SUPPORTING USER STEERING IN LARGE-SCALE WORKFLOWS WITH PROVENANCE DATA

Renan Francisco Santos Souza

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Patrick Valduriez

Rio de Janeiro
Dezembro de 2019

SUPPORTING USER STEERING IN LARGE-SCALE WORKFLOWS WITH PROVENANCE DATA

Renan Francisco Santos Souza

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Marta Lima de Queirós Mattoso
Patrick Valduriez

Aprovada por: Profª. Marta Lima de Queirós Mattoso
Prof. Patrick Valduriez
Dr. Marco Aurelio Stelmar Netto
Prof. Alexandre de Assis Bento Lima
Profª. Vanessa Braganholo Murta
Profª. Lúcia Maria de Assumpção Drummond

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2019

Souza, Renan Francisco Santos

Supporting User Steering in Large-scale Workflows with Provenance Data/Renan Francisco Santos Souza. – Rio de Janeiro: UFRJ/COPPE, 2019.

XIII, 139 p.: il.; 29, 7cm.

Orientadores: Marta Lima de Queirós Mattoso

Patrick Valduriez

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2019.

Referências Bibliográficas: p. 111 – 128.

1. User Steering. 2. Data Provenance. 3. Large-scale Workflows. I. Mattoso, Marta Lima de Queirós *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Agradeço:

A Deus, porque até aqui nos ajudou o Senhor.

À Gisele, minha esposa e motivação maior, por estar ao meu lado desde o primeiro dia da graduação até o último dia do doutorado. Seu amor, apoio e compreensão têm sido essencial para alcançar meus objetivos.

À minha mãe por todos sacrifícios para prover a melhor educação que pôde. Seus conselhos e orações foram especialmente importantes nesses últimos meses.

Agradeço ao Marcos, ao meu pai, à Deuseni, a toda minha família e à família da Gisele pela força e compreensão.

À minha orientadora, Professora Marta Mattoso, que me orientou em todos os momentos, persistentemente, desde o início do mestrado até aqui. Com ela, muito além da pesquisa, aprendi o significado de ensino. Agradeço pela paciência, esforço e cuidado que teve comigo durante todos esses difíceis anos.

Ao meu coorientador, Professor Patrick Valduriez, pela orientação objetiva e precisa. Agradeço-o também pelo apoio com equipamentos (Grid5000) usados nos experimentos e pelo suporte financeiro enquanto estive no período sanduíche na França.

Aos pesquisadores e colaboradores que estão ou estiveram no Laboratório da IBM Research no Brasil. Dentre os quais, alguns tornaram-se amigos próximos que participaram do meu dia-a-dia na labuta dividida entre o trabalho e o doutorado, compartilhando seus conhecimentos e palavras de ânimo. Por medo de cometer a injustiça de não mencionar alguém, ficam aqui meus sinceros agradecimentos a todos os amigos. Agradeço também aos gerentes do BRL que sempre incentivaram meu doutorado, me orientaram e permitiram minha liberação para realização das atividades acadêmicas quando necessário. Agradeço especialmente ao meu gerente Marco Netto que, além disso, aceitou fazer parte do exame de qualificação e da banca desta tese.

Ao Jonas Dias pelas valiosas sugestões dadas durante o exame de qualificação. Aos Professores Vanessa Braganholo Murta e Alexandre Lima pelos conselhos no exame de qualificação e por terem aceitado fazer parte da banca. Agradeço também à Professora Lúcia Drummond por ter aceitado compor a banca.

Aos amigos Vítor Silva e José Camata pela colaboração na pesquisa e ajuda para alcançar os resultados da tese. Aos participantes do projeto SciDISC, especialmente ao Professor Alvaro Coutinho pela orientação, incentivo e parceria nos artigos.

Aos amigos do laboratório LIRMM/Inria em Montpellier.

À equipe administrativa do PESC e do NACAD por toda a ajuda na burocracia. À equipe do NACAD também agradeço pelo suporte com os equipamentos usados nos experimentos (Lobo Carneiro). Aos colaboradores do COPPE-LaTeX, por fornecerem o fonte do template usado para escrever esta tese. Os fontes deste documento encontram-se em https://github.com/renan-souza/phd-thesis.

Finalmente, agradeço a todos meus professores que participaram da minha educação, desde a época do colégio até o doutorado. Certamente tive diversas influências positivas que me inspiraram e ajudaram a chegar até aqui.

