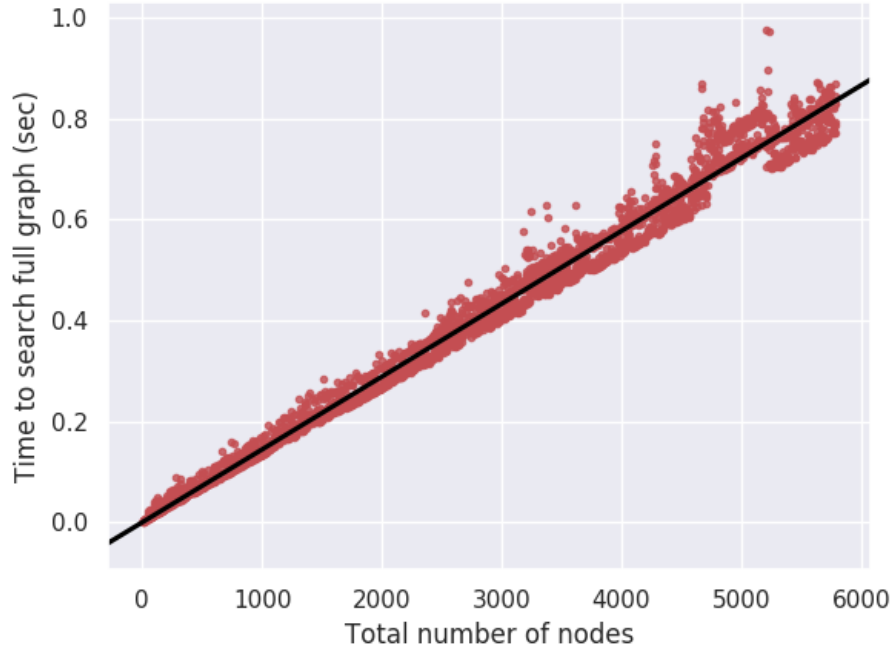

Figure 6.15: Query overhead of deep workflow

7 Reporting

Based on the experimental analysis done in the Experimental analysis chapter, we can now populate a provenance graph with various nodes and properties in them. Here in this section, we try to answer the provenance question we mentioned in an earlier provenance questions chapter. To answer the questions, we use data visualizations. For example, to take advantage of existing technologies to visualize a scenario that can answer a provenance question as well as some other basic questions too that are already in the visualization.

Note that our goal is not to build a better chart or way of visualization but to use the existing methods to visualize them in a good manner. The best way of visualization findings is another research topic and can be incorporated with our work in the future. We try to attach a visualization service with our model since, in the existing research works, the importance of visualization in provenance is stated. We create a number of visualization representation to provide a dashboard as a service to the user. It makes the ProvMod model to leverage the power of reporting for real-time monitoring for the end user.

7.1 Visualization for Graph Related Questions

In this section, we present a graph visualization using Neo4j technology and the ProvMod model's notation that can be used to answer a number of graph related provenance questions. The graph related questions mentioned in earlier formal provenance questions chapter is more abstract and general. Here from Figure 7.1, we present an example that is already implemented in our work and can be used as simple examples. From the graph representation, it is possible to answer these questions:

1. What is the full provenance graph?
2. What is the largest pipeline/workflow?
3. What are the inputs of a module?
4. What are the outputs of a module?
5. What are the anomalous modules with no input and output?
6. What are the anomalous modules that have only inputs but no output?
7. Are there any similar pipelines co-existing in the graph?

8. Are the pipelines similar or nearly similar?
9. What is a property value of a selected node?
10. What the Time Sequence Mapping for a node type T for property P?
11. What is the Data Sequence Mapping for a node type T for property P1 and P2?
12. What is the pipeline path from selected nodes n1 to n2?

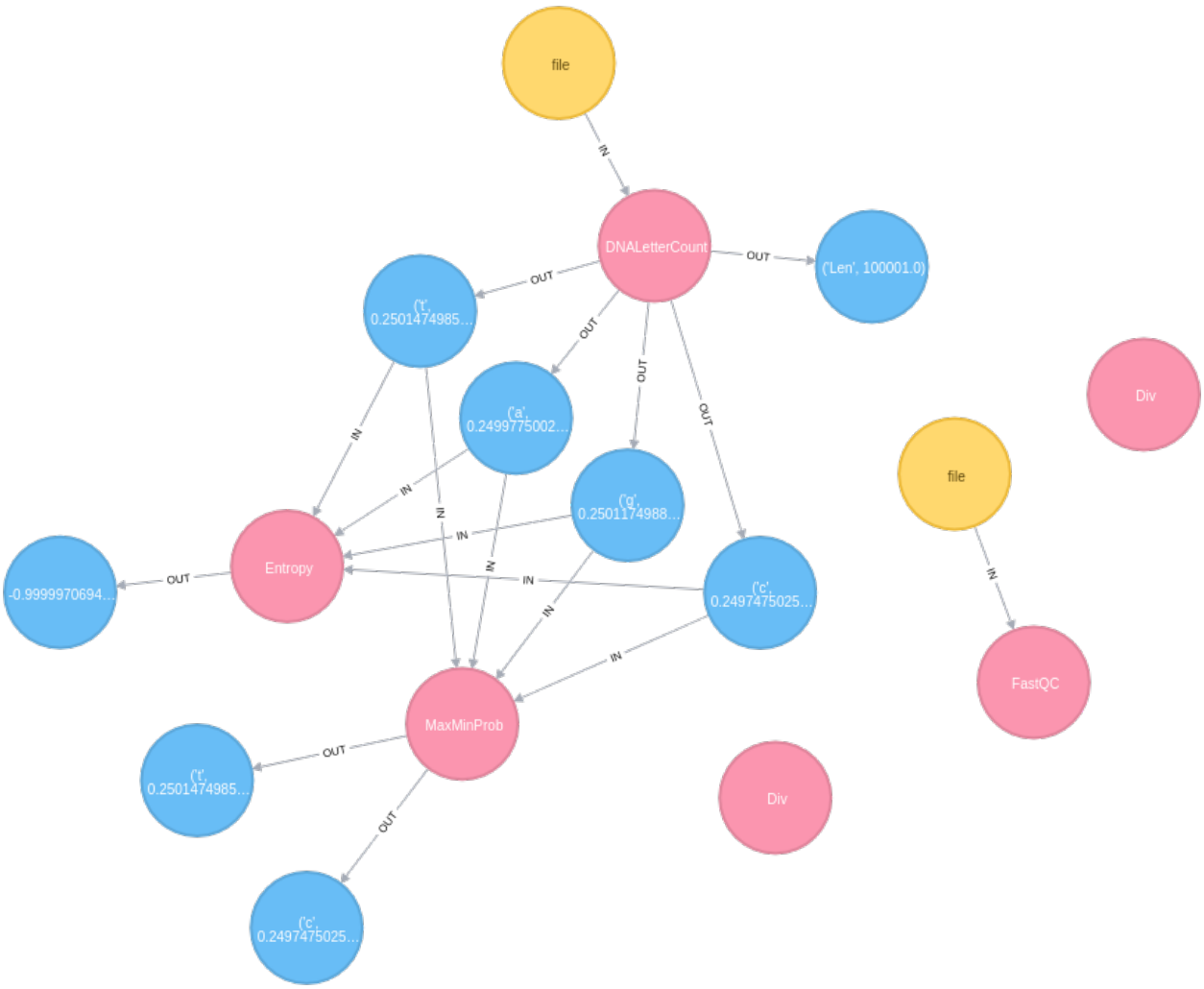

Figure 7.1: A graph visualization to answer graph related question that does not return a single metric value, Objects are Blue, Modules are Red and Files are Yellow in color

7.2 Visualization for Time Series Monitoring Questions

In this section, we present a real-time or fixed time, time-series visualization technique using the Kibana technology. The time-series visualizations can be used to answer several questions from the monitoring

questions criteria:

1. What is the target property (here CPU load) value at time T for a module target M?
2. What is the target property value now for a module?
3. What is the maximum/minimum target property value within a time range for module M?
4. Is there any anomalous property value for a target property and module M (for example sudden zero value in the X axis)?
5. Is there a regular pattern within a time range for property P and module M?

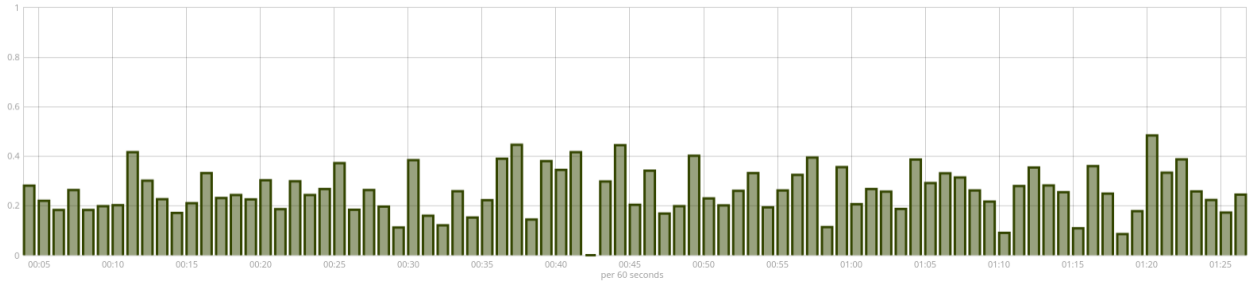

Figure 7.2: A time series visualization for representing real-time, time-series investigation of any property

From Figure 7.3 we can also answer some other questions including the previous ones:

1. For module M1 and M2 is there any similarity for property P (CPU load here), over the time sequence?
2. What could be a possible property P value of another module M2, provided it is correlated to module M1 and M1 has value V for that property?
3. What is the relation between two modules property values V1 and V2 (v1 is bigger or smaller or equal to V2 and so on)?

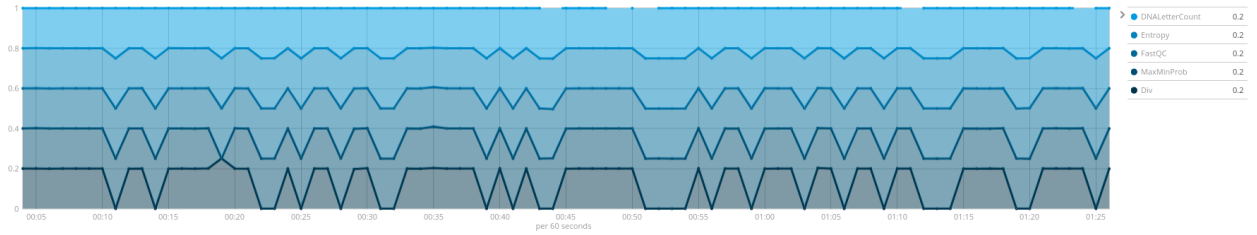

Figure 7.3: A time series visualization for representing real-time, time-series investigation of multiple properties

7.3 Visualization for Monitoring Metrics

In this section, we provide a way to monitor metrics in two fashions. One is to monitor a metric's direct value and another is to monitor within a goal value. This is also real-time.

The questions we can answer from the Figure 7.4 are:

1. What is the metric or property value now for a module M?
2. What is the metric or property value in a fixed-time range for a module M?
3. Which module is showing the maximum/minimum value now?
4. Which module is having the maximum/minimum value in a fixed-time range?
5. Which module's metric seems not to be changing over time?
6. Which module is seeming to be anomalous (by showing a static value or zero) over time?

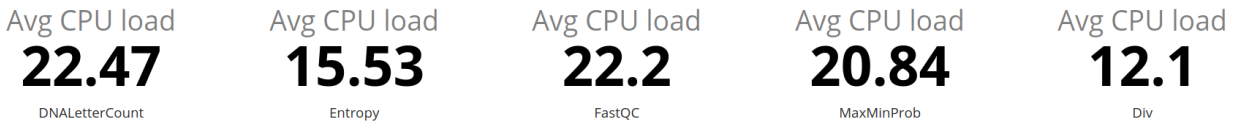

Figure 7.4: An average of CPU load for a fixed-time range for different modules

The gauge chart in Figure 7.5 can answer several other questions including the previous ones:

1. Which module metric (here CPU load) or property is over or equal to a goal value?
2. Which module metric or property is below a goal value?

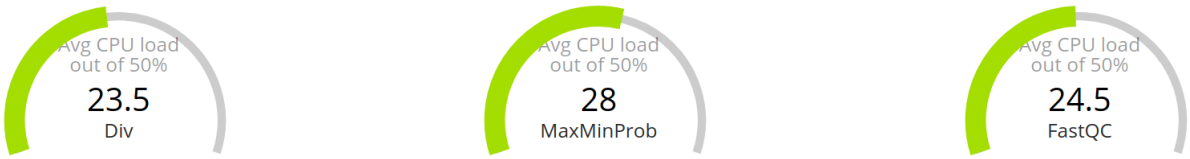

Figure 7.5: An average of CPU load for a fixed-time range for different modules with a goal of 50% CPU load

7.4 Visualizing Proportional Measurements

In this section, we present a way to visualize the proportional measurements for nominal data. This feature is real-time and can be used to get an idea of the ratio of a nominal property frequency/occurrence out of another nominal property.

In Figure 7.6 we can get to know the answer to the question:

1. What are the frequencies of possible values of property P now?
2. What are the frequencies of possible values of property P within a fixed-time range?
3. Are the frequencies changing over time or static?

Figure 7.6: Metric view of two nominal property frequency

The Pie chart in Figure 7.7 also gives another extra questions' answer:

1. What is the frequency ratio of property P (here success or error type) out of all its possible occurrences now?
2. What is the frequency ratio of property P out of all its possible occurrences within a fixed-time range?
3. Is the proportional ratio changing over time or static?

The Figure 7.8 representation can answer more questions by providing a bigger view:

1. For a certain category value K (here tool name), what is the ratio of a property P (here success or error type) frequencies out of K now?
2. For a certain category value K, what is the ratio of a property P frequencies out of K within fixed-time range?
3. Which category K has bigger proportion for any property value V?

7.5 Visualizing Nominal vs Nominal to Ordinal Monitoring

In this section, we present a way that can be used to monitor two nominal properties together while observing another ordinal metric. This is also real-time and can answer several questions:

1. What is the value of the ordinal property P (CPU load for example) for category K1 (FastQC tool name for example) and K2 (MRE error type for example) now?

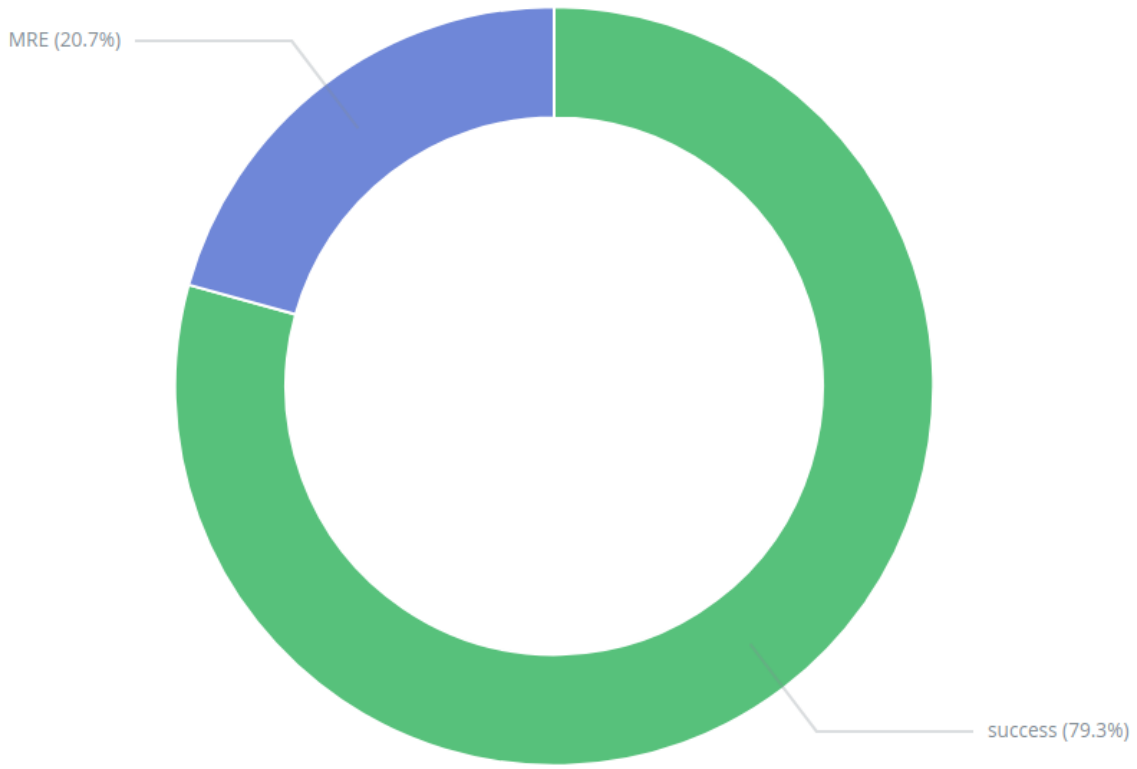

Figure 7.7: Pie chart to provide a proportional view of a nominal property (here success and MRE or module run-time error)

2. What is the value of the ordinal property P for category K1 (FastQC tool name for example) and K2 for a fixed-time range?

3. Which heatmap portions are not changing over time?

4. Which heatmap portion are having zero or the lowest value spectrum?

5. Are two value portions in the heatmap correlated and occurring with the same colour altogether?

7.6 Visualizing Statistical Analysis and Monitoring

Although there are so many statistical measurements, we provide only two of them that are very common and basic. First one is about the percentile distribution of any property and the later one is the standard deviation of that. The statistical questing that can be answered from the Figure 7.11:

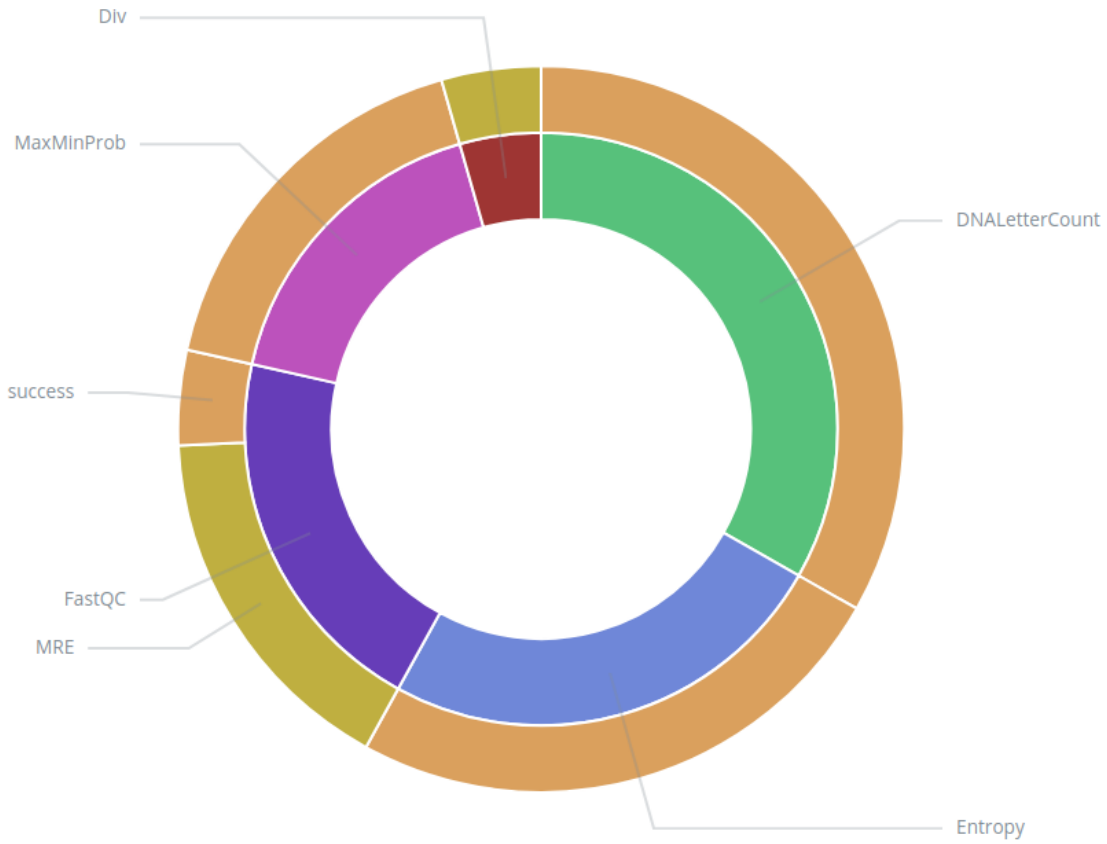

Figure 7.8: Pie chart to provide a proportional view of two nominal property frequencies (here success and MRE or module run-time error along with the module types)

1. What is the distribution of a property P (here CPU load) for a nominal category K (module name) now?

2. What is the distribution of a property P (here CPU load) for a nominal category K within a fixed-time range?

3. Which percentile has the highest/lowest property value for a category K occurrence now?

4. Which percentile has the highest/lowest property value for a category K occurrence within fixed-time range?

5. In which direction the property P is highly/lowly deviated for category K now?

6. In which direction the property P is highly/lowly deviated for category K in a fixed-time range?

From Figure 7.12 we can answer:

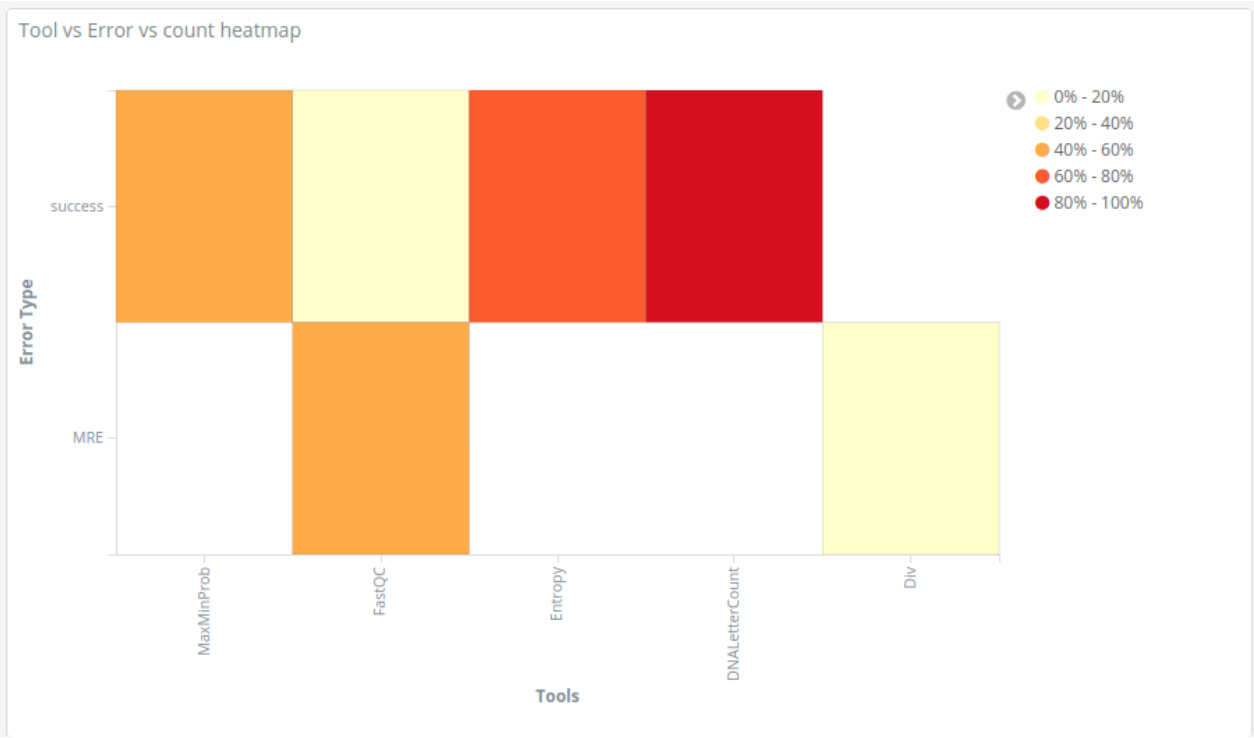

Figure 7.9: Heatmap to represent tool name vs error type to frequency of their occurrence

1. What is the highest and lowest standard deviated values of property P (CPU load) for a category K now?
2. What is the highest and lowest standard deviated values of property P (CPU load) for a category K within a fixed-time range?
3. In which direction a property value V is highly deviated for category K now?
4. In which direction a property value V is highly deviated for category K for a fixed-time range?