A todos, muito, muito obrigado!!! :)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

APOIO A WORKFLOWS DE LARGA ESCALA CONDUZIDOS POR USUÁRIO COM DADOS DE PROVENIÊNCIA

Renan Francisco Santos Souza

Dezembro/2019

Orientadores: Marta Lima de Queirós Mattoso
Patrick Valduriez

Programa: Engenharia de Sistemas e Computação

Workflows em Ciência e Engenharia Computacional (CSE) são de larga escala, requerem execução em ambientes de Processamento de Alto Desempenho (PAD) e têm a natureza exploratória da ciência. Durante a execução, que costuma demorar horas ou dias, usuários precisam conduzir o *workflow* analisando-o e adaptando-o dinamicamente para melhorar a qualidade dos resultados ou reduzir o tempo de execução. Entretanto, eles fazem diversas ações de condução do *workflow*, que precisam ser rastreadas. Caso contrário, eles têm dificuldade de entender como e o que precisa ser conduzido, podem conduzir de forma errada, pode ser difícil explicar resultados que foram consequências das ações e pode ser impossível de reproduzir os resultados. Para resolver esse problema, esta tese propõe uma abordagem que define os conceitos fundamentais para ações de usuário para condução de workflows; introduz a noção de proveniência de ações de condução; e um diagrama de dados compatível com o padrão W3C PROV para modelar dados de ações de condução com proveniência. Além disso, a abordagem apresenta princípios de projeto de sistemas para gerência de dados de ações de condução através da captura, relacionamento com o restante dos dados do *workflow* e armazenamento eficiente. Duas instâncias dessa abordagem foram projetadas e implementadas: uma para ser adicionada a *scripts* paralelos e a outra é usada em um Sistema Paralelo de Gerência de *Workflows*, as quais são as duas formas típicas de se executar experimentos de CSE em PAD. Através de experimentos com casos reais da indústria de Óleo e Gás, mostra-se que a abordagem permite que usuários entendam como suas ações afetam diretamente os resultados em tempo de execução e também que os princípios de projeto foram essenciais para adicionar sobrecarga desprezível à execução dos *workflows* em PAD.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

SUPPORTING USER STEERING IN LARGE-SCALE WORKFLOWS WITH PROVENANCE DATA

Renan Francisco Santos Souza

December/2019

Advisors: Marta Lima de Queirós Mattoso
Patrick Valduriez

Department: Systems Engineering and Computer Science

Computational Science and Engineering (CSE) workflows are large-scale, require High Performance Computing (HPC) execution, and have the exploratory nature of science. During the long run, which often lasts for hours or days, users need to steer the workflow by dynamically analyzing it and adapting it to improve the quality of results or to reduce the execution time. However, to steer the workflow, users typically perform several interactions (called *user steering actions*), which need to be tracked. Otherwise, users find it harder to understand how and what needs to be steered, they can steer in a misleading way, it can be difficult to explain the results that were consequences of their actions, and it can be impossible to reproduce the results. This thesis addresses this problem by proposing an approach that defines the fundamental concepts for user steering action; introduces the notion of provenance of steering actions; and contemplates a W3C PROV-compliant data diagram to model steering action data with provenance. Also, the approach presents system design principles to enable the management of steering action data by capturing, explicitly relating the actions to the rest of the workflow data, and storing these data efficiently. Two instances of this approach were designed and built: one is a lightweight tool to be plugged into parallel scripts and the other is to be used within a Parallel Workflow Management System, which are the two typical ways to conduct CSE experiments in HPC. Using real use cases in the Oil and Gas industry, the experiments show that the proposed approach enables users to understand how their actions directly affect the workflow results at runtime and that the system design principles were essential to add negligible overhead to the HPC workflows.

Contents

List of Figures

List of Tables

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability, p. 80
AI	Artificial Intelligence, p. 49
CFD	Computational Fluid Dynamics, p. 12
CSE	Computational Science and Engineering, p. 1
DBMS	Database Management System, p. 21
HPC	High Performance Computing, p. 1
HTTP	HyperText Transfer Protocol, p. 40
ML	Machine Learning, p. 13
MPI	Message Passing Interface, p. 40
OLAP	Online Analytical Processing, p. 65
OLTP	Online Transactional Processing, p. 75
O&G	Oil and Gas, p. 1
QoI	Quantities of Interest, p. 2
W3C	World Wide Web Consortium, p. 23
WMS	Parallel Workflow Management System, p. 2

Chapter 1

Introduction

How to enable computational scientists and engineers to monitor and understand their experiments when they are steering them on large-scale computers? This is the central question we address in this thesis. With the astonishing evolution of computer science methods, tools, and hardware, more and more scientists and engineers, from a wide variety of scientific domains like Physics, Agriculture, and Geosciences, have gained access to large-scale computers and software that enabled the conduction of experiments with an ever-growing level of detail and amount of data. This culminated in the development of an emerging interdisciplinary field of computing: Computational Science and Engineering (CSE) [16].

CSE combines mathematical models that simulate natural phenomena, numerical algorithms for solving complex equations, High Performance Computing (HPC) techniques to develop programs that can take advantage of modern parallel hardware, and data science techniques to analyze huge amounts of scientific data [16]. This allows for both remarkable scientific discoveries – like the detection of Gravitational Waves, which led to the Nobel Prize in Physics in 2017 [17] – and game-changing business impact in industries that depend on such experiments to generate revenue – like the Oil and Gas (O&G) industry, which leverages CSE experiments to improve (*e.g.*, increase the precision, reduce the environmental impact) oil exploration and production processes [18].

However, conducting such experiments is far from trivial. They have the exploratory nature of science, meaning that typically there is a huge solution space to be explored. An experiment run often lasts for hours or days, even running on large-scale computers, like HPC machines. That is why the computational scientists and engineers (CSE users) are often applying HPC techniques in their software, aiming to run them in parallel and reduce the execution time. Long execution times means that not only the HPC machines are executing for longer, but also the qualified computational scientists and engineers, whose time is quite valuable, are waiting for longer. So, when investigating new tools to be added to their toolset, these CSE

users typically discard any tool that can add significant execution overhead to their already long experiment executions.

In addition, to ease the complexity of the CSE experiments, a widely adopted practice is to model them using a *scientific workflow* (also known as *large-scale workflow* or *workflow*, for short) abstraction [19]. The workflow abstraction helps to break a large CSE experiment into smaller pieces – such as programs, components, functions, or meaningful sections of a simulation code – interconnected through a dataflow (*i.e.*, the data produced by one are consumed by another). These workflows are typically concretized either as a script, *i.e.*, a sequence of computing commands that often invoke highly parallel libraries, or as workflows managed by a Parallel Workflow Management System (WMS), like Chiron [20] and Pegasus [21]. These are the two typical ways of conducting CSE experiments on large-scale computers.

When modeling the workflows, the CSE users specify the *workflow data*, which are the data processed (*i.e.*, consumed and generated) during the workflow execution, like scientific data files and parameters for the algorithms or numerical methods. By varying the workflow data, the users investigate different hypotheses for their experiments, often in a "what-if" basis, analyzing how the variation affects Quantities of Interest (QoI), like precision, convergence, numerical errors. Thus, during the execution, they need to analyze the workflow data to verify their hypotheses, understand the intermediate results, monitor how variations of data or parameters affect the QoIs, and check on the experiment's health. They analyze the experiment's health both from a domain perspective (*e.g.*, is there any result that dissatisfies a physical constraint?) and from a computational perspective (*e.g.*, is the workflow consuming memory, disk, CPU as expected?). Depending on the results, they intervene by changing the workflow data.

In this context, in 2013, MATTOSO *et al.* [22] suggested using provenance data as a good alternative to provide for such experiments' analyses, even at runtime, which has been successfully explored in real experiments [23–27]. Data provenance (*i.e.*, data lineage) contains a structured record of the data derivation process, that is, how data items are consumed and produced, and their data derivation paths [28]. In workflows, provenance data contain not only this historical record but also data about the computational processes and agents (*e.g.*, users and software) involved in the workflow execution [29].

Moreover, the interaction of users with large-scale CSE experiments can generate major improvements in performance, resource consumption, and quality of results. This idea of putting human intelligence to drive complex computational processes, as is the case of large-scale CSE experiments, is often referred to as "Human in the Loop" [30]. Despite the recent breakthroughs in theory and practice of Artificial Intelligence [31], the complexity of CSE domains makes human knowledge critical

for making decisions during an experiment run. Thus, as highlighted in a recent Department of Energy (DOE) report, putting humans in the loop of CSE experiments is still an open problem [32].