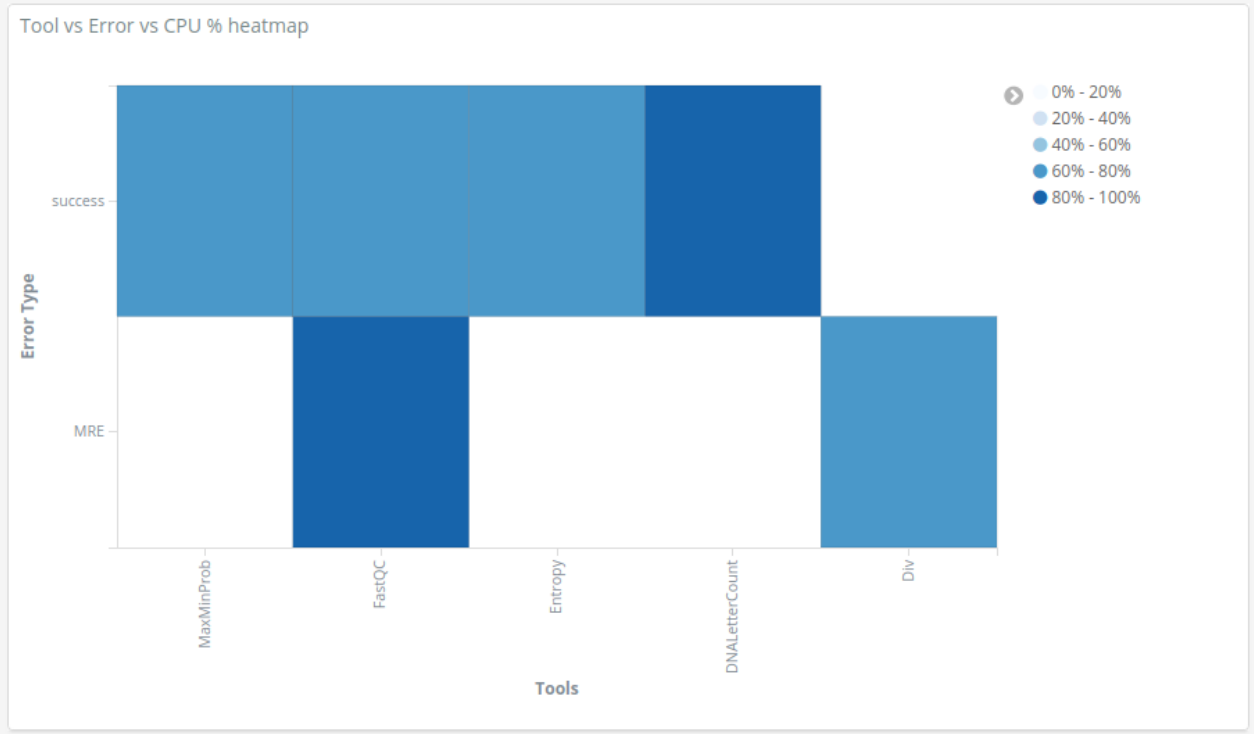

Figure 7.10: Heatmap to represent tool name vs error type to CPU load

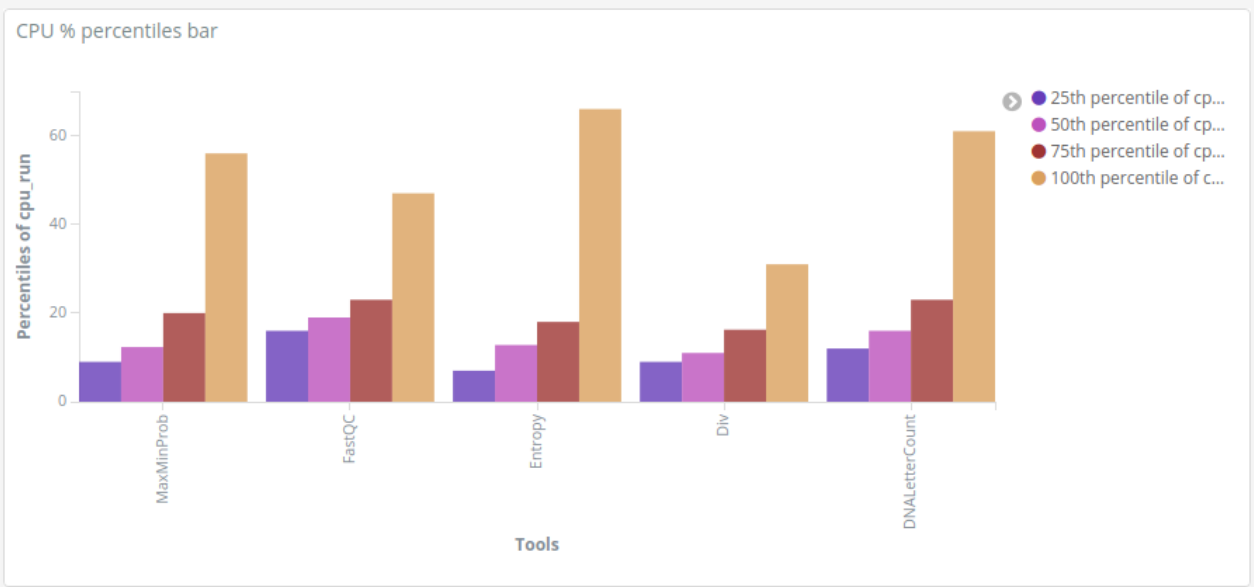

Figure 7.11: Percentile distribution of CPU load grouped by tool names

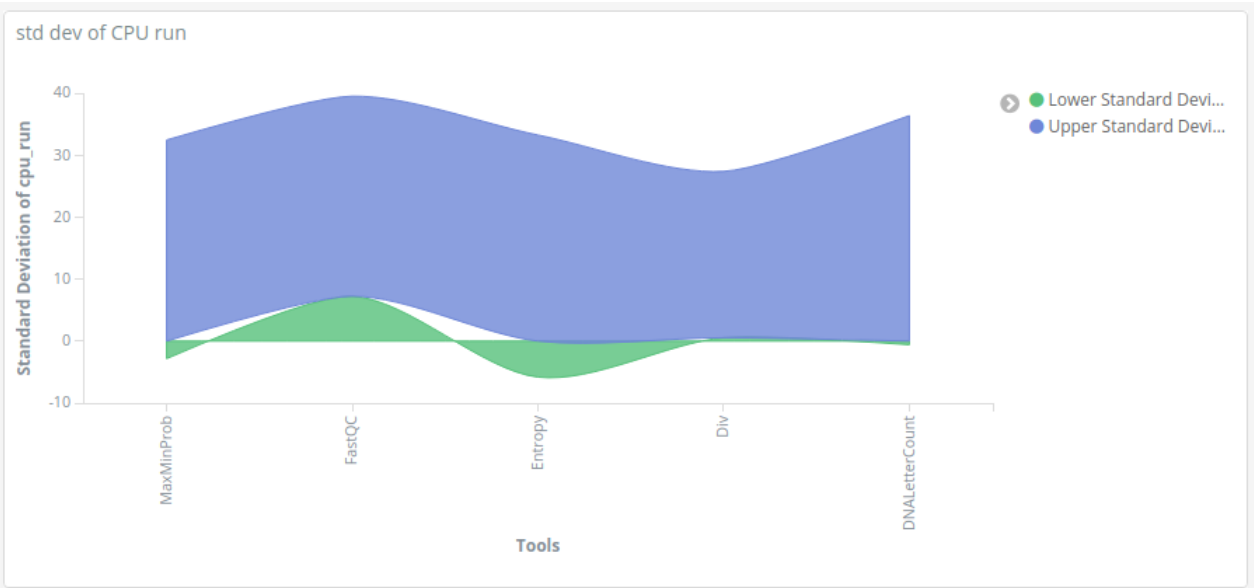

Figure 7.12: Standard deviation of CPU load for different tools

8 Integration and Performance Analysis

In this chapter, we describe how we integrate ProvMod with Apache Taverna and thus offer a new improved version of our model naming ProVerna.

8.1 Integration with Apache Taverna

While integrating with Apache Taverna, our focus is not to get the hands dirty by modifying the source codes of Taverna. it is because, we build our core model ProvMod in such a way that every component that falls under it, is separated as a distinct service. It follows a plug-in architecture that is suitable for easy adaptation with other services from third parties.

8.1.1 Fixed Routine for ProVerna

When we develop any tool including ProvMod for Taverna, to make things simple we follow a fixed routine for each module. It is not mandatory to follow all the time and user may create their own routine but using these routine makes it very easy to integrate ProvMod with Taverna.

The fixed routine for ProVerna:

1. **A tool must take inputs as Objects,**
2. **The input Objects can hold input file paths or parameters,**
3. **The tool must write the result into a file,**
4. **The tool must generate an Object with the saved file path.**

8.1.2 Coding Template

While doing the integration, every tool or function can be easily wrapped using ProvMod and be used as an executable file in Taverna. We provide the developers with a coding template in Figure 8.1 that they can use to create any Taverna tool from scratch, or use an existing tool while enabling ProvMod.

8.2 Benchmark for Comparison

We select a real-world Bioinformatics pipeline for evaluating and comparing our model with Taverna. In Figure 8.2 we present the benchmark. For the analysis, we use a system with Ubuntu Linux 16.04 LTS, 16

GB DDR4 memory, Intel Core i7-7700HQ CPU at 2.80GHz x 8 CPU, 256 GB SSD as secondary memory.

The benchmark takes an input as a .bed file. The file contains exon and SNP IDs for human chromosome 22 [38]. The bed file is converted into a CSV file for analysis. Then from the CSV file a target column ID is selected with a variable that is in our experiment 'exonID'. Then the number of SNPs per exon is count. It is similar to word count for example. Then the top exons are filtered by sorting the previous results. Finally, the top exons and number of SNPs per exon are printed in the console in this pipeline.

8.3 Result Analysis

We run the benchmark for Taverna as well as for ProvMod integrated Taverna or ProVerna. We also run ProVerna in different fashions. For example only keeping the file-based logging excluding the graph database or ELK stack, only file and graph database excluding ELK and the complete ProvMod. The execution iteration ranges from 500 to 5000 in our experiment until we find a significant outcome in our result. The results are presented and discussed in the following.

8.3.1 System Performance Analysis

The comparison is done with Taverna as well as different versions of ProVerna as already mentioned in Figure 8.3. It is clear that, over the iterations, Taverna performance is static for the benchmark. Similarly, for only file-based logging model PM-GDB-ELK, the performance is also static. When we add the graph database in PM-ELK then over the time the performance overhead is increased. It is because the provenance is highly coupled with ProvMod and performs some queries during workflow runs to build graphs. As the number of nodes increases in the graph, small execution time overhead is added to the model. The addition of ELK stack does not add any extra overhead because we can see, the performance line for PM and PM-ELK is parallel to each other.

The execution time is affected by the number of nodes. Thus we also present how it is affected in the model over the iterations in the benchmark. We run around 1000 iterations with the benchmark again and record the execution time to generate Figure 8.4. We can see, the overhead for 1000 iterations is not even more than 3000 milliseconds.

Does the model add any extra memory load to the system for provenance? As we are recording the provenance data, it is always increasing in the graph database. For each iteration of a workflow run, how much provenance data is recorded fully depends on the pipeline, how many nodes are there, the value of Object nodes etc. However, that is totally the responsibility of the graph database to manage and it does not affect the workflow system runs. In Figure 8.5 we compare the Taverna and ProVerna models with our benchmark for 300 iterations and we can see, there is an extra load of memory or around 200MB for ProVerna but that is not increasing over the iterations.

8.3.2 Query Performance Analysis

We also analyze the two kinds of queries that are used during graph building. One is to create a new node and another is to build relations between a new node and a previously existing one. We can see, the node creation is static over iterations because it is only putting a new node in the graph database. However, the relation building query is increasing over time for iterations. It makes clear that the overhead of ProVerna and ProvMod is actually for the relationship building query during workflow run for populating provenance logs.

For offline analysis, a full graph search will retrieve the full workflow provenance graph containing all the nodes and edges. Obviously, as the number of nodes increases, it will increase too. This is explained with Figure 8.7.

For offline analysis, we could use our proposed elementary queries to prepare datasets for data mining and insight findings. How time-consuming are the queries? We present that answer with Figure 8.8. We can see, for 3000 iterations, the Decide query can search a particular node with a very small amount of time (nearly 0 seconds). On the other hand, the two other queries take longer and they increase over the number of nodes in the provenance graph.

8.3.3 Notes on Performance Overhead

The performance overhead is due to the graph building features of our model during workflow runs. Although it is not that high, we can still provide future researchers with some options that they can use to reduce the overhead.

- The graph database can be integrated with an HPC environment,
- The graph database can be managed in a distributed manner,
- Particular logs from particular locations can be stored in particular graph database cluster,
- Processing power can be increased.

RemoveSpace.py

```
import sys
from ProvModel import Module, Object
# create the logic function
def removespace(in1):
    # taking input filename as parameter
    f = open(in1)
    # function logic
    out = open('noSpace.txt', 'w')
    for line in f:
        for word in line:
            for letter in word:
                if letter == ' ':
                    pass
                else:
                    out.write(letter)
    # return output filename
    return out.name

# wrap the logic function with ProvMod
class REMOVESPACE(Module):
    def body(self):
        #unpack input
        in1 = self.P[0].ref
        #apply logic
        res = removespace(in1)
        # pack value into Object
        flow = Object(res)
        #return Object
        return flow

# this script's name as cmd line argument
src = sys.argv[0]
# this scripts first input as command line argument
in1 = sys.argv[1]
# create a tool invocation
# input dataflow
d1 = Object(in1)
# tool initiation *** *** ***
m = REMOVESPACE(d1)
# output dataflow
d2 = m.run()
# output to STDOUT
print d2.ref
```

Figure 8.1: A coding template for the developer that implements RemoveSpace function, to remove any space in the file content