User steering (also known as *computational steering*) is a powerful concept meaning that users are enabled to interactively and dynamically drive a computational process while it is still running (*i.e.*, online, at runtime, during execution). In workflows, *workflow steering* is the ability that allows users to interactively analyze (*e.g.*, inspect, visualize, monitor) or adapt (*e.g.*, tune parameters, change input datasets) the workflow data online [23, 25]. We call *user steering actions* the individual user interactions performed during workflow steering, like asking a monitoring query (in case of *online data analysis*) or tuning a parameter (in case of *online data adaptation*). Workflow steering allows for users to analyze the workflow data at runtime and to intervene in the workflow, also at runtime, to adapt it. The consequences of successful runtime adaptations are achieving workflow results with high accuracy, low error, or high data quality.

The *workflow steering lifecycle* is given by: (i) analyzing the workflow data; (ii) choosing what, when, how to adapt; (iii) adapting the workflow data; and (iv) analyzing the workflow data again to investigate the consequences of the adaptation and returning to the first step until the end of the workflow execution.

Often, one steering action is not enough hence requiring several steering actions to be performed by the users. They repeatedly monitor, fine-tune parameters or perform multiple other steering actions. Every single steering action generates *user steering action data*, which are important information that helps to understand the steering actions and their influence on the workflow data. They consist of data informing: *when* the action happened, *why* the user decided to act, *how* the action occurred, *which* workflow data were analyzed or adapted, *what* was happening before and after the action, *who* acted, and the type of the action itself. The *track of steering actions* is a historical record containing important information that allows understanding the steering actions and their influence on the workflow data, *i.e.*, it is the steering action data properly related to the rest of the workflow data.

However, such steering action data generated at each action are typically not recorded nor are they explicitly related to the rest of the workflow data. Because of this, it is very easy to lose track of the actions, even for experienced users, especially considering the huge amount of data, parallel processing, complex algorithms and software. If users cannot track their steering actions, they find harder to understand how and what needs to be steered, steer in a misleading way (*e.g.*, they can unintentionally tune the same parameter twice), it can be harder to explain results that were consequences of steering actions, and it can be impossible to reproduce the results, jeopardizing the overall experiment and failing to support the workflow

steering lifecycle. Therefore, a critical aspect in the workflow steering lifecycle is to allow for tracking user steering actions to enable users to understand how their steering actions are influencing a running workflow. With such understanding, users have more information available to help them to steer. The importance of allowing for tracking the interaction of users with large-scale workflows has been highlighted in several surveys [19, 32–34].

In our extensive literature review (Chapter 3), we find that the state-of-the-art in the context of supporting CSE experiments lacks concepts and techniques to allow for tracking steering actions in large-scale workflows, which would significantly assist in understanding the steering actions online, which is critical to support the workflow steering lifecycle. As a result, users are left to register the track manually by, for instance, annotating in a separate digital spreadsheet or even in a sheet of paper. Thus, the track is often incomplete, isolated from the workflow data, and not stored in a structured way, which would facilitate data analysis. Therefore, the general problem this thesis addresses is: "how to allow for tracking user steering actions in large-scale workflows?". This is hard because to allow for tracking steering actions, steering action data must be managed, which means *to capture the steering action data, relate them to the workflow data accordingly, and store in a database ready for online data analysis*. There are several challenges associated with managing steering action data. We can group them as follows.

(I) *Steering actions characteristics.* Considering the complexity of the data, the computational tools, methods and software used or developed by the computational scientists and engineers, how to characterize: how users interact with the workflow data, what constitutes the steering action data, and the data relationships between the actions and the rest of the workflow data?

(II) *Enabling the capture and queries.* How to capture the steering action data in a running large-scale workflow and build a database with the steering actions related to the workflow data so users can query it online?

(III) *Efficient capture.* Since attaining high performance in the workflow executions is an essential requirement in CSE, how to manage steering action data while not adding significant execution overhead to the already long execution even in HPC?

The general hypothesis of this thesis is that by managing user steering action data, users can track their actions, which enables online analysis of the steering action data, hence allowing users to understand how their steering actions are influencing a running workflow. Since provenance data management techniques and concepts have been successfully explored to support user steering even in CSE [35–37], we could use provenance one step further for the management of user steering

action data. We can further distinguish user steering action data management between two important aspects: one is to capture steering actions, create the explicit data relationships with the workflow data, and store in a database for online analysis; and the other is to do this while incurring low overhead for efficient performance of the workflow execution. Furthermore, our hypothesis encompasses the two typical ways CSE users conduct their experiments on large-scale computers, *i.e.*, using workflow scripts and WMSs. To validate this hypothesis, we propose *Workflow Steer* (WfSteer), an approach that leverages provenance data to manage user steering action data in large-scale workflows. It resulted from a combination of new concepts, techniques, definitions, and design principles for the management of user steering action data with low execution overhead. More specifically, we can list the following contributions.

(I) A characterization of user steering actions and steering action data, which are the two main concepts introduced in this thesis, along with their corresponding definitions, and definitions on how steering action data explicitly relate to workflow data. We also introduced provenance data management concepts, the notion of provenance of steering actions, and a corresponding data diagram to model steering action data following a widely adopted standard, the W3C PROV. This data diagram gives the data structure for any workflow database that can persist steering action data. Such concepts and definitions can be utilized by workflow scripts or in WMS.

(II) Distributed systems techniques to manage steering action data when users steer a running large-scale workflow, considering the specific requirements for workflow scripts and WMS.

(III) Design principles to maintain low execution overhead while managing steering action data. These principles can be adopted by steering action data management software engineers for building rich workflow databases while adding low execution overhead.

We organize the validation of our hypothesis by considering its two aspects (*i.e.*, to allow for tracking actions and doing it with low execution overhead). For this, we carry out two classes of experiments: one class is to qualitatively investigate whether WfSteer allows for tracking steering actions, using real-world workflows as driving use cases; and the other is to quantitatively evaluate the added data capture overhead. Additionally, since the hypothesis encompasses both workflow scripts and WMS, within each class of experiments we design and build two instances of WfSteer: one, called DfAdapter, is built on DfAnalyzer [27, 35, 38], to support users who use scripts to conduct their experiments; and the other, implemented within Chiron

[26, 39–41], is to support WMSs' users. DfAnalyzer and Chiron were chosen due to their successful unique features of combining runtime provenance data analysis with negligible overhead in CSE workflows [27, 41]. Then, through the practical use of WfSteer using these two instances in real use cases running on large HPC machines, among many other findings, we found that WfSteer in the WMS enabled users to understand how input data were reduced online to yield reduction of 32.4% of the total execution time, hence significantly saving resources; and that DfAdapter allowed users to understand which parameters were tuned that made the workflow finish successfully without memory overflow. We also found that design principles to implement WfSteer, both for scripts and WMS, helped to maintain the execution overhead as low as 1%.

Thesis Organization. During the course of this thesis, some scientific papers in the context of supporting user steering in large-scale workflows have been published [24, 25, 36, 37, 41–48], among which the papers SOUZA *et al.* [25, 37, 42] compose the core contribution of this thesis and drive its organization. In addition to this introduction, the remainder of this manuscript is organized as follows. In Chapter 2, we present the background for this thesis, *i.e.*, user steering in large-scale workflows in CSE. We present the fundamental concepts, terminology, details on how CSE users conduct their experiments using scripts or WMSs, and characterize the scientists and engineers in CSE. In Chapter 3, we show a thorough literature review on user steering support from a data analysis perspective. It is a first of this kind that investigates approaches both for WMSs and scripts in CSE. In Chapter 4, we introduce WfSteer, our theoretical framework. In Chapter 5, we present DfAdapter, one of our implementations of WfSteer. We explain its design decisions, general architecture, and implementation details on managing steering action data in scripts. In Chapter 6, we present our implementation in d-Chiron WMS, as another implementation of WfSteer. In Chapter 7, we present the experiments that support the validation of this thesis's hypothesis. Finally, in Chapter 8, we conclude this thesis by sharing lessons learned and proposals for future work.

Chapter 2

Background: User Steering in Large-scale Workflows in CSE

In this chapter, we provide the background for this thesis. We begin with a brief description on the fundamental concepts of the background (Sec. 2.1), then we present a dataflow-oriented approach that gives the basis for this thesis (Sec. 2.2), afterward, we describe CSE applications' characteristics in detail and showing real-world cases (Sec. 2.3), then we present how the background work has used data management techniques to support CSE experiments (Sec. 2.4), and finally conclude with W3C PROV standard data representation for provenance (Sec. 2.5).

2.1 Basic Concepts and Terminology

This section briefly describes the basic concepts and terminology used in this thesis. The goal of this section is to provide a glossary with informal definitions for the fundamental concepts. Formal definitions and more detailed descriptions with rich examples are presented in the following sections and chapters.