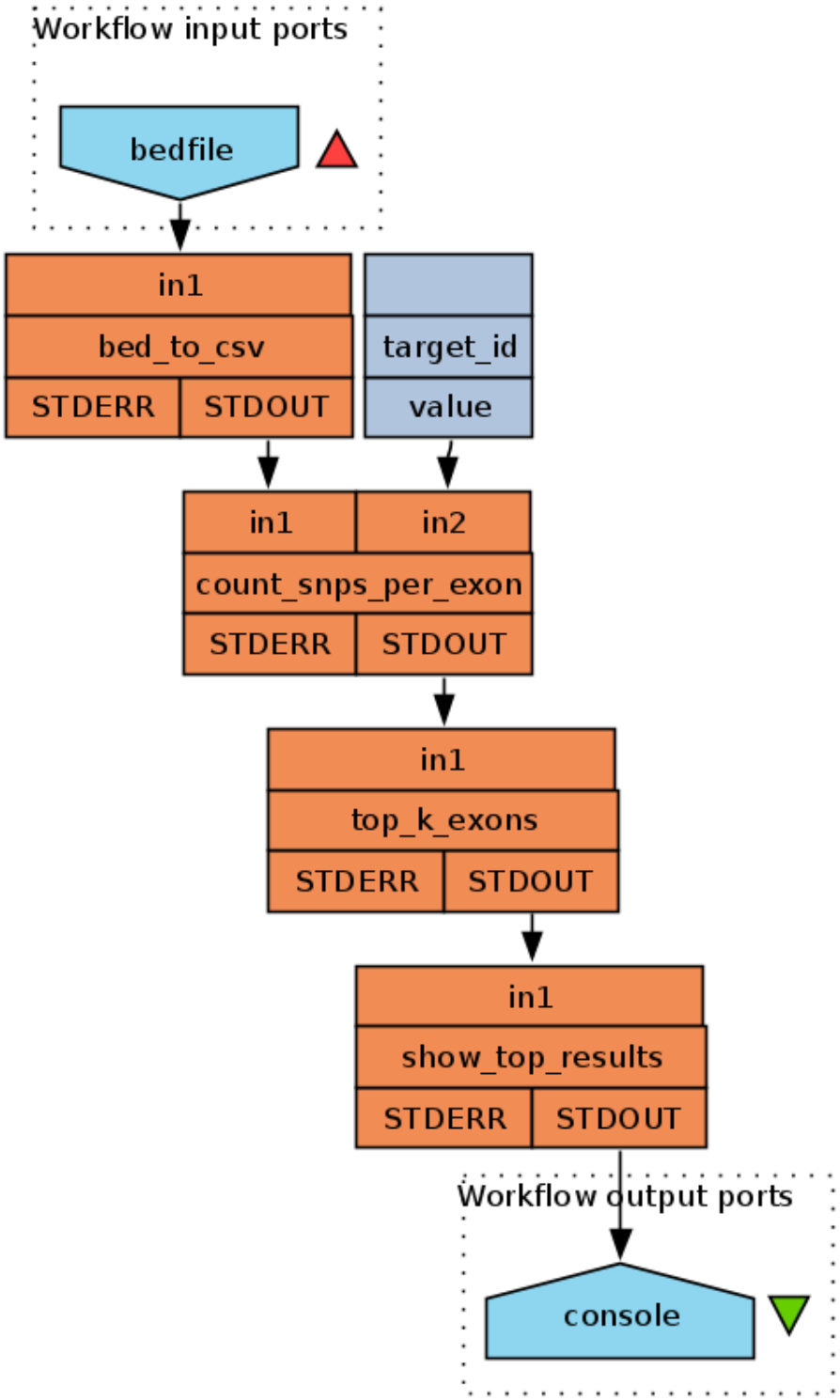

Figure 8.2: A real-world pipeline as the benchmark for ProVerna

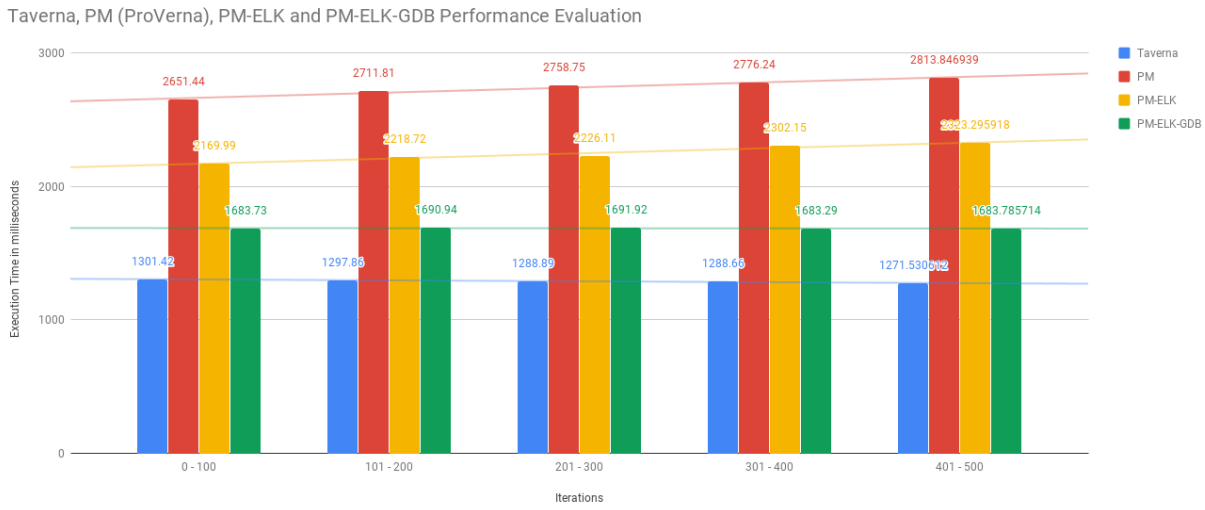

Figure 8.3: Comparing ProVerna and Taverna

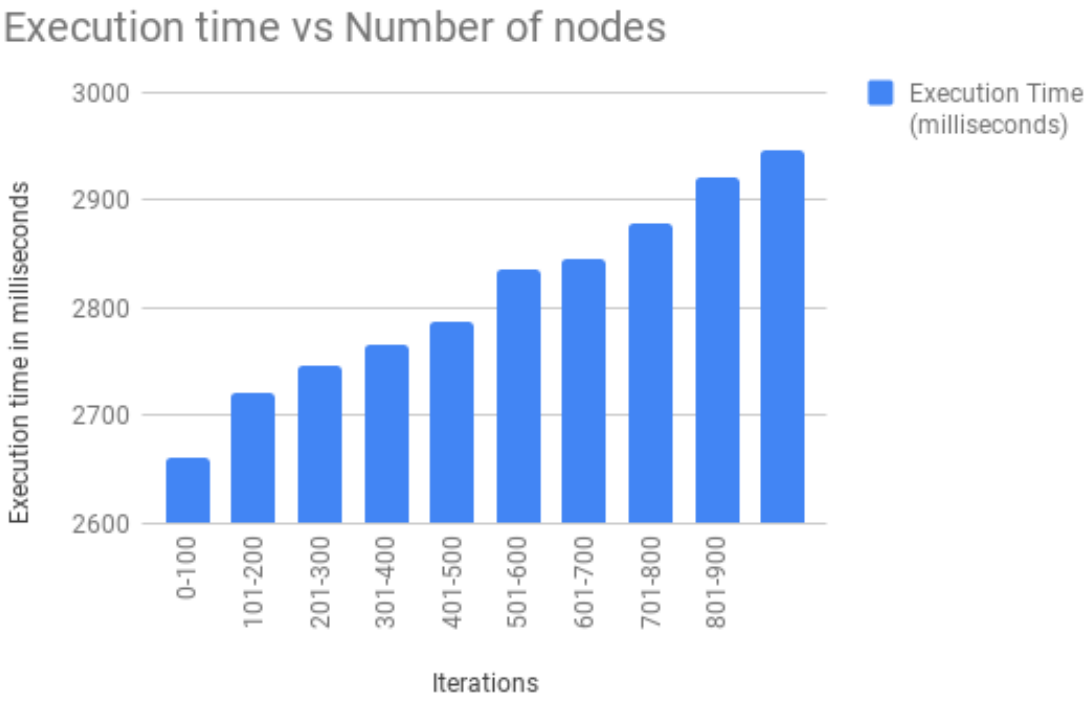

Figure 8.4: Execution time overhead based on the number of iterations

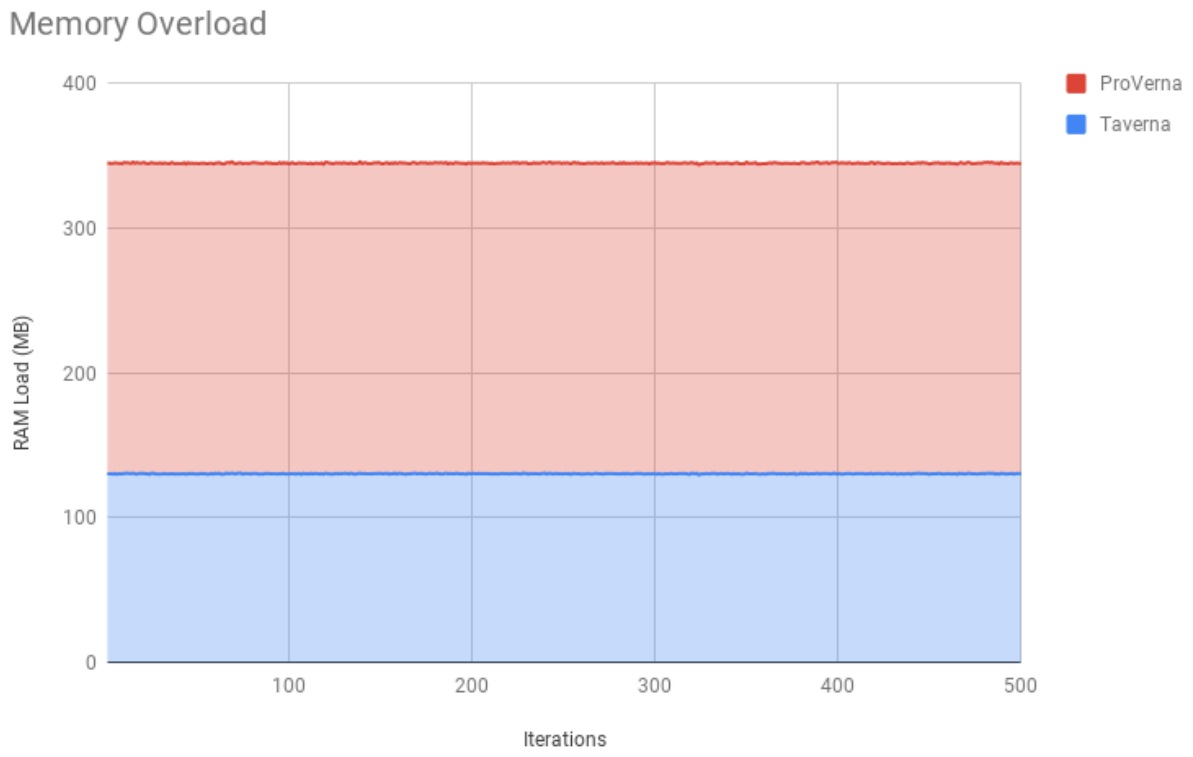

Figure 8.5: Memory overhead based on the number of iterations

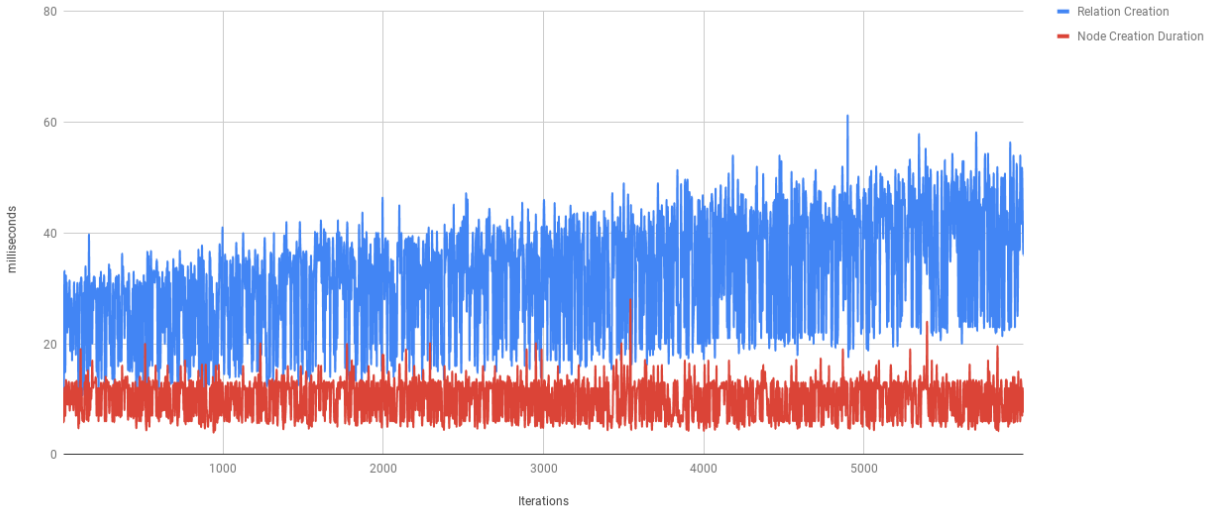

Figure 8.6: Time taken to build graph during workflow run for two kinds of queries

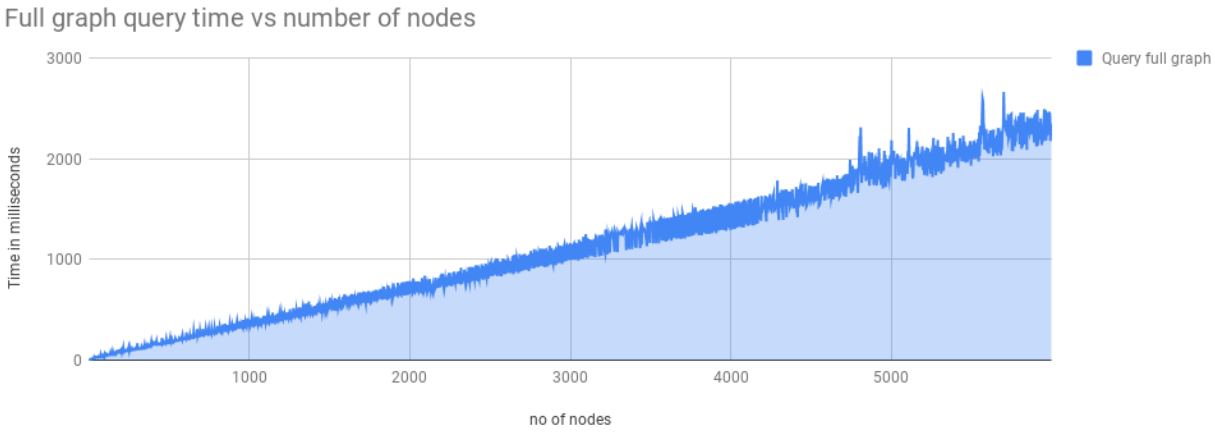

Figure 8.7: Full graph query time based on the number of nodes in the provenance graph

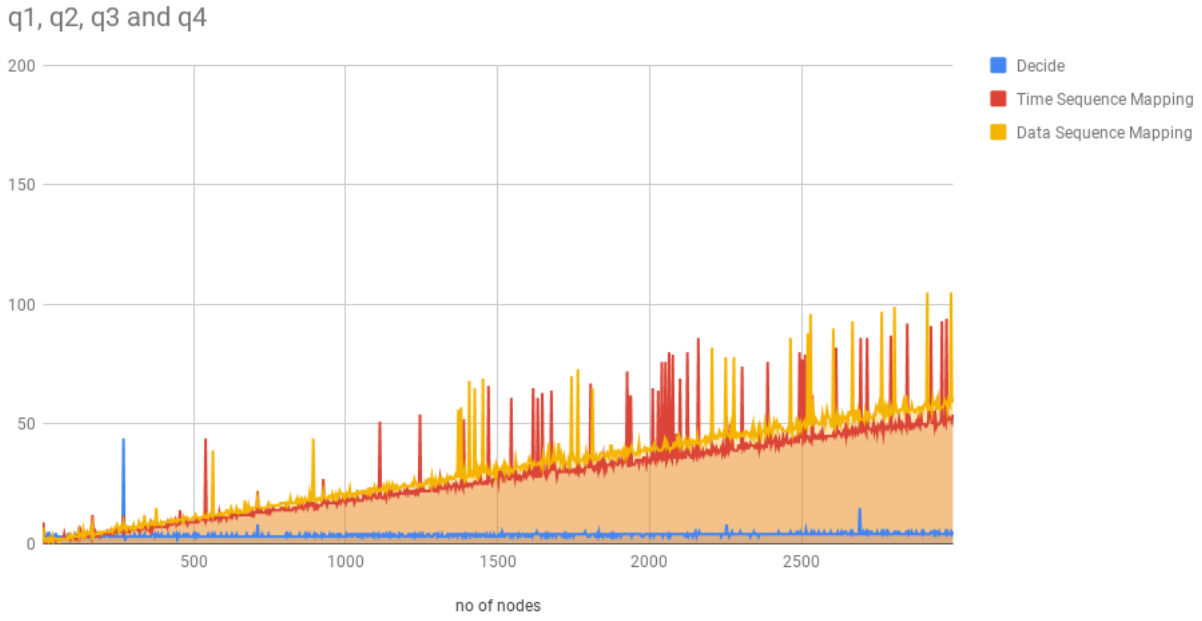

Figure 8.8: Performance of elementary queries Decide, Time Sequence Mapping and Data Sequence Mapping

9 Comparisons

This section is dedicated to present comparisons between ProvMod and different tools with respect to their features. We also present the completeness of our provenance questions classification based on prominent research works.

9.1 Provenance Mechanism Comparison

In Table 9.1 we compare the provenance mechanism of three candidate systems. The first one is an already existing system Apache Taverna, then comes our ProvMod and finally, Taverna integrated ProvMod or ProVerna. Note, Taverna is a complete workflow system that relies on a workflow model clearly. ProvMod is only a model but directly not a workflow system. ProVerna, on the other hand, comes as a complete workflow system inheriting all the features from Taverna and ProvMod. Our goal is to only compare the provenance mechanism of these three. The reader can refer to the taxonomy of Cruz et al. for more explanation of the benchmark for the comparison which is described in Figure 5.3.

9.2 Completeness of Provenance Questions Taxonomy

In this section, we present a comparison of our proposed provenance question taxonomy. The comparison is illustrated in Figure 9.1. The upper portion of the Figure (boxes with orange outline) is the provenance questions taxonomy proposed by our work. The lower portion contains the existing works (boxes filled with grey colour) that were done by prominent researchers in provenance research domain. The directed arrows are to represent which question can be derived from our proposed taxonomy. The blocks that are filled in white color and having no arrows in our taxonomy, are not already covered by the existing works.

We can see, Bunenman et al. [39] only emphasized on the *why* and *where* provenance in their work. Ram et al. present the importance of these two categories based on their work but it was not a comprehensive approach to cover any kind of questions [40]. They proposed a model with seven questions categories. This work mainly targets to provide a comprehensive question classification. Surprisingly, the questions are only focused on the workflow graph but not any other kind of analyses. So, we can see all the questions are covered only by the Decide query that we proposed. Besides, only the *why* query can be related to Data-sequence mapping proposed by our work. Cuevas-Vicenttin et al. [41] proposed different categories of questions. Out of four, three of them can be only be derived by the Decide query.

Table 9.1: Provenance mechanism comparison of Taverna, ProvMod and ProVerna

		Taverna	**ProvMod**	**ProVerna**
Capture	**Tracing**	Eager	Eager	Eager
	Levels	Workflow	Activity limited OS	Activity limited OS
	Mechanism	Internal	Internal	Internal
	Technique	Annotation	Annotation	Annotation
Access	**Visual**	Yes	Yes	Yes
	API	Yes	Yes	Yes
	Query	Internal	Yes	Yes
	Browse	Yes	Yes	Yes
Subject	**Orientation**	Process	Data Process	Data Process
	Phase	Composition Execution	All	All
	Granularity	Fine	Coarse	Fine Coarse
Storage	**Scalability**	Central	Distributed by support	Distribute by support
	Coupling	High	High for automation, low for services	High for automation, low for services
	Persistence	RDF	Semistructured	Semistructured RDF
	Archiving	Time-stamping	Time-stamping	Time-stamping
Access and Language		TriQL	Cypher	TriQL Cypher

Again the last one *mapping* is only related to Data-sequence mapping. Karvounarakis et al. [12] proposed different categories of questions in their work without any formal definition but in a generic way. Three of the four categories of questions can be derived from Decide and only one is related to our monitoring question category. Ghoshal et al. [42] proposed three rules that can be related to our Decide and Data-sequence mapping only.

So, we can see our proposed provenance question classification covers a whole lot of questions that are found from existing works and even more. The classification can also be extended and many more questions can be derived for any single category from the taxonomy.

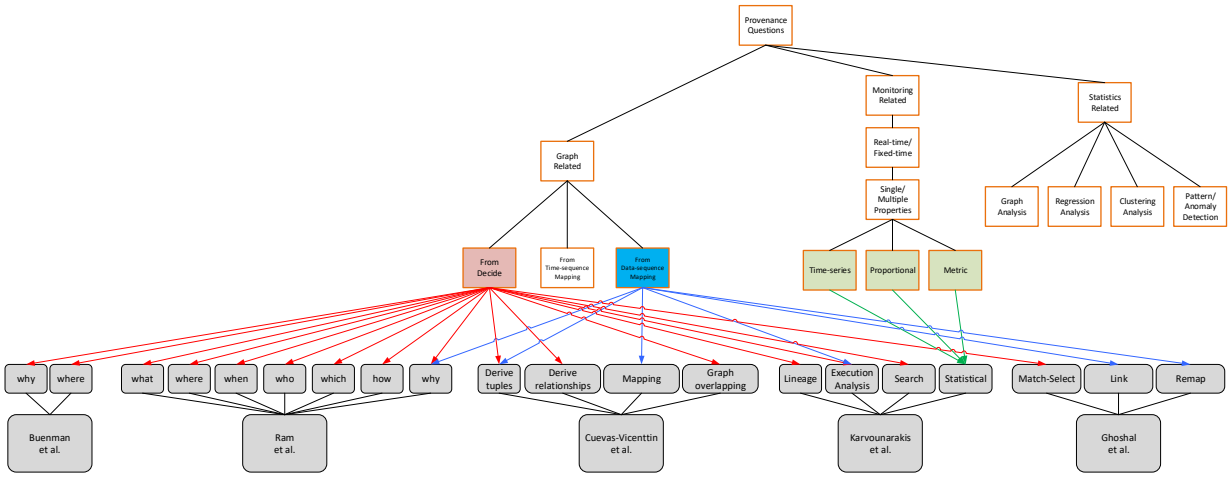

Figure 9.1: Query coverage of proposed taxonomy with respect to existing works

9.3 Comparison of Reporting Service

In Figure 9.2 we present a comparison of our visualization service with existing analytic provenance enabled applications. Those candidate applications are, Taverna [21], ProbeIt [43], Provenance Browser [44], Pro Viewer [45], VIEW [46], SensePath [47] and the work of Chen et al [48].

We can see, ProvMod supports graph visualization and the visualization supports from Taverna, ProbeIt, Provenance Browser and the work of Chen et al. is already covered by our Neo4j graph view. The interactive graph features of Pro Viewer not exactly similar to ProvMod graph view but that is also interactive. Also, manual interactivity can be created at will. To offer the temporal view, we used Kibana for time-series analysis. The spatial view is also supported but not implemented at the moment. Pro Viewer supports such visualizations but that is a domain specific application. SensePath is a prototype but we also support time-series and real-time analysis in ProvMod. Chen et al. only focused on network visualization but that is also covered by ProvMod graph view. Any arrow with a solid line in Figure 9.2 means a certain feature is directly supported in our model and such feature can be regenerated with ProvMod. A dotted arrow means, such features is not directly supported in our model but can be offered using ProvMod. If there is no arrow between a feature of ProvMod and any existing work's feature, it means it is not supported by ProvMod at the moment.

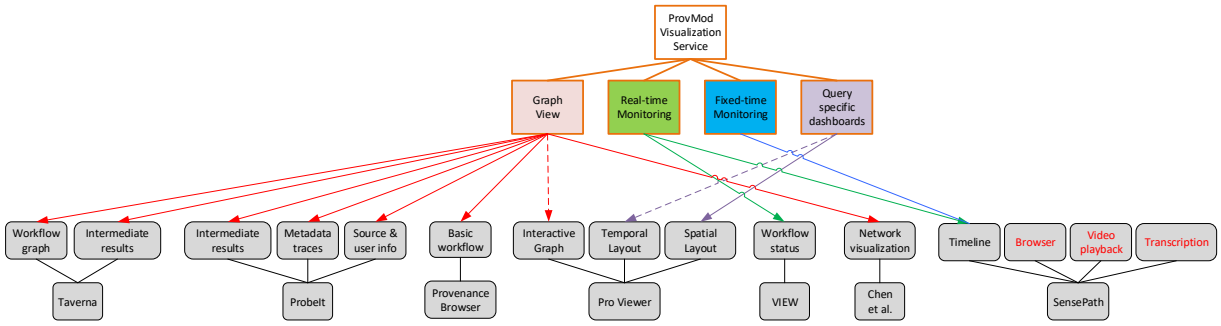

Figure 9.2: Visualization feature comparison with respect to existing analytic provenance research works

10 User Study

We perform different user studies to assess the usability of our model on how easy it is to learn and implement and also to integrate with existing tools. Also, we perform user study for the visualizations so that we can make sure the users find it friendly to answer provenance questions with the Kibana service. At each step we asses the workload using NASA TLX [49] workload index. We have conducted the user study on 11 users till now who ranges from basic programmers to professionals.

10.1 The Method

In this section, we describe each step of the whole user study in details. We conducted two user studies and the first one has four different parts.

10.1.1 Learning ProvMod

In this user study, we make the user read some simple documentation with examples. It also teaches the user about the basic concepts of ProvMod and how to implement it. Then we let them implement some already provided examples so that they can feel confident that they learn the model implementation property. Then we assess the workload.

The documentation is in `https://github.com/rayhan-ferdous/User-Study/wiki`. It contains the description of the user study steps.

10.1.2 Implementing ProvMod

We let them implement a pipeline with ProvMod without any assistance but only the documentation. Then assess the workload for this task.

This step involves using an already ProvMod enabled tool, using a provided function to implement in ProvMod, and implementing a tool in ProvMod from scratch. It is done so that the study can ensure that the user can now implement ProvMod from every angle.

The task can be found in `https://github.com/rayhan-ferdous/User-Study/wiki/4.-Using-ProvMod`.

10.1.3 Inspecting Log Schema

This step makes sure the user is aware that logs are generated automatically. They are asked to inspect the log properties. Also, we take their suggestions on how more log properties can be included.

10.1.4 Learning Taverna

In this step, the user is taught to use the Taverna GUI with a simple example. They learn to run a runnable program via Taverna. Then they are taught to run ProvMod enabled tool in Taverna. After finishing this step, the workload is assessed.

The tutorial can be found in
`https://github.com/rayhan-ferdous/User-Study/wiki/7.-Learn-Integrating-ProvMod-with-Taverna`.

The codes users get in this step, can be used as coding templates for next steps. So, no serious coding intended to problem-solving is necessary during the user study. It is only intended to make the user use the model.

10.1.5 Using ProVerna

This phase provides the user to use ProvMod inside Taverna with a simple coding template. Thus the user learns only the coding schema and immediate implements a new pipeline with ProVerna. In the test pipeline, they use an already existing function to wrap with ProvMod and use an already existing tool for ProVerna. They run the pipeline and we assess the workload.

The task can be found in
`https://github.com/rayhan-ferdous/User-Study/wiki/8.-Using-Proverna`

10.1.6 User Information and Task Load Assessment

During the task load assessment, we take different in formations from the user. They are described below:

1. Fluency level of Python programming,
2. Experience level of working with JSON, and
3. Experience level of working with workflow systems.

Next, we ask them to score on six different questions for each phase stated above (learning model, using model, learning integration and using integration). For this purpose, as already mentioned- we use the NASA TLX workload index for six criteria. They are scored out of 1 to 10. Any score that is close to 1 refers to a very low amount of that criteria. Any score that is close to 10 refers to a higher amount. The six criteria can be measured in various ways based on the goal of the study. Here we describe, how a criterion is measured for each of them:

1. Mental demand: How much mental and perceptual activity was required? Was the task easy or demanding, simple or complex?
2. Physical demand: How much physical activity was required? Was the task easy or demanding, slack or strenuous?
3. Temporal demand: Was the pace slow or rapid?
4. Confusion level: How unsuccessful were you in performing the task? How unsatisfied were you with your performance? How confused were you about the task?
5. Effort: How hard did you have to work (mentally and physically) to accomplish your level of performance?
6. Frustration level: How irritated, stressed, and annoyed versus content, relaxed, and complacent did you feel during the task?

10.2 Evaluating Dashboards

This section is about assessing the visualizations we provide using Neo4j Browser and Kibana to make sure the user is fully getting the real-time monitoring support, graph query support with visualizations, and statistical views of the log data. This is also to assure that, the user can get the provenance questions' answers.

10.2.1 Assessing Graph Visualization

We present the user with the graph visualization and let them answer the graph related provenance questions. We also let them create datasets by using two query templates in Cypher. Then we assess the workload for that.

We present the user a graph of a workflow simulation running in the background. They remain completely unaware of the workflow pipeline structure. There are also such modules that create an error at different time points. We present them the graph visualization of that simulation and ask the following questions:

1. What is the output of module Entropy?
2. What is the input of module FastQC?
3. For a selected module Div, what is the cpu_run property value?
4. run query: match(n:Module) return n.time, n.NAME order by n.time : do you get a dataset?
5. From the query, can you answer which module invocations occurred serially over time?
6. match(n:Module) return n.time, n.NAME order by n.time - can you use this query template to get a time-ordered dataset for n.cpu_run property?