Workflow

CSE experiments are characterized by the execution of one or more CSE applications, which are software that implement complex algorithms and process large volumes of scientific data in large-scale computers [16]. These CSE applications can be modeled using a *scientific workflow* (also known as *large-scale workflow* or *workflow*, for short) abstraction [19, 49, 50]. Modeling CSE applications as workflows contributes to the overall CSE experiment because it makes the workflow data explicit, which need to be analyzed and potentially adapted. In this thesis, we use the term *workflow* as an abstract concept that means a composition of pieces interconnected through data. In traditional workflow literature, these pieces are

usually called *workflow activities*, or *activities* for short When concretized, an activity can be either a black-box program, program functions, methods, or blocks of code in a script, depending on the CSE applications being modeled as a workflow [19, 25, 35, 51].

The Two Ways CSE Users Conduct their Experiments: Workflow Scripts and WMSs

In this thesis, we consider the two typical ways that CSE users conduct their experiments on large-scale computers.

One way is by developing *parallel CSE workflow scripts*, or *workflow scripts* for short. These scripts execute a sequence of computing commands that typically invoke highly parallel libraries for data processing or performing complex computations [49, 50].

The other way is by using *WMSs*, like Chiron [20], Kepler [52], and Pegasus [21]. Differently than the previous way, where CSE users write their own scripts to control the parallelism of their applications, when using a WMS, users typically rely on it to manage the parallel execution of their applications. Cases for WMSs are executions of scientific software that process the same computation over large datasets in parallel, commonly searching for a solution in a large solution space, as a parameter sweep in an embarrassingly parallel execution exploiting data parallelism [19, 25, 35].

Workflow Data and Dataflow

When modeling the workflows, the CSE users specify the *workflow data*, which are the data processed (*i.e.*, consumed and generated) during the workflow execution, like scientific data files and parameters for the algorithms or numerical methods.

The workflow data are organized as *datasets* and the activities operate over the data to *transform* them. Thus, in a workflow execution, *input datasets* are consumed by *data transformations* that produce *output datasets*, which can be consumed by subsequent data transformations, forming a *dataflow* [25, 35] .

User Steering

User steering (also known as *computational steering*) is a concept that means that users are enabled to interactively and dynamically drive a computational process while it is still running (*i.e.*, online, at runtime, during execution).

In workflows, *workflow steering* is the ability that allows users to interactively analyze (*e.g.*, inspect, visualize, monitor) or adapt (*e.g.*, tune parameters, change input datasets) the workflow data online [23].

To steer a workflow is the action of performing workflow steering.

Steering actions are the individual user interactions performed during workflow steering, like asking a monitoring query (in case of *online data analysis*) or tuning a parameter (in case of *online data adaptation*).

When users steer, they generate *user steering action data*, which are important information that helps to understand the steering actions and their influence on the workflow data. They consist of data informing: *when* the action happened, *why* the user decided to act, *how* the action occurred, *which* workflow data were analyzed or adapted, *what* was happening before and after the action, *who* acted, and the type of the action itself.

The *track of steering actions* is a historical record containing important information that allows understanding the steering actions and their influence on the workflow data. In other words, it is the steering action data properly related to the rest of the workflow data.

To track a steering action is to query (or analyze) its track, *i.e.*, its historical record.

To *allow for tracking steering actions*, one needs to *manage steering action data*, which means *to capture the steering action data, explicitly relate them to the workflow data, and store in a database ready for online queries.*

Provenance Data

Data provenance (*i.e.*, data lineage) contains a structured record of the data derivation process [28]. It is a natural way to represent a track. In workflows, the provenance data contain not only provenance information about the data items consumed and produced, with their data derivation paths, but also data about the computational processes and agents (*e.g.*, users and software) involved in the workflow execution [29]. Provenance data helps to explain which and how processes were utilized to derive each data value, in varying granularities. Provenance data provide for data analysis, result quality assessment, data authorship, reliability, and reproducibility of the experimental results [28, 53, 54]. Such features are considered as important as the scientific achievement or the business value itself, since the CSE experiment's reliability can be compromised without provenance [53]. In this thesis, we leverage provenance data management techniques to allow for tracking steering actions.

One can further subclassify workflow provenance data into *prospective* and *retrospective provenance data* [28, 53]. Prospective provenance data provide the specification (*i.e.*, the definition, the structure) of the workflow, with its composing activities and possible data derivation paths. Retrospective provenance data provide the provenance information of the data values that have been consumed or generated during the workflow execution, together with metadata about the com-

putational processes that consumed or generated them. In summary, prospective provenance informs what should happen when the workflow executes, whereas retrospective provenance informs what happened when the workflow executed. In past work, prospective and retrospective provenance data combined have been explored to deliver insightful data analyses to scientists and users from a wide variety of scientific domains [24, 25, 35, 55–57].