7. What is the cpu_run property value for a module at the beginning of the time point?
8. run query: match(n:Module) return n.cpu_run, n.memory_run : do you get a dataset?
9. Can you use the dataset to map data for analysis (e.g. to find the correlation between cpu_run and memory_run or to draw a scatter plot for cpu_run vs memory_run)?
10. From any selected node DNALetterCount to node Entropy, write the pipeline path (e.g. n1-¿n2-¿n3).
11. Name 2 or 3 modules that lie in similar graph patterns from the full graph.
12. Name 2 or 3 modules that lie in the longest pipeline(s) in the full graph.
13. Name a module that is lonely (no edges from or to it).
14. Name 1 or 2 module that seems to be anomalous to you in any way.
15. For a selected module named Div, what is the error property value?

In the questions mentioned above, every term is relevant to the provenance graph showed to the user. The main goal of asking these questions is to ensure that the users can use the graph visualization to answer the mentioned provenance questions. We are not assessing the correctness of the answers here because they can always answer the questions in a correct manner. Rather, we are assessing the workload to answer them.

10.2.2 Assessing Monitoring Visualizations

This step is about letting the user use the time-series, pie, bar, donut or such charts and metric views that has real-time data streaming capability.

We also let the user use the visualizations related to statistical analysis and answer related provenance questions. Then let them answer the questions related to monitoring and assess the workload.

We present them with several dashboards and ask the following questions:

1. What is the maximum CPU load for FastQC in a selected time range?
2. What is the value of CPU load at a selected time point for Div?
3. Name 2 modules that have a co-occurring pattern for CPU load?
4. Name a module that seems to you anomalous in any way.
5. What is the avg CPU load out of 50% CPU for Div module now?
6. Which module is having the highest CPU load out of 50
7. For the last 50 modules, what is the avg. CPU load for Entropy?

8. Is the last 50 modules' avg. CPU load (from metric view) and current CPU load (from gauge meter view) similar for tool Div?
9. If yes, is the tool Div, executing in a uniform manner with respect to CPU usage?
10. How many MRE errors occurred until now for the time range?
11. How many FastQC tool invocation occurred until now for the time range?
12. For only module FastQC, what is the frequency of MRE error (in number or %)?
13. For only module FastQC, what is the ratio of MRE error (out of only FastQC invocations)?
14. Name a module that has 100% MRE error.
15. Out of all module invocations, what is the invocation ratio of module Entropy?
16. What is the MRE/(MRE+Success) ratio over the time range now?
17. What is the frequency of Entropy module's MRE error occurrence?
18. For module Div, for MRE error, what is the cpu_load?
19. Name a module, for which MRE error frequency is within 0-20%.
20. For FastQC, which quartile has the highest CPU load distribution?
21. For FastQC, what is the 3rd quartile for CPU load?
22. In which quartile, the Entropy module's CPU load is highly deviated?
23. What is the upper standard deviation of Div module's CPU load?
24. In which direction the CPU load is highly deviated for Div module?
25. Are the charts are real-time?
26. Can the charts be used to select a time range?
27. Can multiple properties and parameters be compared using the charts?

10.3 User Information and Skill Level Demonstration

We present the user information to make the reader aware of the users experience and skill level. The figures in 10.1, 10.2, 10.3, 10.4 and 10.5 presents their skill and experience level.

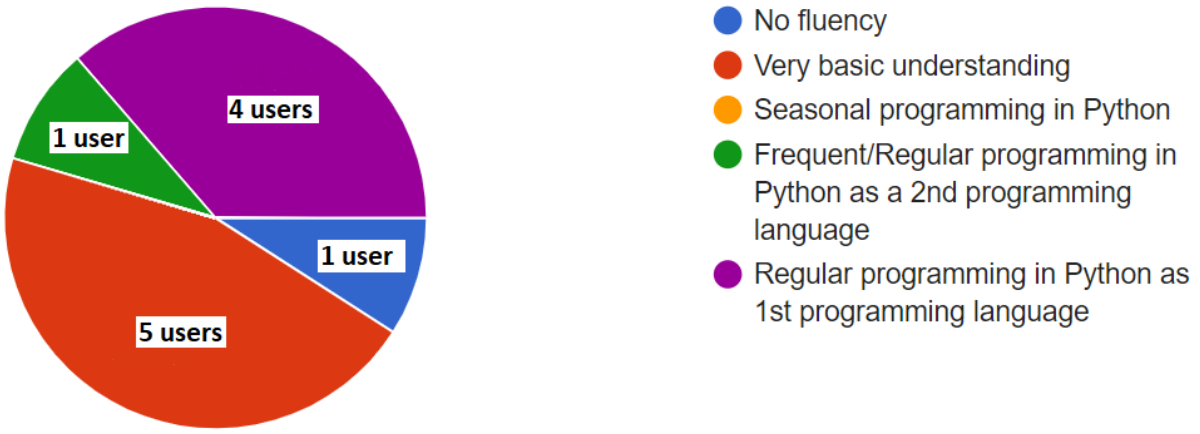

Figure 10.1: Python programming fluency of different users

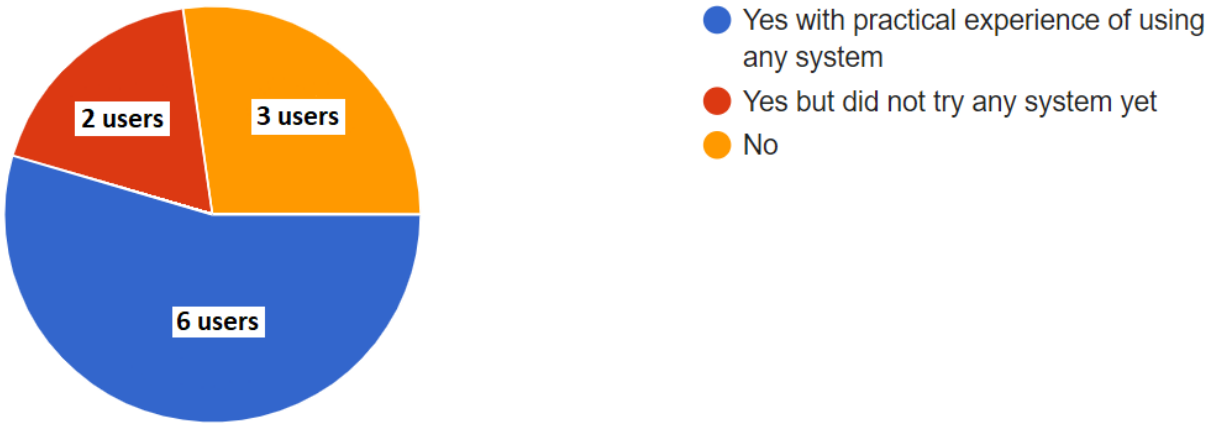

Figure 10.2: Workflow experience

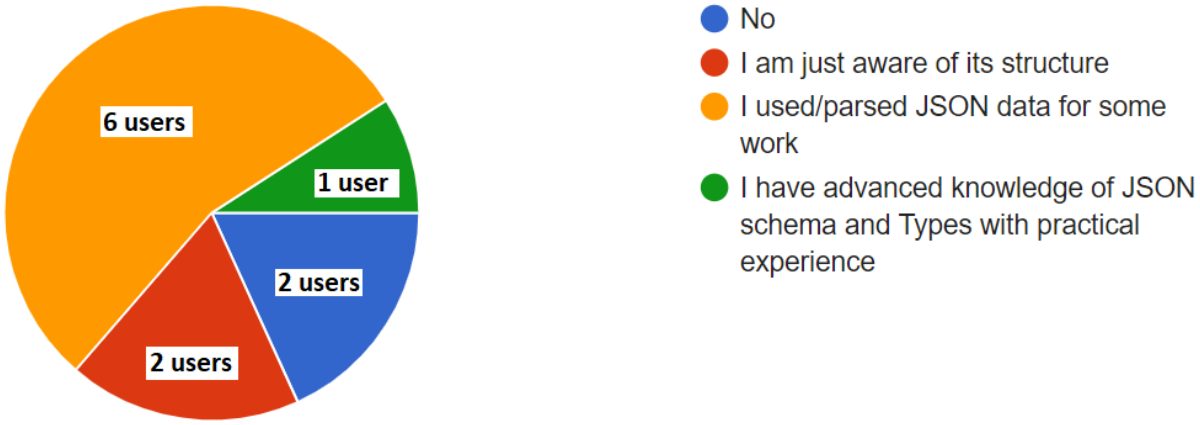

Figure 10.3: Experience level with JSON

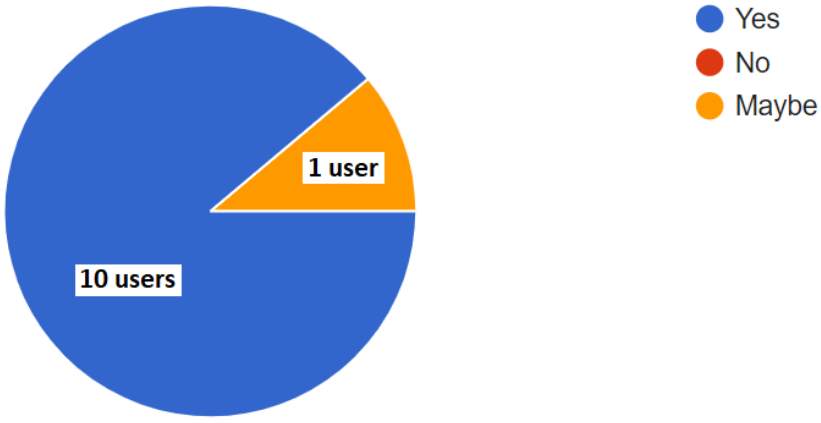

Figure 10.4: Feedback on the question: Is the coding template reusable?

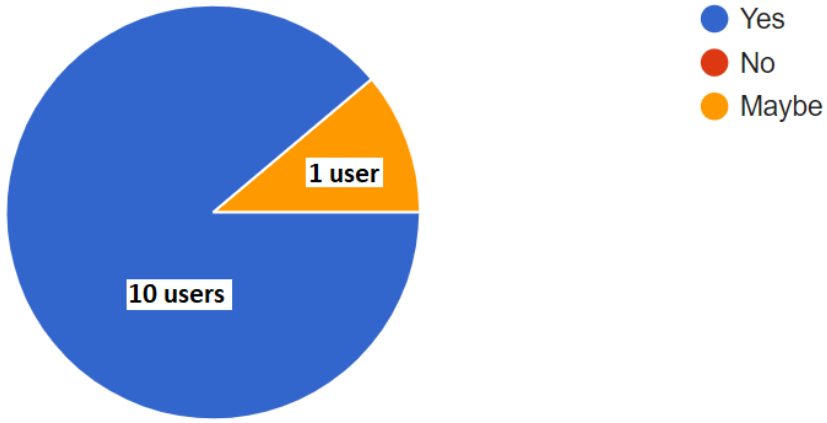

Figure 10.5: Feedback on the question: Is the documentation simple, concise and to the point?

10.4 NASA TLX Score

The average NASA TLX workload index is presented in Table 10.1. Each category of the six measuring criteria is basically measuring a negative metric for the system. So, a higher value in any metric refers to a bad quality of the system. From the table we can see, the mental demand, physical demand for the model related tasks are always less than 4. A lower value (less than 3) in temporal demand for the model related tasks means the tasks were fast to complete. Lower value in confusion (less than 1.5) means that the users had high confidence and success rate for the given tasks. They also need a low effort to learn and use the model as a whole (effort score is lower than 3.5 for each case). They are also not that much frustrated while learning and using the model.

For visualization, we can see it has only a higher mental demand score than any other tasks, but that is even lower than 4.5. It is because, while using the visualizations, it was their first time. Over the time it can be decreased if they become familiar to the system. However, all the users could properly answer all the visualization related questions. So, it is clear that the visualizations of different provenance questions are relevant and clear. The other scores are low in value as usual. Our goal is not to provide the best visualization techniques, but to answer provenance questions using visualizations while offering modern auditing features.

Table 10.1: Average TLX score out of 10 from the user study. Scale 0 to 10 refers to low to high score.

Metric	Learn ProvMod	Implement ProvMod	Learn ProVerna	Implement ProVerna	Visualization
Mental Demand	3.6	3	3.4	2.6	4.3
Physical Demand	2.2	2	2.2	2.1	2.3
Temporal Demand	2.5	2	1.8	1.8	2.3
Confusion	1.4	1.3	1.4	1.2	1.5
Effort	3.3	2.8	2.7	2.6	2.9
Frustration	3.7	3.7	3.6	3.6	1.6

In Table 10.2 we present the median of the users' scores to present the centrality of the scores. By comparing between the corresponding scores from Table 10.1 and 10.2, we find a low difference. So, the average score can be taken as a good measure of task load in most of the cases that we can imply.

Again, to represent the sparsity of users' scores, we present Table 10.3 that shows the Standard Deviation. Standard Deviation is not a very good statistic measure to represent the sparsity, but as our number of users is not very large, we think sticking with this metric at the moment is a good idea and makes things simpler.

Table 10.2: Median for TLX scores from the user study

Metric	Learn ProvMod	Implement ProvMod	Learn ProVerna	Implement ProVerna	Visualization
Mental Demand	3.5	3	3	2.5	4
Physical Demand	2	2	2	2	2
Temporal Demand	2	2	2	1.5	2
Confusion	1	1.5	1	1	1
Effort	3	3	2	2	3
Frustration	2	2.5	2	2	1

Table 10.3: Standard Deviation for the TLX scores from the user study

Metric	Learn ProvMod	Implement ProvMod	Learn ProVerna	Implement ProVerna	Visualization
Mental Demand	1.26	1.05	1.58	1.26	2.07
Physical Demand	1.03	.67	1.48	1.52	1.65
Temporal Demand	1.08	.67	1.03	1.55	1.71
Confusion	1.17	1.06	1.17	1.23	1.65
Effort	0.82	1.14	1.42	1.65	1.63
Frustration	3.4	3.43	3.72	3.72	2.54

10.5 Limitations of The Study and Future Goals

We admit that performing the user study only on 11 users is not a good choice. Indeed, we had to exclude two users' data (13 users took part in total). It is because one user left the study in the middle of the participation. Another user was providing the scores randomly in a very fast manner that we noticed which gave us false data. Another concern was, the user study setup was made completely ready for the users and they had to come to the setup to take part. It gave us the chance to ensure that the users were participating properly and the system is also functioning as expected. Due to other time limitations, we could not increase the number of users at this time.

In the future, we have a plan to provide the complete system as a web-based software system and conduct the study online with a lot of users. That will not only make the study to be parallel for multiple users, but also few false data will not affect the study score that much.

11 End User Features

There are several user-based features that we would like to describe in this paper in brief in the following sections.

11.1 Tool Development

To develop an existing tool such as FastQC we can code like Code snippet in Figure 11.1.

To develop the tool MaxMinProb (that takes the length and count of four bases as five distinct inputs and outputs the bases name with maximum and minimum probability), we refer the reader to see code snippet in Figure 11.2.

Further details and tutorial is in `https://github.com/rayhan-ferdous/Provmod`.

```
class FastQC(Module):
    def body(self):
        import subprocess


        cmd = 'FastQC/./fastqc ' + str(self.P[0].ref.name)
        returned_value = subprocess.call(cmd, shell=True)

        r1 = File(open(str(self.P[0].ref.name)[:-3] + '_fastqc.html'))
        r2 = File(open(str(self.P[0].ref.name)[:-3] + '_fastqc.zip'))

        #print r1, r2

        return r1, r2
```

Figure 11.1: Implementing existing tool FastQC in ProvMod

```
class MaxMinProb(Module):
    def body(self):

        b1 = self.P[0].ref
        b2 = self.P[1].ref
        b3 = self.P[2].ref
        b4 = self.P[3].ref

        data = (b1, b2, b3, b4)

        #print data
        maxval = 0
        maxname = ''

        minval = 1
        minname = ''

        for i in data:

            if i[1] >= maxval:
                maxval = i[1]
                maxname = i[0]

            if i[1] <= minval:
                minval = i[1]
                minname = i[0]

        return Object((maxname, maxval)), Object((minname, minval))
```

Figure 11.2: Developing a tool from scratch

11.2 Workflow Implementation

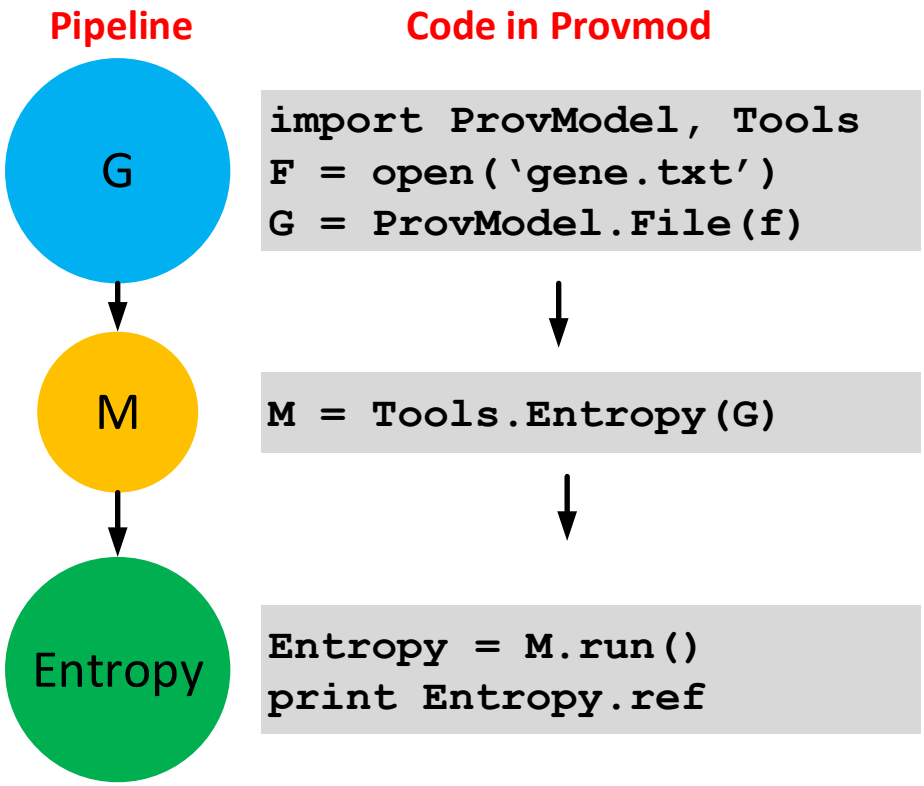

Figure 11.3: A pipeline to implement the tool Entropy from ProvMod library with input file gene.txt, to produce a data item with the entropy value in it

To implement a pipeline very easy and few steps could be followed from Figure 11.3. The steps are:

1. Import the model in Python,
2. Create model-based data instance,
3. Create a tool instance with input instance,
4. Run the tool.

11.3 Logging Configuration

The logging schema is presented in Figure 5.6. The logging schema is configurable (expandable or reducible) from the configuration files in ProvMod and is JSON structured to adapt to any kind of NoSQL database technology. Expert users can use Elasticsearch to do such things.

11.4 Tuning Log Levels

The user can tune the log levels according to their needs and can even disable it. They can also enable or disable any services involved in ProvMod but it is not recommended to disable any service other than log level tuning.

11.5 End User Reporting

The reporting is done through a Kibana front-end in our implementation. The dashboard is a domain/goal specific and user-oriented topic in Kibana. Different users can even make their own dashboards within the same working team. So, rather than presenting the dashboard, we present the visualizations of several provenance questions that is discussed already in Reporting Chapter.

11.6 Integration

The user can easily integrate the workflow programming model ProvMod with any existing workflow system. All that existing system need is to allow any executable programs to run within it. The logging features and automation in logging and log management is integrated by default. The user also gets real-time monitoring all ready to run and use. If they want to develop any tool from existing sources, they can easily follow the coding template that we provide. The steps are:

1. Import or write the logic,
2. Wrap the logic with ProvMod,
3. Make it executable.

12 Two Software Engineering Use Cases of ProvMod

In this section, we present two different use cases of ProvMod. The first one is about clone detection and visualization. The second one is about architecture extraction from legacy systems at functional dependency level. Both the works extend our model to the data provenance level and make use of our proposed elementary questions.

The questions we are answering in this approach involves data provenance, not workflow provenance. Since our model is built focusing on workflow provenance, custom instrumentation and development may be necessary to adopt any approach with the model. The questions now involve the data product which is taken to the workflow visualization space for analysis. So, using elementary queries, we can answer different questions.

12.1 Clone Detection Meets Provenance

To perform the clone analysis we create a parser to analyze any clone detection results from a clone detector. We choose NiCad [50, 51] to do that. Then using the parser, we create different code fragments as nodes in the graph. Now, using the Decide and/or Data-sequence mapping query and from the results of NiCad, we build a graph for different clones.

The full process is illustrated in Figure 12.1.

In Figure 12.2 we present a clone visualization for a subject system JHotDraw [52] which is considered as a very good software that was designed by following software design principles. The nodes are tagged with an ID found from the result of NiCad.

We also apply the technique to analyze BioBlend- a Python library for working with Galaxy [53] API. The result is shown in Figure 12.3. Finally, we show the application of this method even on a mobile version of Apache Taverna workflow system which is illustrated in Figure 12.4.

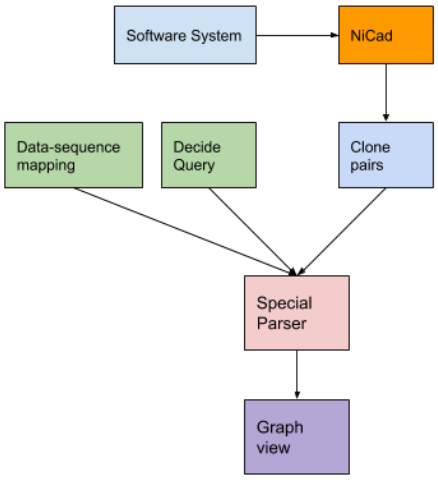

Figure 12.1: Specialized clone analysis approach with the elementary queries

12.1.1 Questions about Clone Analysis

We can answer several questions with this approach that is related to clone research. They are:

1. **What are the clone clusters of any subject system?**
   The answer is the graph itself holding the function level clusters.

2. **For any clone cluster, how many clones are there?**
   The visualization simply presents the number of function level clones with the number of nodes in the graph.

3. **Is the system containing a higher amount of clones?**
   From the visualization, if we find that most of the clusters are having a high density of nodes, then the answer yes.

4. **For any node, which is a unique code fragment, what is the filename?**
   We can use the interactive visualization to get the corresponding filename of any node.

5. **For any node, what is the starting and ending line number?**
   We can again use the interactive visualization to retrieve this answer that is saved as a property for each node.

6. **What is the ID of a code fragment from NiCad?**
   For any code fragment, the visualization is directly representing it with a node labelled with that ID.

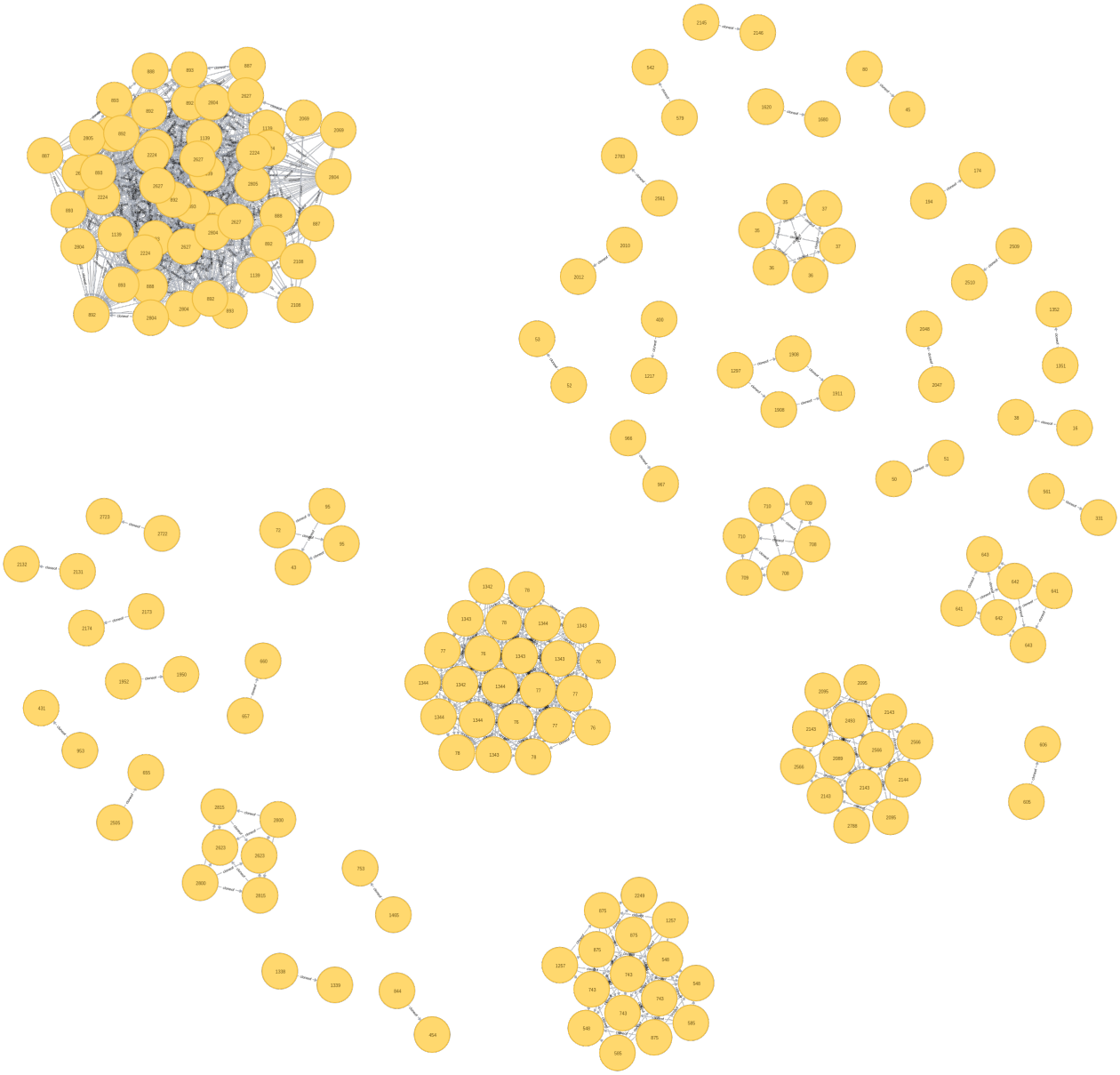

Figure 12.2: JHotDraw clone visualization. JHotDraw is a notable tool for its adherence to design principles. The nodes are function level code fragments labeled with an ID from the NiCad results. All clones are clustered in the graph. Any node connected to another means that they are both clones to each other.

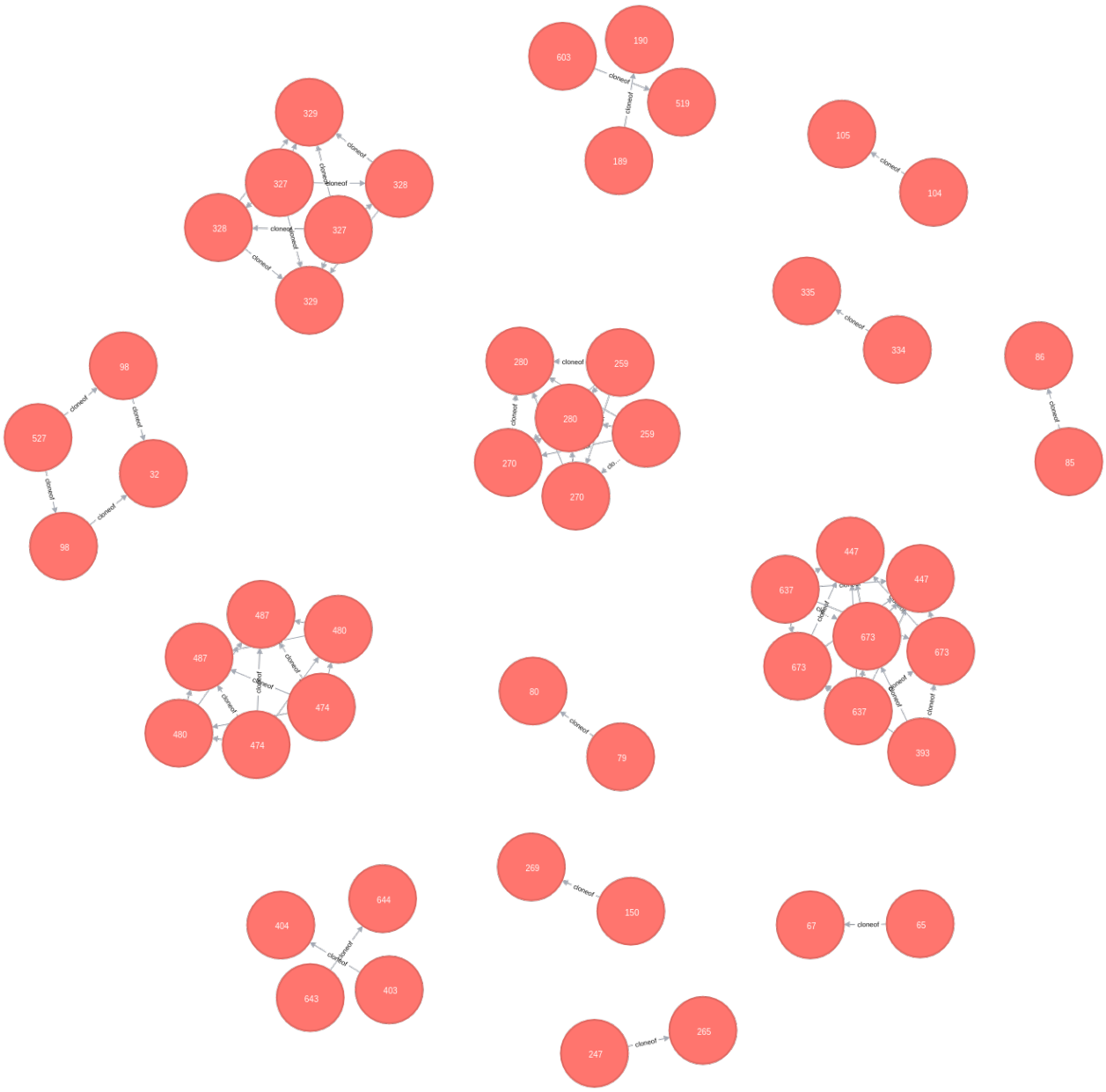

Figure 12.3: Bioblend - A Galaxy library's clone visualization

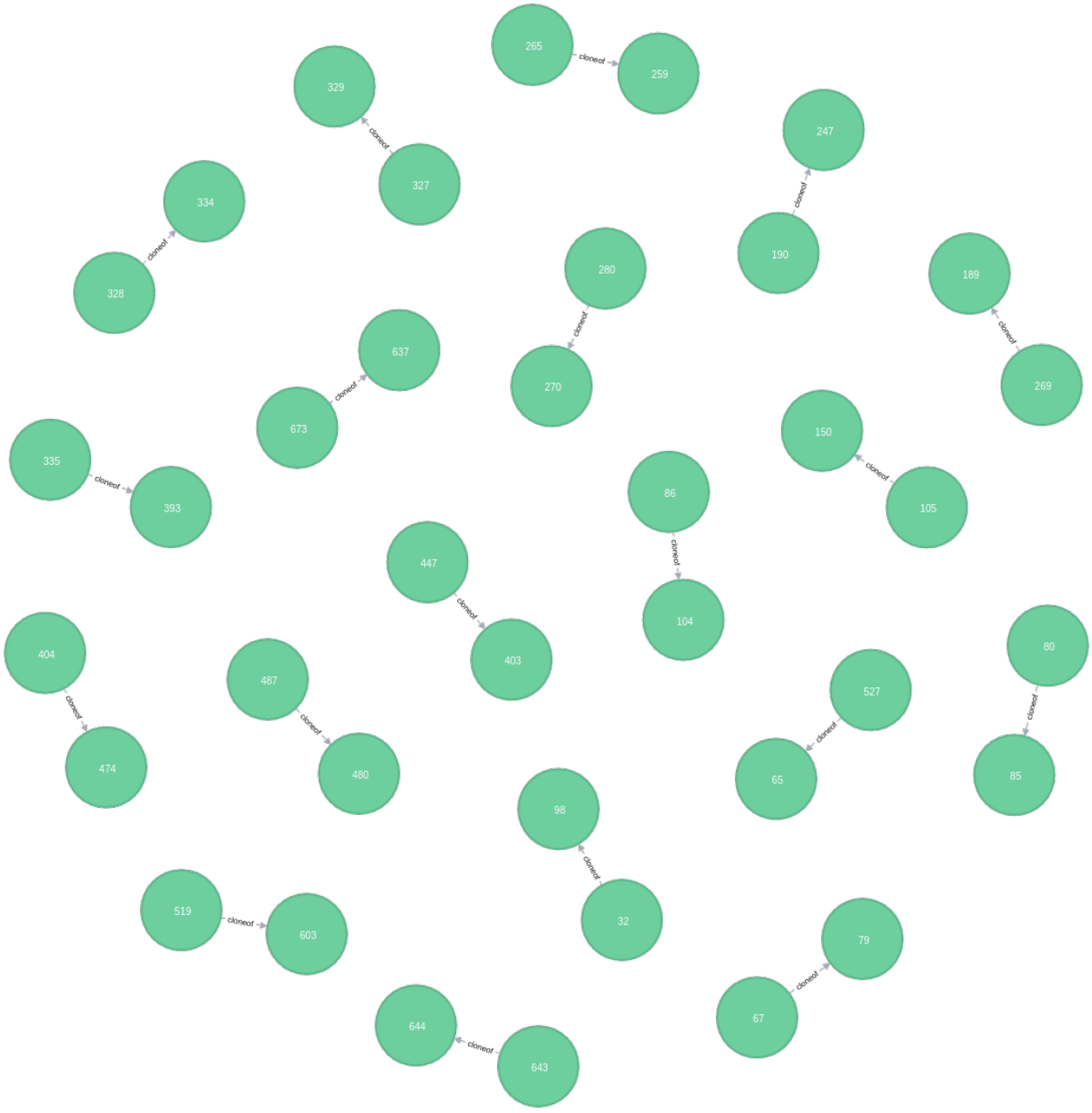

Figure 12.4: Taverna Mobile clone visualization

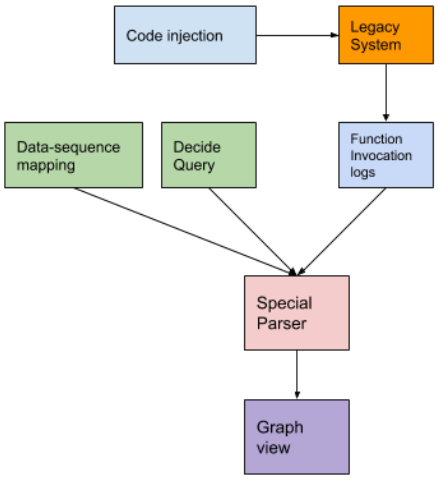

Figure 12.5: Specialized functional dependency extraction approach with the elementary queries

12.2 Architecture Extraction of Legacy Systems

We also apply our model in a specialized approach to analyze a legacy system for basic architecture extraction. We use a legacy subject CRHM [54] which was developed using Borland C++ and it cannot be used in a modern environment. So, architecture extraction to promote migration is necessary. Function level dependency extraction is one of the very first problems of architecture extraction using dynamic analysis. The process is illustrated in Figure 12.5. We perform some code injection or instrumentation to create process logs. These logs are captured when a function is executed or exited in the program. Using the logs, we use the Decide and Data-Sequence Mapping queries to create a graph view of the logs using a special parser.

A simple scenario that we considered is to open the system and open any project after CRHM starts. The full function dependency graph can be found in Figure 12.6. Inside the graph view, we can use the Neo4j to filter only the root node that is labelled as 'crhm' in Figure 12.7. With the Neo4j interaction now, we can trace a directed path of interest for the function call dependency. It is marked with the red line in Figure 12.7.

12.2.1 Questions about Architecture Extraction

We can answer several questions from the graph view that we extracted. They are:

1. **What is the entry point of the system?**

   Here, the entry point of the system is the node labelled with 'crhm' in the visualization. Entry point means, the system's call graph starts from that node and it is acting as root.

2. **Which functions are executed immediately after the program starts?**

   Any node that is directly connected to the root node labelled with 'crhm' are the answers to the

question.

3. **Which function is responsible for a certain GUI activity?**
   For any particular usage of the system, if we immediately extract a graph then that graph is the answer to this question. The directed edges will also answer the order of different operations from the usage. So, we simply use the system for any GUI related task and answer this question.

4. **What is the functional dependency path from function A to B?**
   For any node A to another node B, the directed graph paths are the answers. There could be multiple numbers of paths based on the system design.

5. **Is there any function that is executed recursively?**
   If any node has a directed edge that is actually creating a self-loop, then it is used recursively.

6. **Which function is very common or highly related to other functions having many nodes?**
   If any node has a higher number of the adjacent node connected to it, then it is very common or highly related to other target functions.

7. **Which function did run as a completely distinct service?**
   If a node is completely lonely (has no edge from it or to other nodes), then the function corresponding to the node was executed but totally as a distinct service.

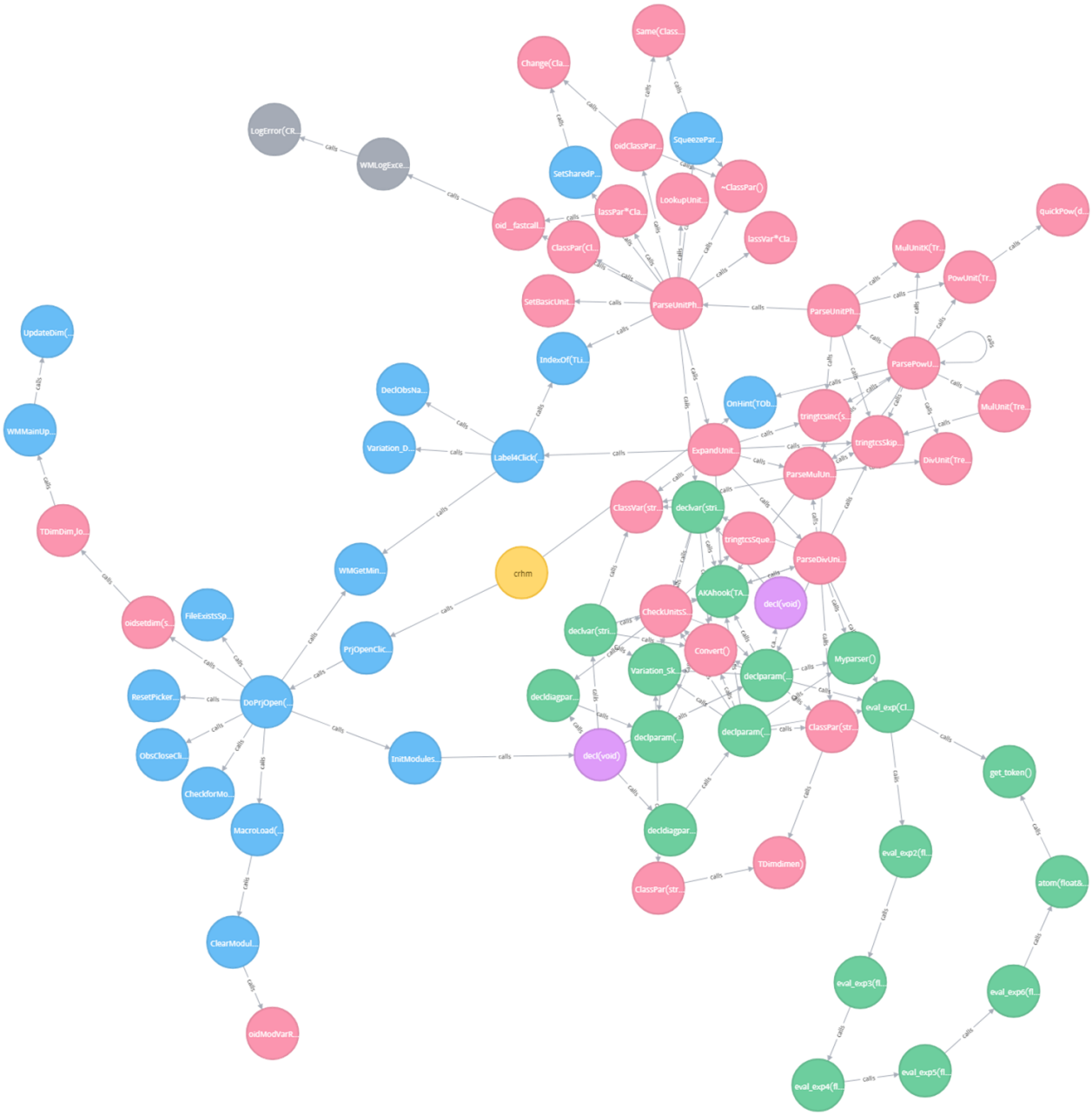

Figure 12.6: A functional dependency graph for a specific usage scenario of CRHM. The system starting point is the yellow root node labeled with 'crhm'. Nodes system functions. Those are categorized and colored on the basis of their source file name. The nodes are connected with directed edge labeled 'calls'. So, if node A calls node B, there is a directed edge from A to B.

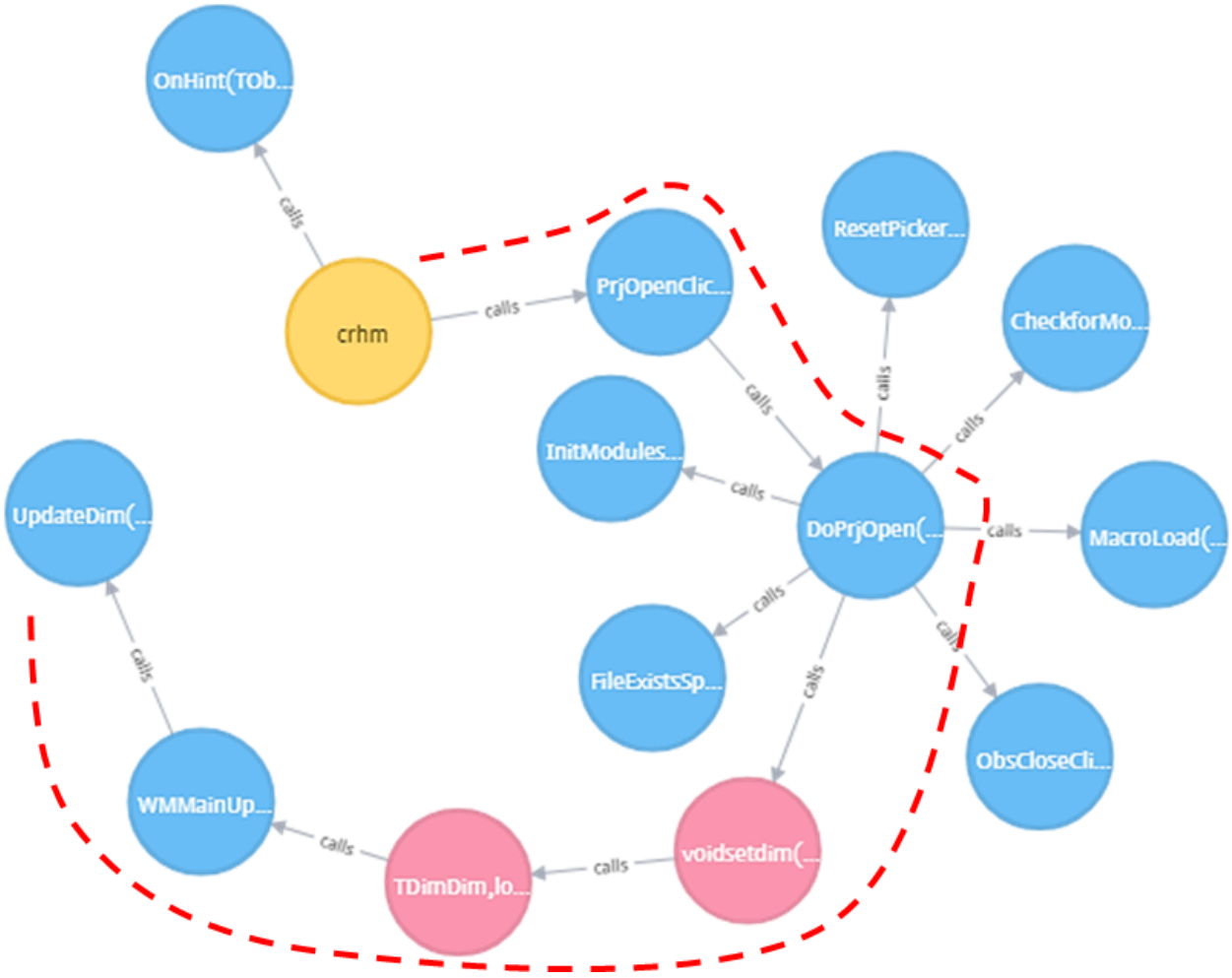

Figure 12.7: A dependency graph for selected functions from the graph for a specific scenario

13 Related Works

In this chapter, we discuss the works related to our research. We categorize them into several categories for the readers' convenience.