2.2 A Dataflow-oriented Approach for Workflows

Data management in workflows is critical due to the inherent complexity of the scientific domain data and the HPC requirements, *e.g.*, the exploitation of data parallelism. For steering, it is essential because it makes explicit the workflow data, which need to be analyzed and steered by the CSE users. In this section, we provide the theoretical background for a "dataflow-oriented approach for workflows". It has been been proposed by SILVA *et al.* [35], influenced by IKEDA *et al.* [51] and OGASAWARA *et al.* [20], and has been refined and extended in the publications that compose this thesis. It gives the theoretical foundation for this thesis and is the basis for the formal definition of *"user steering action"*, to be defined in the following chapters.

The dataflow-oriented approach provides constructs whose essence is to promote to first-class-citizens the elements of data flowing throughout the activities, as opposed to an approach centered on the parallel execution control. Following this approach, activities are called *data transformations* that compose the *dataflow* in a workflow, so that the dataflow is the main conceptual artifact to be managed, as opposed to the parallel execution control, which is typically managed in WMSs. In this approach, the workflow data, consumed and generated by the data transformations, are organized as datasets, such as large data files or large matrices or mashes. In terms of the dataflow-oriented approach, the term *dataset* has slightly different semantics: it is a logical representation, concretized as a data view composed of metadata or scalar data values, of the physical datasets processed by the transformations. This data view contains only representative data, which can potentially improve the overall data analysis of the workflow data [35, 37]. For instance, if a certain data transformation consumes a large matrix as a physical dataset, one could extract only representative metadata about this large matrix, its shape, and a reference pointer to the physical dataset (*e.g.*, a file path to where this matrix is stored in the file system). These metadata would compose the logical dataset of the physical dataset. From now on, if not explicitly distinguished, the term dataset is used as a logical dataset, with references to the physical dataset processed by the data transformation. In this section, we precisely define these concepts.

Definition 2.1. Dataset, Data Elements, and Data Values. A dataset DS is composed of data elements, *i.e.*, $DS = \{e_1, ..., e_m\}$. Each data element e_i, $1 \leq i \leq m$, is composed of data values, *i.e.*, $e_i = \{v_1, ..., v_u\}$. Datasets are further specialized into *Input Datasets* (I_{DS}) and *Output Datasets* (O_{DS}).

Definition 2.2. Data schema and attributes. Data elements in a dataset DS have a data schema $\mathbb{S}(DS) = \{a_1, ..., a_u\}$, where each element data value v_j is associated with an attribute a_j, $1 \leq j \leq u$. Thus, a data element can also be represented as a set of ordered pairs $\{(attribute, value)\}$, s.t., $e_i = \{(a_1, v_1), ..., (a_u, v_u)\}$. Moreover, attributes have a data type (*e.g.*, numeric, textual, array).

Definition 2.3. Data Transformation. A data transformation is characterized by the consumption of one or more input datasets I_{DS} and the production of one or more output datasets O_{DS}. A data transformation is represented by DT, where $O_{DS} \leftarrow DT(I_{DS})$.

Definition 2.4. Data Derivation Path. Let DT_α and DT_β be data transformations and let $E \subset DS$ be a set of data elements produced in an output dataset DS generated by DT_α. If DT_β consumes the elements E, then DS is also an input dataset of DT_β. This defines a data derivation path between DT_α and DT_β through E, represented as $\varphi = (E, DT_\alpha, DT_\beta)$.

Definition 2.5. Dataflow. A dataflow is represented by $Df = (T, D, \Phi)$, where T is the set of data transformations participating in the dataflow, D is the set of datasets consumed or produced by the data transformations in T, and Φ is the set of data derivation paths between the data transformations in T (adapted from background work [35, 37, 51]).

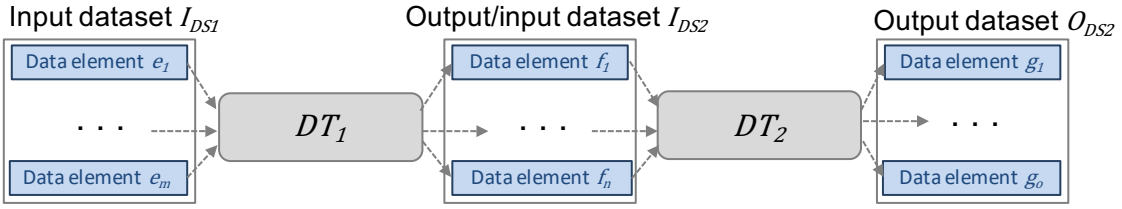


Figure 2.1: Data derivation paths between data transformations.