13.1 Surveys

There have been different surveys on the state of the art of provenance research. Such survey works provide a broad overview of this topic. These works also help to identify the open problems of this research topic, present challenges, state of the art and so on.

Introducing several workflow systems to the reader could be helpful at first. Such systems are Galaxy [53], Apache Taverna [21], Kepler [55] and so on. Learning and playing with these systems can let the user know about their domains, features and goals. Galaxy and Kepler are solely made for bioinformatics research where Taverna is a general purpose workflow system. All of them are very prominent workflow systems. Taverna also has many plug-ins later developed for improving the workflow system to be applicable to other domains of research. We describe and refer to these systems because to best understand our work, the reader must have some basic experience with the workflow systems.

A taxonomy of WFMSs was presented in the work of Yu et al. [56]. Note, taxonomies differ based on the researchers and various taxonomies were proposed by different researcher groups over the decades till now for WFMSs. They classify WFMSs based on their elements, structure, model, composition system, Quality of Service (QoS) constraints and QoS assignments. In brief, differences of design in WFMSs have already started at the time of this study. Style of information retrieval and feature of fault tolerance were also focused on designing WFMSs. Directed Acyclic Graph (DAG) of Non-DAG based workflows have already been introduced. Different constraints such as, time, cost, fidelity, reliability and security were also marked to be necessary which are mandatory parts of any data-intensive systems. Based on their study, the authors also considered a number of WFMSs and compared among them. A multi-dimensional classification model was proposed by Ramakrishnan et al. [57] with respect to different aspects and angles. Such criteria are size of workflows (based on number of tasks, length of workflow chain, parallel tasks), resource usage (computation time, data sizes, maximum task processor width), structural pattern (sequential, parallel, mesh or their combinations), data patterns (data reduction, production or processing) and usage scenarios (interactive, event-driven or user-oriented). They also provide various examples of different workflow models from different domains and compare them based on their proposal. Another taxonomy is presented by Yu et

al. [58]. that characterizes and classifies different approaches related to WFMSs based on their modelling and implementation. They present a concise taxonomy where workflow design (structured with DAG or Non-DAG), support of parallelism, job scheduling, architecture (centralized or decentralized), fault tolerance, data management and related points are considered. Based on their taxonomy, they also compare different workflow systems in their study. Studying such taxonomies helped us get an idea of different important angles and components of any workflow systems. After considering such important components, we can move forward to design a workflow programming model where no important point will be missing.

Mining workflow systems using transaction logs, different approaches to mining them and issues were presented by Van der Aalst et al. [5]. They present a life cycle of workflows consisting of four phases (design, configuration, enactment and diagnosis). It presents the relations between workflow modelling and mining as well as mining and logging where mining depends on logging but mining is not biased by modelling at the first place. This work mentions the importance of closely monitoring workflow events at run-time (that is real-time monitoring in other words) enables Delta analysis that finds any difference between the design implementation and actual execution of a system. The work also mentions workflow mining is important in Business Process Analysis, Knowledge Management, Business Process Re-engineering, Business Intelligence and so on. However, they do not distinctly focus on workflow modelling from the core, rather only collects event logs without any details of the workflow process. They mention, for larger workflow models, mining becomes more difficult and workflow logs contain noise. Such noise could be generated by any system errors for many reasons. This statement simply projects the importance of error detection using workflow logs. They also proposed a common XML log format. With the help of this work, we get the motivation of a close monitoring system in any workflow system. Such a feature can be guaranteed with provenance and we also feature it in our model. A common log format is necessary but never complete and perfect for any kind of system. To overcome this issue, we offer a customizable and simple log schema in our model.

A survey on data provenance research for e-science was conducted by Simmhan et al. [2]. They propose taxonomies based on different use cases, modellings and structure of provenance systems and features. Depending on that study we come to know about the applications of provenance in data quality measurement, audit trailing, replication and re-usability, attribution and ownership as well as information retrieval. Another topic is the subject of provenance that defines the level of granularity in any provenance enabled system which can be either fine-grained or coarse-grained. The subject could also be data-oriented or process-oriented. The representation of provenance could hold syntactic or semantic information as well as contents. The representation scheme could also be of type annotation or inversion. In annotation type, the provenance data carries more detailed annotated information while in inversion type (a lazy approach in other words), carries compact information to derive necessary information later on. Provenance storage is another crucial topic which could be scaled or stored locally as overhead to the system. Different dimensions are also important in provenance, the visual graph of the queries and APIs for offering services. These dimensions define important implementation goals to design a provenance enabled system. Based on their taxonomy, they

also compare existing provenance systems. From this study, we get the idea about provenance granularity. Our programming model is about coarse-grained provenance. It obtains an eager approach to provenance. Finally, the importance of graph analysis, querying and service-oriented architecture is emphasized in our model and we incorporate each of them in the model.

Bahsi et al. [26] only focuses on conditional workflow management, a feature of WFMSs that enables the user to apply a logical condition, surveys different workflow systems and analyze them. Consequently, the loop feature was also discussed. It states different workflow systems support conditionals (from their first or later versions) in different ways of modelling but the basic conditional structure can be used in other complex cases. Our programming model offers a way to feature conditional workflow building. Such a feature is neglected in many previous works.

A survey focusing on provenance is conducted by Freire et al. [20] for computational tasks in workflow systems. They present the concept of prospective and retrospective provenance. Prospective provenance is basically known as workflow provenance and retrospective provenance is another term for data provenance. They also state the importance of causal relation derivation using provenance data. The work presents different capturing mechanisms. For example, fusing or attaching provenance with workflow system itself, process-based capturing with instrumentation, OS based provenance. It becomes clear from the study that there are several advantages of workflow provenance where the provenance facility is tightly coupled with the system, pretty straightforward and can leverage the support of system APIs. Process-based provenance is target process oriented and instrumentation is a must. They also present a layer based provenance model and compares a number of workflow systems. We take the recommendation from this work and make the provenance management portion of our programming model tightly coupled with the programming model. Note, we only couple the provenance management service with the programming model but not any workflow system. If a workflow system is integrated with our programming model, the provenance facility is automatically supported. To provide a simple tool development process, we offer a simple tool development template and such tools can be run as independent programs. They can be adapted to any platform (cloud service, docker and so on) consequently.

The work of Davidson et al. [6] explains the importance of provenance and particularly focuses on the challenges and opportunities in their work. They describe the overview of this research, supports in existing systems, emerging applications and finally outline the open problems with directions. They state capturing, modelling, storing and querying are all important features of a provenance enabled system. The first usefulness of provenance information is the reproducibility in research that they mention that is also associated with further findings. They mention that provenance simplifies data analysis and exploration by enabling flexible re-use of workflows, scaling it a larger dimensional space. Comparing data products become much easier. Provenance also enables collaboration and that promotes social analysis to discover the wisdom of the crowd. Data mining and visualization research are also tightly coupled with provenance research. The open problems they mention at the end of this work touches every angle of provenance research. The

need for handling large volumes of heterogeneous data in a distributed manner is clearly mentioned. Also, it needs to offer usability through simplicity to the user which is of paramount importance. Analysis and creation of insightful visualizations from provenance data are crucial for debugging and better understanding which is another open problem. This could lead to data pattern mining and discovery. Such processes are extremely hard and time-consuming that involves workflow design and refining. To provide interoperability, it is also necessary to combine provenance information from different systems and models which is an open problem. Such a provenance system that lets the user treat data models from databases and workflow modules uniformly is another open problem in this research. Our programming model can offer reproducibility for already developed tools. Different workflow systems can make use of the model at a single time. The model uses a single graph database to manage the provenance data which is separate from other tool related database systems. Thus we can solve the first issue. We also treat data and modules as simple nodes in our model. Thus we solve the second issue in our programming model.

Provenance already entered cloud computing and faces challenges that are discussed in the work of Abbadi et al. [7] based on different cloud services. Provenance is important for data-intensive scenarios that is clearly mentioned in this work that is necessary for debugging, administration, error detection and security which is facilitated through logging, data forensics, monitoring and so on. Logs and provenance are highly demanding in cloud computing. Although log and provenance are two different things but closely related because provenance depends on logs. Provenance refers to the origin and lineage of any data product where logs refer to event tracing. The work also presents a taxonomy of cloud and their components. Finally, the work mentions several challenges such as- provenance support needs adaptability and custom configuration for new systems; configuring log structure requires domain knowledge which is time-consuming and error-prone and automated logging is necessary to overcome them; user access to the cloud facility is necessary for trust issue; logs need security protection. In our programming model, the log structure can be modified or extended. The mechanism to manipulate any new log properties can be modified in the model too. Both of these is independent of any tool developers. Also, the model does not depend on any workflow system architecture at all. So, either an existing system or a new workflow system, our programming model can be easily integrated with automated provenance support with it.

A very compact but concise introductory discussion is done in the work of Carata et al. [15]. They discuss the source of provenance data and the target users of such data to shed light on the importance of provenance. They mainly focus on data quality and provenance capturing mechanism in a very brief and descriptive manner. They also describe different components of a provenance system, levels of provenance and Big Data-enabled provenance systems. Alongside different properties of provenance system, they also discuss the necessity of integrating provenance into existing workflows which is important. They mention, such a matter is burdensome and different systems focus on different motivations to manage the integration process. To answer provenance questions either exploratory analysis (when users have no idea about data pattern) or directed analysis (using database queries to obtain patterns from aggregation) could be performed.

Sometimes, even a hybrid approach of both of them could be done. There is also overhead associated with a provenance enabled system. The temporal overhead and spatial overhead, as well as security, are important topics in a provenance system that they also mention. Finally, they state research challenges and opportunities. They mention, querying and visualization is an important domain and needs to go even further. Automation is important to go beyond humanly managed queries, data exploration, pattern finding, error detection, validation and diagnosis is another part where tremendous research is necessary. Collaboration with distributed systems is also vital that they mention to manage heterogeneous data and offer long-term storage and Big Data support. Security, as well as privacy, should also not be compromised. To reduce the burden of provenance integration with any workflow system, we design our programming model ProvMod following SoC [59] design principle where every component is considered as distinct service. Provenance is handled automatically and provenance through visualization is offered. In the long run, provenance collection will convert into big data for any big system. Such big data with heterogeneous data structure is handled with JSON data structure. JSON is adaptable to any NoSQL database technology which is suitable for big data applications.

Provenance for big data is discussed by Glavic [10] that the author claims, the topic did not get much attention. However, provenance is a major requirement for big data applications. The work mentions transformation and data provenance. Again, transformation provenance is another term for workflow provenance. Based on granularity, the work also mentions coarse-grained (workflow provenance) and fine-grained (data provenance) provenance. The work projects the importance of provenance on debugging, trust and security and probabilistic data. The term 'Big provenance' is stated in this work. The work also presents several challenges for Big provenance such as- due to heterogeneous data in big data, a common structure is necessary; big data systems tend to use simpler programming models while distributing data and processes and performance information could also be required to include with provenance data; data sources may vary for data products and source related information is necessary to be included with Big provenance. The work also mentions two ways of provenance tracking- to ship important provenance queries answers with the data products to the end user or the actual provenance with the data products. The author also states, data-centric performance measure as well as monitoring and profiling are also important and need provenance facility. Our programming model is based on workflow provenance or coarse-grained provenance. We provide a simple coding template to implement any tool with ProvMod. Thus the automated provenance and provenance analysis services are easily integrated with any tools. If any tool already supports big data and scalability, ProvMod will also support those features with all of its own features.

The work of Liu et al. [60] conducted a survey on data-intensive scientific workflow systems. They also proposed a layered architecture (that is of five layers) and a taxonomy based on parallelization and scheduling algorithms. Based on these contributions, they evaluate and compares different workflow systems. The workflow provenance, data modelling challenges and different approaches are discussed by Bowers et al. [61] where they also emphasize usability, extendability and several OPM supporting features. We also design

our model following a layered architecture so that it can support the mentioned recommended features such as- usability and extendability.

13.2 Automated Logging

Automated logging is of great importance in provenance. Although logging and provenance are two different matters, provenance is never complete without automated logging. Logging involves capturing raw events of any system. On the other hand, provenance involves the analysis of those raw logs and gathering insights about workflow or data.

On log management and exploration for e-science, we can mention the work of Zhao et al. [62]. They mention different levels of logs from abstract views. They are the organization (user, design and hypothesis of an experiment), process (workflow design, configuration, input and output, source and time-related information), data (lineage information derived from process logs) and knowledge level (annotations in form of structured or semi-structured format) logs. They present a Web of provenance that is connected through "Semantic Glue". The work particularly focuses on the Bioinformatics domain and designs the research methodology accordingly. We introduce event tags in our model and all the logs remain on the same log space. At any time, from the provenance graph- the logs can be obtained and filtered based on the need of analysis. This makes the implementation not only simpler, but also new systems do not need to adapt to any further log levels.

An user-oriented data provenance model is proposed by Bowers et al. [27]. After introducing necessary terms and concepts on workflows and provenance, they propose the model from a mathematical background that is capable of capturing simple to complex workflow transactions. They also discuss querying workflow traces by explaining different provenance questions for a particular scenario. In contrast, we consider any user as a single property of any log event. Any log event can be created by any data item or module invocation in our model, independent of the users or systems. So, whether the system is being used by multiple users or not- they are directly logged as a property. Each user is identified by a unique ID in our model.

The work of Allen et al. [22] focuses on heterogeneous data based systems and gaining provenance from new applications in an open world rather than a closed world system. In defining open world systems they consider several properties such as, multiple systems with no prior control, legacy systems without prior provenance support and capturing application (process) level interaction. They rely on Java Messaging Service (JMS), SOAP and so on that are basically for enterprise level web services. They also integrate their model with MULE [63]. Their model is inspired by OPM [64]. To support the heterogeneous data sources we introduce a JSON based log structure in our model. As already mentioned before, JSON can be used for any NoSQL database technology. For web service based information transactions, we only rely on REST, which is lightweight, simple and widely popular now.

In the scenario of cloud computing, how data tracking should be done is studied in the work of Zhang et

al. [8]. They review the current tracking mechanisms with provenance supports and proposes a classification of provenance with respect to granularities. They proposed a model and mentioned the challenges in such a scenario. They clearly mention that provenance is of extreme importance in cloud computing. To verify the authenticity of data products, provenance is necessary due to anonymity on the cloud. Reproducibility and re-usability have to be featured from provenance. Origin and fault detection is necessary for cloud systems as well as storage management, anomaly detection, searching, access control and so on. Confidentiality and auditing such matter are the challenges for cloud computing adoptions. Other than these they also propose a view of provenance granularities that ranges from the application itself to a virtual machine, physical machine, cloud system and finally the Internet. For each of them, they present the required components. For the cloud computing perspective, they introduce several challenges. They describe, Elasticity is a challenge introduced by virtualization that is a feature promised by cloud computing. Other common challenges are diverse structures of data that leads to unstructured data from heterogeneous sources. Their availability and configuration differ. Hardware failure is normal for cloud systems and fault tolerance is necessary. At different levels of granularity, different challenges also occur. A well-designed user interface and better API is required to offer custom annotation and application-specific capturing of provenance. Virtual machines' performance is a very critical issue while considering provenance. Virtual to physical mapping creates a further challenge for provenance on the cloud. On the cloud data communication among different network groups are necessary and its provenance is also necessary. The work also mentions the major requirements and properties for cloud provenance. The requirements are coordination between storage and computer facilities, adaptable and extendable provenance API for custom tools, security elements to ensure integrity of data, auditability, confidentiality, provenance data consistency to pertain data retrieval, atomicity by saving atomic provenance information with data to pertain data storage, causal lineage ordering using provenance, data independent of long-term persistence to discover removed data, and efficient querying. They also propose their own approach and compare it with other few cloud provenance systems. Our programming model ProvMod adapts atomic logging by only capturing workflow events with a simple JSON schema. The schema holds only a key-value pair based information that is kept as simple as possible. Any further information discovered from the logs is done through graph data analysis and real-time monitoring services. Causal lineage is easily obtained by provenance graph.

To create a universal data provenance framework Gessiou et al. [65] proposed an approach using dynamic instrumentation. They implemented their proposed model on top of DTrace [66] and evaluated for different scenarios (file systems, database transactions and HTTP requests). They claim their instrumentation approach to be generic at system and function call level. They capture very basic information through code injection to generate logs. Their work is based on the relational database technique (SQLite [67]). Their scenario is focused on provenance querying thus for web browser based evaluation, they try to correlate between a search term and a final URL. So, a search term is the provenance for the URL that the user is seeking in their work. Our model adopts graph database for managing provenance of the workflow system but keeps

the tool related database separate from the provenance database. Consequently, ProvMod not only remains independent of any particular database technology but also gets all the features of the graph database to manipulate provenance data.

Capturing as well as querying provenance of workflows in real-time is also very important that becomes clear from the work of Costa et al. [11]. Such systems that leverage the power of workflow implementation, are mainly data-intensive systems and may take weeks to finish execution in high-performance computing environments. Consequently, runtime provenance analysis becomes necessary even before the tasks are finished. For fine-tuning parallel processing, it is also necessary. For large scale systems, error detection and management is necessary as it takes time and only waiting for the jobs to finish will be devastating to the experiment. The real-time monitoring also helps not to compromise performance and reliability of any parallel processes. They state their main idea to be, to re-execute any failed job inside already running environments. Such an approach helps a lot in performance management because those failed jobs can easily make use of the extra resources that are already allocated for them beforehand. Also, it allows staying within the same running workspace on the cloud. Such feature is called elasticity. Throughout the time-consuming execution, users are also able to be aware of the status of the system. Another issue is to map among different tasks in a distributed system. They also propose a model for runtime monitoring of provenance with querying features but that model is based on relational database SQL. ProvMod on the other hand, can inherit all the features of any tool and also offers automated provenance. So, if any cloud system or tool offers elasticity, ProvMod enabled tool will also offer that. For run-time analysis and monitoring, ProvMod also comes with a visualization support. It answers every possible provenance questions and the user does not need to write complex queries afterwards. Still, they can create their own dashboards, visualizations with their own queries and aggregations.

How provenance could be captured from logs while considering it as a big data problem is discussed by Ghoshal et al. [42]. They propose a rule-based approach. The main contribution of their work is, they propose to get the most out of existing logs while compromising the completeness of logs to reduce instrumentation and increase the ease of provenance capturing. Their model focuses on capturing provenance from logs while such logs are not intentionally created to capture provenance. The rules are for event capturing as well as provenance derivation. An XML based rule language is proposed that can be used to map provenance events from logs into structured provenance events. There are different levels of log and that could be used to capture different provenance granularities that they claim. They also classify the provenance that is processed provenance and data provenance. The rule engine has three rules that are used to extract provenance from raw log files which work depending on XML specification. They propose Match-Select Rule for direct selection of data value for a provenance event, Link Rule to specify links between two different provenance events and Remap Rule create a rule-based alias. The main problem with this approach is, if any information about the target data is completely absent in the log files, it is never possible to retrieve any provenance information from the logs. Logs also need to be managed to reduce any redundant information beforehand which makes

the impression that, the log structure was already designed to capture provenance (which is actually not). Consolidating logs from various sources in distributed systems is also another challenging problem for this approach. Our model, in contrast, offers an easy way to integrate it with any tools and existing systems. The work of Ghoshal et al. do not target direct provenance oriented logs in their approach. ProvMod, on the other hand, works with such logs that are oriented to provenance. The properties of those logs are also followed by a recommended JSON schema. The properties in the JSON schema is necessary to answer provenance questions that we mention. Also, they proposed three rules to analyze the logs. We propose a provenance question taxonomy which can even be extended further. Their work also relies on a certain level of instrumentation while compromising provenance, but our model does not rely on instrumentation and does not compromise workflow level provenance.

Imran et al. [18] proposed a layer based reference architecture for provenance and visualization from Big Data perspective. The layers are separated into storage, application and access layer. Other than presenting the use cases, they also present their design goals. The design goals are to prioritize less overhead for provenance, user-based custom annotation support, visualization support, scalability, heterogeneous data models support, security and flexibility in data cleaning. They present different user roles. For database support, they also consider graph database alongside relational database. While the relational database is useful for structured data, the graph database is useful for semi-structured to unstructured data items that they clearly mention. They state meaningful data (which's structure or pattern is known or discovered) items could be published to High-Performance Computing for further analysis. NoSQL based data sources are referred for data storage. It is also encouraged to use JSON or BSON documents for data management. For querying CLI based interface could be used for users who are comfortable with command line tools. Visualization could be offered from different angles by their reference. User-based visualization focuses on visualizing user-centric information. They also project importance on time-based information that generally produces a time series analysis. We also adapt the graph database in ProvMod to provide structured or semi-structured data support. Visualization is taken as the primary way of answering a number of provenance questions in our work rather than plain text.

A provenance middleware which is also claimed to be generic is proposed by Arab et al. [68]. They represent an algebraic graph of database operations in their work as well as declarative and rewritable rule specification language, multiple database background support and also query optimizer. An optimizer applies heuristic and cost rules that are used to rewrite the queries into SQL and later optimized by the core database system. They also mention their query language is extensible through rewrite specification language (RSL). Although extendable, their work depends on SQL based database technology. On the other hand, we rely on non-relational graph database technology that is tightly coupled with ProvMod.

13.3 Data Provenance and Querying

There has been a tremendous amount of research over the last decade on data provenance. Although our work is not focused explicitly on data provenance, we still present a brief review of data provenance as it is always related to workflows and provenance research. Data provenance research is important mainly for getting an idea of the state of the art of provenance querying. Different research and techniques were emerged on this matter to query data provenance. Also, data provenance research comes before workflow provenance research. So, to fully understand the work presented in this paper and move forward, we need to study data provenance research. Querying is highly related to provenance, no matter that is about data provenance or workflow provenance. So, to cover every aspect of provenance questions/queries, we study data provenance.

Buneman et al. [28] raise some technical issues of data provenance that are emerged by the ongoing research. The issues they mention are basically created due to heterogeneous data sources, types and structures. They state a need for querying with multiple database sources and citing them with a general fashion. They also mention the general data citation approach received very little attention. Version controlling and management of data using such versioning is also necessary that becomes clear from their discussion. Such management could be used to regenerate lost data in case of any error or failure. Finally, they project the importance of a general standard in data provenance.

The work of Foster et al. [69] presents different case studies for handling replicated data for offering data synchronization for provenance. They describe the use cases of Lenses [70] and Orchestra [71] to discuss how they handle replicated data. They discuss how Lenses manage ordered data and data chunks. Orchestra is a collaborative data sharing system and in their work, they present the methods Orchestra uses to offer security and incremental maintenance of data. Program slicing is an old technique to understand and debug the program where the program is broken into multiple result parts that contribute to generating the final result of the program. The work of Cheney et al. [72] proposes their ideas on program slicing that could be used to understand provenance in databases. To overcome the manual recording or capturing of provenance Vansummeren et al. [73] presented an automated way of provenance recording for SQL database technology. They present from a theoretical background, how provenance querying and updating support could be offered in an automated way for relational database systems.

The issues that occur while building a practical provenance systems are discussed in the work of Chapman et al. [9]. The features they present that should be present in a provenance system are choice of granularity, exact provenance capturing for each particular data item not any generic information, variation in provenance data content based on the application (configurability in other words to remove unnecessary data from logs), capturing non-automated processes that are required and falls under the question 'How much provenance to capture?'. Under system issues, they mention- unique property to identify data items, provenance storage size, execution information in detail, inter-system based provenance information are required. To offer usability- users ability to configure provenance information from different levels, queriability with data, error detection

and fixation are necessary that they clearly state in their work. They target on two issues and proposed their idea on how they could be solved. First one is capturing a non-automated process that could be tweaked by using an appropriate architecture, to force the user to use the system in a systematic manner. The second issue is to reduce the provenance store that can be achieved by using the approach of Chapman et al. [74]. This method uses a family of Factorization algorithms and Inheritance algorithms to break down data produces into smaller pieces and decrease redundancy. Inheritance algorithms could be used to reduce dataset properties and represent them in a compressed fashion.

A categorization of existing approaches in data provenance is done by Glavic et al. [75]. Provenance models are categorized in terms of the model world, provenance identification, transformation and their representation or support, source representation or destination. Each category is again categorized into further classes. Query and manipulation functionalities are further criteria of their categorization where different data manipulation operations are mentioned. For storage and recording, different strategies are also mentioned. Based on their categorizations, they also compare different existing systems.

ProvMod does not work with the data provenance level though, it can offer user-based annotation in the data item while building pipelines. Such custom annotation based information can be later used to derive versions. Also, ProvMod can be adapted to support different systems and consequently heterogeneousity. ProvMod also supports automation. So, if the model is extended to support data provenance in future, data provenance automation will be available in the model. Again, ProvMod supports non-relational graph database to manage provenance. On the other hand tool databases are kept separate. Error detection and fixation is based on workflow provenance in ProvMod. Certain data items involved with certain errors can be traced back using the workflow provenance of ProvMod.

The research problems in data provenance are again discussed by Tan et al. [23]. We come to know about the sequence-of-delta and timestamping approach to capture provenance. Sequence-of-delta approach records the difference between two consecutive versions at once and in timestamping approach, provenance is recorded with event occurrence time. The applications of provenance they mention are- trustworthiness of data, sharing knowledge via annotations and data verification. For future research, they also provide some guidelines. They state, it is necessary to investigate whether the lazy or eager approach is better than one another in capturing provenance. To query provenance, query extension support is necessary. Classical minimum difference capturing was used to save data with well-structured data semantics. Such a compact result at the end is not always complete and enough. So provenance information must be completely preserved or easily regenerated when needed. Historical queries to the archive can also be saved for later repeated use which is another complementary approach in this issue. ProvMod during provenance capturing follows the timestamping approach. It also holds provenance information as minimal as possible in the logs. The provenance information is also captured immediately at the occurrence of an event and thus ProvMod adopts an eager approach.

A virtual data provenance model is proposed by Zhao et al. [76]. The virtual model proposed by

them is integrated with the system alongside semantic annotations with powerful query capability. They present a schema for provenance annotations by considering different data related logical elements. They also present a general model for provenance and various annotation queries but that is for SQL or XML database technologies, which are structured. ProvMod on the other hand, specifically designed to support unstructured provenance data.

Buneman et al. [77] discussed why and where questions for characterizing data provenance. They tried to follow a syntactic approach that can lead to a general data model but applied to relational and hierarchical database systems. The "why" provenance refers to any source data that influence the target data product generation and where provenance refers to the location from which the target data product was generated. They discuss the topic from a mathematical and modelling background. A new semantics of data provenance is presented by Ram et al. [40]. They name it W7 model where the provenance questions are framed using 7 query phrases- "What", "When", "Where", "How", "Who", "Which" and "Why". They claim the model is general and extensible enough that can capture provenance semantics for so many domains. In general, their model is an ontology that they present. They base their work on Bunge's theory [78] where some of the important properties are defined to be highly coupled with any provenance data. Such properties are 'state, event and history' (that says an event is a change of state in any entity), 'action, agent, time and space' (that says an event occurs when it acts upon another entity and happens in a time-space). The W7 model further explains the meaning of all Ws besides defining provenance (that provenance is a tuple of all Ws). They mention, "What" denotes any event that affects any data item. "Where" means the location information of the event. Clearly, "When" refers the event time. "Who" refers to the agent involved and which refers to the program information related to the event. "How" and why are a bit confusing. In W7 model, how refers to the action (for example a user executed the workflow in a high-performance environment) that generated the event but "Why" refers to the reason (for example the reason to an error could be a file exception) to that event. They show an example of their model for Wikipedia.

Our workflow programming model ProvMod covers all the questions and categories mentioned in these works with even simpler provenance query classification through our proposed taxonomy.

Provenance information for Web data is discussed by Hartig et al. [24]. They proposed a model that is suitable for the web. They present another ontology for Web provenance that considers actor, execution, artifact and other attributes as entities. Their ontology describes the internal relations among those entities. They also present the relationships for data access and creation. Accessing provenance-related metadata is one of their proposal to obtain provenance information. They also presented a number of provenance vocabularies that is from Dublin Core Metadata Terms [79]. They also state that heterogeneous data sources create problems in distinguishing between different providers and find a relation among them, which is the first open question they present. The lack of provenance metadata also generates provenance incompleteness that is mentioned as the second open question in their work. To solve this problem, we present a simple JSON log structure. We consider all the logs in the same log space and users are only identified with a unique

ID. Different sources can also be identified with another property if wanted.

To bridge workflow and data provenance, Koop et al. [80] presented a work that makes use of strong links. The work is mainly on storing intermediate data states that can later be used for data source verification, data reproducibility, caching and sharing. Such a work is the work of Asterdamer et al. [25] that proposed a database style workflow provenance. It is discussed in the next section and our model is influenced by the capabilities of that model.

Querying data provenance is only focused by Karvounarakis et al. [12]. They present a tuple based semiring provenance [81] and develop a query language. Their query language ProQL is based on compact graph-based representation. They present a translation scheme from ProQL to SQL. They present four use cases of graph queries. The use cases are- 1) the derivation of a provenance tuple (or graph), 2) relationships between tuples, 3) results that could be derived from a given mapping and 4) identification of overlapping provenance between two graphs. They also present a number of use cases for tuple annotation computation. They involve- 5) incremental maintenance, 6) lineage, 8) trust issue, 9) rank and 10) access control. Their work only focuses on graph query. On the contrary, ProvMod tries to cover every aspect of provenance questions possible.

The Open Provenance Model (OPM) that was born due to the Provenance Challenge [64]. The first Provenance Challenge was focused to provide the community to give an understanding of provenance enabled systems' capabilities. The second Provenance Challenge was focused on establishing interoperability of systems while exchanging provenance data. This actually gave birth to the OPM in the work of Moreau et al. [19]. The scopes of OPM that the authors describe are- 1) information exchange between systems (through a compatibility layer that offers shared facility), 2) allow developers to extend by building and sharing their tools on that model, 3) defining the model, 4) to offer digital representation of any provenance entity that could be generated by any kind of systems, 5) defining core rules set to apply on provenance to perform any provenance related queries. Other than only presenting the requirements, they also present some non-requirements. Those are- 1) OPM does not focus on the internal representation of the system, 2) OPM does not focus on any computer parsable-syntax related to provenance, 3) it does not focus on specifying any protocol concerning data storage, 4) it does not focus to specify any querying protocol over provenance any repository. The core overview of OPM can be found in [19]. They define nodes as Artifact (immutable piece of state), Process (action or series of actions) and Agent (any entity as a catalyst of a process or control) as well as relationships between them. They define a number of dependencies such as- 1) causal relationship (represented by a directed edge), 2) artifact used by a process, 3) artifacts generated by a process, 4) process triggered by a process, 5) artifact derived from an artifact, and 6) process controlled by agent. They also define Role. The overlapping and hierarchical description and temporal constraints and observation time are also discussed. The third Provenance Challenge showed the need to extend provenance information in OPM entities which is basically necessary for interoperability purposes. So they also present an annotation framework. In our paper, we also discuss which design requirements ProvMod follows or not.

To infer fine-grained data provenance from scientific workflows, a declarative rule-based approach is proposed by Bowers et al. [82]. The work offers a high-level language to apply user-defined rules explicitly on the workflow execution traces. They claim to take the burden of determining provenance dependencies in their work and provide the user to work solely on discovering provenance dependencies. Simply speaking, they try to separate the querying facility completely from other components of the system that is adaptable to OPM [64] and W3C Prov [83]. Inspired by this work, we also keep the provenance analysis portion of ProvMod separate besides offering both offline and real-time analysis.

Several rule-based data provenances tracing algorithms are presented by an approach of Zhang et al. where the algorithms could be applied to trace actual data provenance upon files that were under the threat of breaking customer data protection [84]. Distributed data provenance for large-scale and data-intensive computing systems got the attention of Zhao et al. [85]. They name FusionFS [86] and SPADE [87]. FusionsFS implements distributed metadata management and SPADE uses a graph database to manage and provide support for provenance features. As already mentioned, the use of a graph database can be scaled to big data. Here the difference is, SPADE and FusionFS are focused on data provenance where ProvMod is focused on workflow provenance.

13.4 Workflow Provenance and Querying

Workflow provenance research is the main goal of this work. Here in this section, we present different recent prominent and influential works. Since this section is about (workflow) provenance again, provenance querying is highly involved in this matter again.

Automatic generation of workflow provenance and its importance is discussed in Barga et al. [17]. They argue that workflow provenance data should be automatically generated and proposed a layer-based model. In their work, they mention different levels labelled with L. L0 represents abstract service descriptions where L1 for service instantiation, L2 for data instantiation of workflow, and L3 for run-time provenance for workflow execution. Each level captures abstract activities with a predefined semantics that consists of several components including relations among them. Automatic provenance capture got further attention by Barga et al. [16] where they present an approach the captured information in a relational database system (SQL). They focus on several provenance questions from different angles and proposed a layered model for resulting provenance. Being relational database dependent, their provenance model architecture and schema are very structured for each layer and levels that they consider. On the contrary, our model does not rely on a particular structured log schema. Besides, we take the recommendation of layered architecture and automation in provenance logging.

How to store and query on workflow provenance metadata using RDBMS is discussed in the work of Chebotk et al. [88]. The work presents two schema mapping algorithms to map OWL provenance ontology based on relational databases, two mapping algorithms to map provenance RDF metadata to relational

databases and finally, a schema-independent SPARQL to SQL translation algorithm. Although they present a schema-independent SPARQL schema, it is useful when the system is intended for SQL. ProvMod here targets NoSQL.

Provenance in scientific workflow systems and certain issues are discussed in the work of Davidson et al. [89]. They also try to focus on getting meaningful provenance information through different user views. The goal is to manage provenance of nested scientific data and to understand the difference in provenance for similar data products. They make use of a nested tree structure of XML named as collection-oriented modelling and design (COMAD) [90] to achieve this. A hybrid querying system to analyze nested data and lineage graph is explored by Anand et al. [91]. They claim the existing systems before them, ignored the over structured data including XML. They also try to propose a generic provenance model to handle nested data to can update the semantics whenever necessary. XML is not directly adopted in ProvMod but JSON schema can easily be transformed into XML. Big data systems tend to follow a simpler structure and over structured data will increase the development burden in the long run. So, we try to keep the log schema and structure as simple as possible.

Frame Logic based query language (FLOQ) is proposed in Zhao et al. [92] based on some selected characteristics of workflow provenance. They also describe different queries for retrieving lineage, virtual data relationship, annotation, aggregation and modification. ProvMod does not explicitly propose any query language because it might be felt limited in the long run. So, we give a way to incorporate any query language with ProvMod and show the implementation with Cypher.

The importance of fault tolerance mechanism in provenance is presented in Crawl et al. [29]. The different usages of provenance are very clearly presented in this work. Fault tolerance is a process for error or exception handling in workflows. When a sub-workflow produces an error, all executions that are under a certain actor-is stopped. The actor handles the error itself. In brief, the fault tolerance mechanism is discussed in this work for Kepler [93] system. We do not directly attach any fault tolerant mechanism other than graph database for provenance, but it can easily support any fault-tolerant database with the tool library. So, any fault tolerant data item that is necessary to be read or written, can adapt with such database (CouchDB for example).

A taxonomy of provenance for WfMSs is proposed by Cruz et al. [1]. They also present the lifecycle of provenance that is one of the first of its kinds. They mention composition, execution and analysis are the three phases of workflow lifecycle. Conception and reuse fall under composition phase where distribution and monitoring fall under execution, and visualization and query fall under analysis phase. Such phases not only gives us information about the lifecycle of workflows but also sheds a light of importance on each topic in SWfMs. Later a taxonomy is discussed from every possible angle of a workflow management system. They are about provenance capturing mechanism, accessing such information, subject systems and storage mechanism of provenance. Each topic is described in further details in their work. They also compare a number of workflow systems based on the proposed taxonomy. We highly rely on this taxonomy to describe our model's provenance mechanism and to compare with other notable workflow system's provenance

mechanism. The reader should remind that ProvMod is not a workflow system but holds an automated provenance mechanism. On the other hand, different workflow systems supporting provenance also comes with a provenance mechanism. So, we consider this taxonomy of Cruz et al. to compare the provenance mechanism of ProvMod and any other workflow system's provenance.

Database style workflow provenance is discussed in the work of Amsterdamer et al. [25]. They make use of Pig Latin technology to expose the module functionalities. This way they capture the internal states and fine-grained dependencies of such modules. Their approach also makes it possible to transform a number of graph operations (ZoomIn and ZoomOut) that allow choosing expected granularity. The queries they present are formed in Pig Latin framework. Based on the provenance graph definition, they present the Zoom operation as well as Deletion Propagation. In this work, three different workflows are considered as benchmarks for evaluation and the proposed model is evaluated from different angles. The work relies on SQL database model which is structured but ProvMod is different in that case. The implementation also has to follow Pig Latin framework primarily. So adapting this model with any new system might not be supported after all due to architecture and orientation of a workflow system.

A provenance repository is proposed by Cuevas-Vicenttin et al. [41]. They call it PBase and based on ProvONE and extends the W3C PROV standard [83]. It leverages the power of Neo4j graph database [33] and offers querying provenance through it. ProvONE was added to VisTrails [94] through XML serialization. Other workflow systems can be supported by ProvONE but need the use of necessary wrappers. They introduce several query categories such as- lineage, execution analysis, search and statistical queries. They incorporate their work with NoSQL database technology and thus integrates Neo4j as it is focused on graph data. Several Cypher query specification is provided in this work. They mention finding an unsatisfactory performance of Neo4j when querying large traces. An alternative could be of adding new indexing and encoding techniques that the state. They achieved major performance by applying tree cover encoding for readability queries by Agarwal [95]. Their prototype only supports VisTrails exported XML trace files for creating new databases. PBaseGUI can be used through RESTful Web Services [36]. The user interface can be used to query the stored traces. They mention finding graph similarity and complex keyword search to be their future work. They also plan to support other workflow systems as well as RDF/SPARQL graph databases as alternatives. PBase has some similarity with ProvMod in a sense that it made use of Neo4j graph database. Although, it depends on XML in some of the parts. ProvMod does not depend on any such data structure other than JSON at all. Also, they propose a set of provenance questions categories. They do not strictly define how to put any questions under a certain category but only provide some examples for each category. Our proposed provenance question taxonomy not only covers those questions but more about provenance in any system. Also, most of the questions proposed in this work fall under only the graph related questions in our work. They also provide Cypher queries which seem to be deprecated in the latest version of Neo4j. We provide our questions on a template basis so that such a situation can be avoided in the future.

13.5 Analytic Provenance: Querying with Visualization

Analytic provenance is another topic that has emerged from workflow provenance and involves data visualization, which is also another research domain. It is highly related to auditing a workflow system in real-time or so. Important data visualization and better way to visualize any information are also important research topics in this research category.

One of the earliest works (specifically applications) on analytic provenance is VisTrails by Callahan et al. [96] that clarifies the importance of visualization in data analysis and exploration. VisTrails captures the provenance information about the visualization process and manipulated data, it offers reproducibility. It is focused on the visualization pipelines and processes, but not directly provenance modelling. Rather it makes use of provenance in another sense for visualization. VisTrails takes care of the data and metadata of visualizations. So, it works with the provenance of visualization. It is generally good for direct image processing pipeline based research. ProvMod may consider any tool or process as a module in the provenance graph from any domain.

Rio et al. [43] in their work, shed light on the importance of visualization support for provenance. The combination of provenance and visualization is very crucial to get an intermediate and final understanding of the derived results from the systems in an easy manner, although rarely done together that they mention. Alongside, they also propose tool Probe-It for scientific provenance visualization. The tool provides different views for provenance visualization. Result view (for the final and intermediate data results), justification view (for metadata with process traces) and provenance view (for source and usage information). Their work seems to be the very beginning of provenance visualization although not that older. The views represent three basic provenance questions from another angle. ProvMod covers all the questions proposed by their work and even more. The visualization of ProvMod is also designed as a service and can be ported to any new system.

A visual scientific workflow management system is proposed by Lin et al. [46]. They propose such a tool with a reference architecture. They mention seven key architectural requirements for any scientific workflow management systems. They are- user interaction support with interface customizability, reproducibility, heterogeneous service and software tool integration, heterogeneous data management, high-end computing support, workflow monitoring and failure management and interoperability. They also present the reference architecture in a layered manner. Their proposed system VIEW has several advantages that they state; such as- services are loosely coupled, services are given abstraction and autonomy, can be reused, services are discoverable and interoperable. Although they focus on building a visualization supported system, they mention it to be very primary and only contains workflow status based information in the reporting. Our model comes with Kibana for offering visualization. The visualizations can be used to create dashboards, shared and reused later by other users. Kibana also relies on Elasticsearch for real-time log streaming and that is managed through distributed data management. These two services are also loosely coupled with the

model.

A navigation model for exploring workflow graphs is proposed by Anand et al. [97]. The model focuses on visualizing provenance traces only, with directed graphs using provenance browser [93]. Provenance Browser proposed by Anand et al. [44, 98] supports high-level query language QLP [99] but again, it is only limited to presenting the workflow graph in a basic manner. ProvMod, on the other hand, offers workflow regeneration through graph database features. So, advanced queries and graph analysis is possible beyond simple and limited features. Also, only graph related provenance is emphasized in Provenance Browser but no other monitoring features.

Pro Viewer is a graph visualization tool that enables interactive exploration of provenance data, proposed by Kohwalter et al. [45]. This tool provides several visualization features related to graph (graph merge, collapses and filters, temporal and spatial layout). Basically, the tool is built for visualizing geolocation based provenance data. The visualization features of Pro Viewer is limited and the questions the features are answering is already covered by our proposed taxonomy. Such limited visualization might not fit into every problem scenario.

Only network data visualization is focused by Chen et al. [100] where the graph is of great advantages. Big data provenance and visualization are focused on the work of Chen et al. [48] where the graph visualization is again promoted. The history chain to show the relationship between different data products is done by complex graphs. Another feature is partition provenance that presents information about partial provenance graph and rearranges the graph components in temporal order. This work again focuses on only the graph related question from our proposed provenance question classification. It also emphasizes only one instance of that category that is about searching clusters in the provenance graph.

Survey of analytics provenance is performed by Nguyen et al. [101]. They present some survey on provenance capturing mechanism. For visual provenance analytics, we come to know about different methods (representation of states, actions and layouts of actions). How time can also be encoded with a provenance graph is well referenced in this work. They also present thy way of integrating provenance space and data space by replacing graph nodes with visual blocks. Reasoning process visualization is also discussed. The referred approaches try to create compact visualizations which is a matter of further study and evaluation. To support analytic reasoning, such systems should offer navigation and less recalling should be needed. Also, systems should be able to reuse performed analysis. Reasoning processes can also be graphically documented inside the systems. Such visualization services should also support collaboration and presentation. SensePath is a tool for analytic provenance created by Nguyen et al. [47] that features several views to help user working with provenance such as, timeline (for temporally ordered visualization), browser view (web page where a certain action was performed), replay (to show screen capture video) and transcription (detailed information about a selected action) view. The work is more of a tool (superficially prototype) than a provenance model and an approach to offer a generic user action capturing system emphasized on HCI. Comparing with our provenance question taxonomy, SensePath is also limited in what it is offering to answer provenance questions.

Provenance in visual data analysis was characterized by Ragan et al. [102] for helping researchers with organizing their work. First, they discuss the importance of provenance in workflows systems. The different perspectives of provenance they mention are- better workflow management, tracing data, graphically representing the provenance, the provenance of interaction and discovering insights from provenance data. An organizational framework that they propose classifies provenance information into data (history of changes in the system), visualization (history of visualizations), interaction (history of user actions), insight (history of discoveries) and rationale (history of reasoning). The purposes of provenance are also classified into recall (maintenance of memory/awareness related to states), replication (reproducibility of different steps and components), action recovery (maintenance of action history), collaborative communication (sharing information), presentation (displaying insights) and meta-analysis (reviewing the process). They also review recent works based on their framework. The analytic provenance, its process, interaction and insight all these topics are discussed by North [103] briefly but very concisely and states, this visual analytics is related to other fields such as scientific information visualization, and Human-Computer Interaction (HCI). They state, the final results of any analysis are of great value, but the intermediate analysis of data products is also not negligible. They contain important insights into the reasoning of the process. Analytic provenance can be considered as a foundation of the science of visual analytics that they also mention. While discussing this topic, there are several key questions researchers can ask. They are, Perceive (how is information visually provided to the end-user for better reasoning?), capture (what type of user interaction to be captured and the level of semantic information to be saved?), Encode (How to save the user interaction and what is the mechanism?), recover (how to recover the user's reasoning process?) and reuse (how a visual analytics system can help user through learning?). These works can lead to creating better and relevant visualization technique which is a future work of ProvMod.

13.6 Software Engineering

We also present further deeper use cases of our approach. To do so, we consider two very important problems. One of them is clone detection is software systems. Clone detection is a very important research problem. It is important for ensuring a better design for optimized security and fewer bugs in any system. The important is discussed widely by Roy et al. [104, 105].

We take into account, the NiCad clone detector tool [50, 51]. It is a prominent tool for clone detection. We use the latest version of NiCad (version 5 at the time of writing) in our work to merge clone analysis with our approach.

Besides, we consider another bigger problem domain about software architecture extraction from legacy systems. This is a research topic of high demand and different methods are also present. One such method is dynamic analysis [106].

Again, for the very basic architecture extraction, we follow some of the steps of Fittaku et al. [107]. We

follow the approach of instrumentation for modifying the target system and get live traces fo to function calls. Then using our proposed approach, we analyze the extracted information. We choose CRHM as the legacy system [54]. It is a legacy system and no more supports a modern computing environment. It was developed in Borland C++.

14 Discussions

The programming model ProvMod we proposed is oriented to answer provenance questions. One of the fundamental goal of our work was to gather all kinds of provenance questions from every possible angle in a formal manner, but not just randomly collecting them from existing works. That was done by proposing an extendable provenance question taxonomy. The taxonomy classifies provenance questions based on a very simple criterion. The principal category in this taxonomy is 'graph related questions' that proposed three elementary queries. These queries are generated by observing the most basic provenance graph model. These queries can be further used and combined to derive many other provenance graph related questions.

The model ProvMod leverages the proposed provenance question taxonomy and stands on an automated provenance graph based programming model. Being automated, the users can easily implement workflow while getting automated provenance logs. The workflow programming model being oriented to answer any provenance questions, any new tools or service based workflow systems can be integrated with our model. All ProvMod features are then brought to the target system. We show the ease of integration by integrating ProvMod with Apache Taverna and call the whole new system ProVerna. Although the developers have full Independence to use ProvMod, we provide them with a tool integration coding template to help develop tools faster.

The performance of ProvMod is analyzed from various angles, using three kinds of workflows. We specifically investigate whether there is any effect on the model due to a certain type of graph expansion. Also, a benchmark is used for evaluating every system components of our model. For all these purposes, we use different workflows from Bioinformatics and Plant Phenotyping research. We also evaluate the completeness the provenance questions taxonomy, which is the most important part.

Our model is implemented using cutting-edge technologies such as Neo4j Graph Database, ELK Stack (Elasticsearch and Kibana specifically) and so on. Based on the implementation, we demonstrate the provenance mechanism standing on prominent works. The implementation features are also evaluated. Finally, we also evaluate the usability of our system for different use cases and find a good score.

Although the work covers a wide range of topics, the work is just a beginning. Provenance, workflow, data analytics and visualization, graph analytics, log analytics and programming model design are not an easy task and has their own grand areas of research. So due to different limitations, we cannot claim our work is the most perfect or best of its kind. We rather figured out different points that immediately leads to very important future research. For example, the architecture we proposed can be implemented with a variety of protocols. The protocols can be selected targeting the network data and user goals. So, which protocol

is best for a certain use-case is an important research matter. We incorporate NoSQL Database technology with our work and fault tolerance is a big concern for Big Data. Also, different graph database support can be incorporated to perform a comparative study to see which one gives the best performance. We do not provide the users with different tools, but an interface to integrate existing tools while offering automated provenance and modern provenance analytic features. For that reason, we do not focus on distributed computing tools. Still, the log analytics can be done in a distributed manner through ELK Stack. That is more of a configuration than research. Still, there may be important questions about analyzing provenance logs in a distributed manner. Finally, we presented visualization techniques for each provenance questions. Here our goal was to answer them with visualizations but not to find the best visualization technique, which could be another important future research.

15 Conclusion and Future Work

We propose a comprehensive provenance query classification based on prominent research works in provenance. Based on the taxonomy of provenance questions, we build a workflow programming model ProvMod that is oriented to answer workflow provenance questions. During the design and implementation of the model, a number of recommended features to design such a model is taken into account from the state of the art research works. We also evaluate the model after integrating ProvMod with Apache Taverna. ProvMod offers reduced burden in designing workflow, integrating provenance in any workflow system, graph database support, custom and NoSQL logging, less performance overhead and real-time monitoring and data visualization services to answer provenance questions. Alongside the theoretical model we proposed, we also implemented the model using various cutting-edge technologies. We integrated the model with existing prominent workflow system Apache Taverna. A comprehensive comparison of the completeness of the proposed provenance query classification and different features were performed from different angles. We also conduct a performance analysis of the model with several Bioinformatics and Phenotyping analysis workflows. A user study was conducted engaging users from various levels of experience to assess the usability of the model. The model can also be used to extend various research. We show two such use case by applying our model on clone detection in software systems and architecture extraction of legacy systems.

Our proposed work can lead to different long term research contributions in the field of Data Science, Big Data, Data Analytics, Data Mining, Machine Learning, Deep Learning, Software Engineering and so on. As our immediate next work, we planned to implement the model in R. It will offer language-specific flexibility to use the model and can even ignore some implementation layers. Thus the model may work even faster for the target environment. Java and other prominent programming languages can be also the targets of implementation.

Although that is a good idea, language-centric implementation is burdensome. So, we emphasize creating an automated wrapper framework that will keep using the proposed model and convert existing tools into a ProvMod module. Our proposed coding template can be used to do so with ease. The developers can also build their own template at will.

Our model is solely based on Graph Database. It is a matter of fact that, users who are likely to use SQL or SQL based databases may find using the model bit complex at first. Besides, Graph Database might not show the expected performance in all the cases. So, in the future, we plan to work on integrating SQL based databases with the support of distributed computing.

Another point of view is, other Graph Databases may outperform Neo4j Database technology. Finding

the answer to the question- 'which Graph Database is best in which scenario?' can lead to a comparative study.

ProvMod used only the HTTP protocol in its architecture for the initial implementation and experimentation. For certain scenario and data pipeline categories, 'which protocol works best?' is another important research question if we want to ship the model completely to the cloud platform to offer a cloud service.

Next, a whole number of users can be engaged to use the model over a quite large amount of time. Such usage will provide us with big crowd data. Analyzing that data will result in important usage patterns for a better user-oriented recommendation, job handling and resource management. Again, such usage data also holds workflow execution information which is important. It can be used to figure out certain pipeline configuration to error, configuration to data products and configuration to performance information. For both of these two research goals, Machine Learning and Deep Learning techniques will be of great necessity. A large number of users can also be engaged in user studies that will represent a solid and stronger usability score of our model.

Integrating our model with other prominent workflows can be another applied topic of future research works. It may lead to a generic and easy to plug-in workflow architecture for future data science domain. Ultimately, cross-domain research will be even more flexible. While different domains cross together, we can justify- how complete our provenance questions taxonomy is.

In this work, we recommended different possible data visualization techniques for different provenance questions. We completely give the users to customize or select the visualizations. However, it is an important topic of HCI research to find out which visualization is best for a certain type of questions. It will also lead to a backward regeneration of the provenance questions taxonomy on the basis of data visualization.

Finally, as we showed our model's application on two Software Engineering use cases (clone detection and architecture extraction), the research can also be taken to such core Software Engineering topics. Graph-based clone detection in software systems using graph metadata generated by the model could be very useful. Such graph metadata can also be used to extract even granular or high-level software architecture standing on the basis of our study. In conclusion, the research study we present in this thesis and the outcome 'ProvMod' can heavily contribute to future research.

References

[1] Sérgio Manuel Serra da Cruz, Maria Luiza M. Campos, and Marta Mattoso. Towards a taxonomy of provenance in scientific workflow management systems. *2009 Congress on Services - I*, pages 259–266, 2009.

[2] Y L Simmhan, B Plale, and D Gannon. A survey of data provenance in e-science. *ACM SIGMOD Record*, 34(3):31, 2005.

[3] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance techniques technical report iub-cs-tr618. *Science*, 47405(3):1–25, 2005.

[4] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A framework for collecting provenance in data-centric scientific workflows. *Web Services, 2006. ICWS'06. International Conference on*, pages 427–436, 2006.

[5] Wil M.P. Van der Aalst, Boudewijn F. Van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and A. J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

[6] Sb Davidson and Juliana Freire. Provenance and scientific workflows: challenges and opportunities. *Proceedings of the 2008 ACM SIGMOD*, pages 1–6, 2008.

[7] Im Abbadi and John Lyle. Challenges for provenance in cloud computing. *USENIX Workshop on the Theory and Practice of Provenance (TaPP'11)*, 2011.

[8] Olive Qing Zhang, Markus Kirchber, Ryan K.L. Ko, and Bu Sung Lee. How to track your data: The case for cloud computing. 2012.

[9] Adriane Chapman and HV Jagadish. Issues in building practical provenance systems. *IEEE Data Eng. Bull.*, 30(4):38–43, 2007.

[10] Boris Glavic. Big data provenance: Challenges and implications for benchmarking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8163 LNCS:72–80, 2014.

[11] Flavio Costa, Vítor Silva, Daniel de Oliveira, Kary Ocaña, Eduardo Ogasawara, Jonas Dias, and Marta Mattoso. Capturing and querying workflow runtime provenance with PROV. *Proceedings of the Joint EDBT/ICDT 2013 Workshops on - EDBT '13*, page 282, 2013.

[12] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, 1(212):951, 2010.

[13] Sérgio Manuel Serra da Cruz, Maria Luiza M Campos, and Marta Mattoso. Towards a taxonomy of provenance in scientific workflow management systems. In *Services-I, 2009 World Conference on*, pages 259–266. IEEE, 2009.

[14] Yogesh L Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.

[15] Lucian Carata, Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Margo Selter, and Andy Hopper. A primer on provenance. *Communications of the ACM*, 57(5):52–60, 2014.

[16] Roger S. Barga and Luciano A. Digiampietri. Automatic capture and efficient storage of e-science experiment provenance. *Concurrency Computation Practice and Experience*, 20(5):419–429, 2008.

[17] R. Barga and L. Digiampietri. Automatic generation of workflow provenance. *Provenance and Annotation of Data*, pages 1–9, 2006.

[18] Ashiq Imran, Rajeev Agrawal, Jessie Walker, and Anthony Gomes. A layer based architecture for provenance in big data. *2014 IEEE International Conference on Big Data (Big Data)*, pages 29–31, 2014.

[19] Luc Moreau, Juliana Freire, Joe Futrelle, Robert McGrath, Jim Myers, and Patrick Paulson. The Open Provenance Model. *Future Generation Computer Systems*, 27(6):743–756, 2011.

[20] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T. Silva. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[21] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(suppl_2):W729–W732, 2006.

[22] M David Allen, Adriane Chapman, Barbara Blaustein, and Len Seligman. Capturing provenance in the wild. *Management*, pages 98–101, 2010.

[23] Wang Chiew Tan. Research problems in data provenance. *IEEE Data Eng. Bull.*, 27(4):45–52, 2004.

[24] Olaf Hartig and Olaf Hartig. Provenance information in the web of data. *Proceedings of the Linked Data on the Web LDOW Workshop at WWW*, 39(27):1–9, 2009.

[25] Yael Amsterdamer, Susan B. Davidson, Daniel Deutch, Tova Milo, Julia Stoyanovich, and Val Tannen. Putting Lipstick on Pig: Enabling database-style workflow provenance. pages 346–357, 2011.

[26] Emir M Bahsi, Emrah Ceyhan, and Tevfik Kosar. Conditional workflow management: A survey and analysis. In *Dynamic Computational Workflows Discovery Optimization and Scheduling*, volume 15, pages 283–297, 2007.

[27] Shawn Bowers, Timothy Mcphillips, Bertram Ludascher, Shirley Cohen, Susan B Davidson, and Bertram Ludäscher. A model for user-oriented data provenance in pipelined scientific workflows. *Lecture Notes in Computer Science*, 4145(4145):133–147, 2006.

[28] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Data provenance: Some basic issues. *Computer Vision ACCV 2010*, 1974(December):87–93, 2000.

[29] D. Crawl and I. Altintas. A provenance-based fault tolerance mechanism for scientific workflows. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5272:152–159, 2008.

[30] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM, 2010.

[31] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[32] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[33] Jim Webber. A programmatic introduction to neo4j. In *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pages 217–218. ACM, 2012.

[34] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* " O'Reilly Media, Inc.", 2015.

[35] Yuvraj Gupta. *Kibana Essentials.* Packt Publishing Ltd, 2015.

[36] Jason H Christensen. Using restful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 627–634. ACM, 2009.

[37] Simon Andrews. FastQC a quality control tool for high throughput sequence data, 2015.

[38] X Chen and PF Sullivan. Single nucleotide polymorphism genotyping: biochemistry, protocol, cost and throughput. *The pharmacogenomics journal*, 3(2):77, 2003.

[39] Peter Buneman and Sb Davidson. Data provenance–the foundation of data quality. *Book Data provenance–the foundation of data quality'*, pages 1–8, 2013.

[40] Sudha Ram and Jun Liu. A new perspective on semantics of data provenance. *CEUR Workshop Proceedings*, 526:1–6, 2009.

[41] Víctor Cuevas-Vicenttín, Parisa Kianmajd, Bertram Ludäscher, Paolo Missier, Fernando Chirigati, Yaxing Wei, David Koop, and Saumen Dey. The PBase scientific workflow provenance repository. *International Journal of Digital Curation*, 9(2):28–38, 2014.

[42] Devarshi Ghoshal and Beth Plale. Provenance from log files: a BigData problem. *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 290–297, 2013.

[43] Nicholas Del Rio and Paulo Pinheiro. Probe-It! visualization support for provenance. *Advances in Visual Computing*, 4842:732–741, 2007.

[44] MK Anand. Provenance browser: Displaying and querying scientific workflow provenance graphs. *Data Engineering (ICDE)*, pages 1201–1204, 2010.

[45] Troy Kohwalter, Thiago Oliveira, Juliana Freire, Esteban Clua, and Leonardo Murta. Provenance and annotation of data and processes. 7525:71–82, 2012.

[46] Cui Lin Cui Lin, Shiyong Lu Shiyong Lu, Zhaoqiang Lai Zhaoqiang Lai, a. Chebotko, Xubo Fei Xubo Fei, Jing Hua Jing Hua, and F. Fotouhi. Service-oriented architecture for view: A visual scientific workflow management system. *2008 IEEE International Conference on Services Computing*, 1:335–342, 2008.

[47] Phong H. Nguyen, Kai Xu, Ashley Wheat, B. L.William Wong, Simon Attfield, and Bob Fields. SensePath: Understanding the sensemaking process through analytic provenance. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):41–50, 2016.

[48] Peng Chen and Beth A. Plale. Big data provenance analysis and visualization. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 797–800, 2015.

[49] Sandra G Hart. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50, pages 904–908. Sage Publications Sage CA: Los Angeles, CA, 2006.

[50] 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.

[51] 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.

[52] Erich Gamma and T Eggenschwiler. JHotDraw Project, 2005.

[53] Jennifer Hillman-Jackson, Dave Clements, Daniel Blankenberg, James Taylor, Anton Nekrutenko, and Galaxy Team. Using galaxy to perform large-scale interactive data analyses. *Current protocols in bioinformatics*, pages 10–5, 2012.

[54] JW Pomeroy, DM Gray, T Brown, NR Hedstrom, WL Quinton, RJ Granger, and SK Carey. The cold regions hydrological model: a platform for basing process representation and model structure on physical evidence. *Hydrological Processes: An International Journal*, 21(19):2650–2667, 2007.

[55] 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.

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

[57] Lavanya Ramakrishnan and Beth Plale. A multi-dimensional classification model for scientific workflow characteristics. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science - Wands '10*, pages 1–12, 2010.

[58] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM SIGMOD Record*, pages 44–49, 2005.

[59] Philip A Laplante. *What every engineer should know about software engineering*. CRC Press, 2007.

[60] 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, dec 2015.

[61] Shawn Bowers. Scientific workflow, provenance, and data modeling challenges and approaches. *Journal on Data Semantics*, 1(1):19–30, 2012.

[62] Jun Zhao, Carole Goble, Robert Stevens, and Sean Bechhofer. Semantically linking and browsing provenance logs for e-science. *Semantics of a Networked World. Semantics for Grid Databases*, pages 158–176, 2004.

[63] Andreas Pieber. *Flexible engineering environment integration for (software+) development teams*. 2010.

[64] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, et al. The open provenance model core specification (v1. 1). *Future generation computer systems*, 27(6):743–756, 2011.

[65] Eleni Gessiou, Vasilis Pappas, Elias Athanasopoulos, Angelos D. Keromytis, and Sotiris Ioannidis. Towards a universal data provenance framework using dynamic instrumentation. *IFIP Advances in Information and Communication Technology*, 376 AICT:103–114, 2012.

[66] Brendan Gregg and Jim Mauro. *DTrace: dynamic tracing in oracle Solaris, Mac OS X and freeBSD*. Prentice Hall Professional, 2011.

[67] Mike Owens and Grant Allen. *SQLite*. Springer, 2010.

[68] B Arab, Dieter Gawlick, V Radhakrishnan, H Guo, and Boris Glavic. A generic provenance middleware for database queries, updates, and transactions. *Proceedings of the 6th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, page 8, 2014.

[69] J Nathan Foster and Grigoris Karvounarakis. Provenance and data synchronization. *IEEE Data Eng. Bull.*, 30(4):13–21, 2007.

[70] Aaron Bohannon, J Nathan Foster, Benjamin C Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang: resourceful lenses for string data. In *ACM SIGPLAN Notices*, volume 43, pages 407–419. ACM, 2008.

[71] Todd J Green, Grigoris Karvounarakis, Nicholas E Taylor, Olivier Biton, Zachary G Ives, and Val Tannen. Orchestra: facilitating collaborative data sharing. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1131–1133. ACM, 2007.

[72] James Cheney. Program slicing and data provenance. *IEEE Data Eng. Bull.*, 30(4):22–28, 2007.

[73] Stijn Vansummeren and James Cheney. Recording provenance for sql queries and updates. 2007.

[74] Adriane P Chapman, Hosagrahar V Jagadish, and Prakash Ramanan. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 993–1006. ACM, 2008.

[75] Boris Glavic and Kr Dittrich. Data provenance: A categorization of existing approaches. *Btw*, pages 227–241, 2007.

[76] Yong Zhao, Michael Wilde, and Ian Foster. Applying the virtual data provenance model. *Provenance and Annotation of Data. Springer Berlin Heidelberg*, pages 148–161, 2006.

[77] Peter Buneman, Sanjeev Khanna, W Tan, and Tan Wang-Chiew. Why and where: A characterization of data provenance. *Proceedings of International Conference on Database Theory (ICDT)*, 1973(January):316–330, 2001.

[78] Mario Bunge. *Treatise on basic philosophy: Ontology I: the furniture of the world*, volume 3. Springer Science & Business Media, 1977.

[79] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. Dublin core metadata for resource discovery. Technical report, 1998.

[80] David Koop, Emanuele Santos, Bela Bauer, Matthias Troyer, Juliana Freire, and Cláudio T. Silva. Bridging workflow and data provenance using strong links. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6187 LNCS:397–415, 2010.

[81] Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 31–40. ACM, 2007.

[82] Shawn Bowers, Timothy McPhillips, and Bertram Ludäscher. Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7525 LNCS:82–96, 2012.

[83] Paolo Missier, Khalid Belhajjame, and James Cheney. The w3c prov family of specifications for modelling provenance metadata. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 773–776. ACM, 2013.

[84] Olive Qing Zhang, Ryan KL Ko, Markus Kirchberg, Chun Hui Suen, Peter Jagadpramana, and Bu Sung Lee. How to track your data: Rule-based data provenance tracing algorithms. pages 1429–1437, 2012.

[85] Dongfang Zhao, Chen Shou, Tanu Maliky, and Ioan Raicu. Distributed data provenance for large-scale data-intensive computing. *Proceedings - IEEE International Conference on Cluster Computing, ICCC*, (1), 2013.

[86] Dongfang Zhao and Ioan Raicu. Distributed file systems for exascale computing. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC12), doctoral showcase*, pages 267–276, 2012.

[87] Ashish Gehani and Dawood Tariq. SPADE: support for provenance auditing in distributed environments. In *Proceedings of the 13th International Middleware Conference*, pages 101–120. Springer-Verlag New York, Inc., 2012.

[88] Artem Chebotko, Xubo Fei, Cui Lin, Shiyong Lu, and Farshad Fotouhi. Storing and querying scientific workflow provenance metadata using an rdbms. *Proceedings - e-Science 2007, 3rd IEEE International Conference on e-Science and Grid Computing*, pages 611–618, 2007.

[89] S Davidson, S Cohen-Boulakia, A Eyal, B Ludascher, T Mcphillips, S Bowers, M K Anand, and J Freire. Provenance in scientific workflow systems. *Data Engineering Bulletin*, 30(4):1–7, 2007.

[90] Edward A Lee and Thomas M Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[91] Manish Kumar Anand, Shawn Bowers, Timothy McPhillips, and Bertram Ludäscher. Exploring scientific workflow provenance using hybrid queries over nested data and lineage graphs. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5566 LNCS:237–254, 2009.

[92] Yong Zhao and Shiyong Lu. A logic programming approach to scientific workflow. pages 31–44, 2008.

[93] 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.

[94] 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.

[95] Rakesh Agrawal, Alexander Borgida, and Hosagrahar Visvesvaraya Jagadish. *Efficient management of transitive relationships in large data and knowledge bases*, volume 18. ACM, 1989.

[96] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, and T Silva Huy T Vo. Using provenance to streamline data exploration through visualization. 2006.

[97] M.K. Anand, S. Bowers, and B. Ludäscher. A navigation model for exploring scientific workflow provenance graphs. *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, WORKS '09, in Conjunction with SC 2009*, 2009.

[98] M K Anand Bowers, S., Ludascher, B. Techniques for efficiently querying scientific workflow provenance graphs. *International Conference on Extending Database Technology (EDBT)*, pages 287–298, 2010.

[99] Chunhyeok Lim, Shiyong Lu, Artem Chebotko, and Farshad Fotouhi. OPQL: A first OPM-level query language for scientific workflow provenance. In *Services Computing (SCC), 2011 IEEE International Conference on*, pages 136–143. IEEE, 2011.

[100] Peng Chen, Beth Plale, You-Wei Cheah, Devarshi Ghoshal, Scott Jensen, and Yuan Luo. Visualization of network data provenance. *2012 19th International Conference on High Performance Computing*, (DataMASS):1–9, 2012.

[101] PH Nguyen, K Xu, and BL Wong. A survey of analytic provenance. *Middlesex University*, 2016.

[102] Eric D. Ragan, Alex Endert, Jibonananda Sanyal, and Jian Chen. Characterizing provenance in visualization and data analysis: An organizational framework of provenance types and purposes. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):31–40, 2016.

[103] Chris North, Richard May, Remco Chang, Bill Pike, Alex Endert, Glenn A. Fink, and Wenwen Dou. Analytic Provenance: Process + Interaction + Insight. *29th Annual CHI Conference on Human Factors in Computing Systems, CHI 2011*, pages 33–36, 2011.

[104] Chanchal K Roy, James R Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of computer programming*, 74(7):470–495, 2009.

[105] Chanchal Kumar Roy and James R Cordy. A survey on software clone detection research. *Queens School of Computing TR*, 541(115):64–68, 2007.

[106] Eleni Stroulia and Tarja Systä. Dynamic analysis for reverse engineering and program understanding. *ACM SIGAPP Applied Computing Review*, 10(1):8–17, 2002.

[107] Florian Fittkau, Jan Waller, Christian Wulf, and Wilhelm Hasselbring. Live trace visualization for comprehending large software landscapes: The ExplorViz approach. 2013.