Figure 2.1 illustrates these basic concepts, with two chained data transformations, DT_1 and DT_2, with a data derivation path between them through data elemets of the dataset I_{DS2}, which is both an output dataset for DT_1 and an input dataset for DT_2.

Definition 2.6. Semantics of attributes. Each attribute $a_i \in \mathbb{S}(DS)$ is grouped according its semantics $\Sigma(DS)$, s.t.:

$$\Sigma(I_{DS}) = \{F_I, V_I, P_I, L_I\} \text{ and}$$
$$\Sigma(O_{DS}) = \{F_O, V_O, C_O, L_O\}$$

where:

- F_I and F_O contain attributes that represent pointers to input and output files, respectively;
- V_I and V_O contain attributes for extracted data or metadata from input and output files, respectively;
- P_I contains attributes for general purpose input parameter values of the data transformation;
- L_I contains attributes used by an iteration loop, *i.e.*, used for data transformations that evaluate a loop;
- L_O contains attributes for output values especially related to an iteration in case of data transformations that evaluate a loop; and
- C_O contains attributes for any output values that are explicit data transformation results.

Such added semantics improves the data modeling of the dataflow and allows specifying which attributes of a dataset are parameters to be steered. For example, attributes that represent parameters of data transformations, like parameters of a numerical solver, filter thresholds, ranges, have the semantics P_I and are often the main target of fine tunings.

Attributes with F_I and F_O semantics represent references to large raw (textual, imagery, matrices, binary data) scientific data files in a wide variety of formats depending on the scientific domain (*e.g.*, FITS for astronomy, SEG-Y for seismic, NetCDF for Computational Fluid Dynamics (CFD) simulations).

V_I and V_O contain representative data (often scalar values or metadata) that compose the views over the large raw data files. They facilitate users to have a big picture of the content of the files through them.

Besides large scientific data files produced by data transformations, they may produce explicit output results, represented using the C_O semantics. They are scalar values or small arrays that are meaningful for the overall result data analysis. Examples are QoIs, like errors, convergence, and accuracy;

In data transformations that evaluate loops, each iteration may be modeled as a loop evaluation execution, like $while\ i\ < MAX$. Examples of attributes with semantics L_I are loop-stop conditions (*e.g.*, MAX in the case loops if the form of $while\ i < MAX$ or *threshold* in case of loops in the form of $while\ error > threshold$). Examples of L_O are attributes that contain the current values being

used to evaluate a loop, which are updated at each iteration (*e.g.*, i and MAX). Moreover, the semantics of a dataset DS may not apply to all attributes. For example, if a data transformation does not evaluate a loop, the semantics $\Sigma(DS)$ of the datasets processed by this data transformation do not contain L_I or L_O.

2.3 CSE Applications

As briefly described in Section 2.1, we consider the two typical ways that CSE users conduct their experiments on large-scale computers: one is by executing their applications as workflow scripts and the other is by using a WMS to manage their applications. In Section 2.3.1, we provide details for workflow scripts illustrating examples of real-world applications that we use as use cases for our experiments; Section 2.3.2, for applications for WMSs; and in Section 2.3.3 we characterize the scientists and engineers involved in CSE.

2.3.1 CSE Applications in Workflow Scripts

Writing and executing CSE applications in workflow scripts is one of the two typical ways CSE users automate the execution of their CSE experiments in large-scale computers. This way is typically followed by CSE users who are experts in programming complex scientific computing tools, like numerical solvers. The notion of "script" used in this thesis means that it is a sequence of computational steps, developed by CSE users, which automates a batch execution of a CSE experiment on an HPC machine. These scripts can be written in a scripting language, like Python or Shell, or a compiled language. However, as described by SILVA *et al.* [36], the software ecosystem for developing these applications involves more than writing scripts or invoking a chain of legacy scientific codes. CSE users develop their simulation codes based on complex mathematical modeling that results in invoking components of CSE frameworks and libraries. For example, components are invoked to provide for: (i) support for partial different equation discretization methods like libMesh, FEniCS, MOOSE, GREENS, OpenFOAM; (ii) algorithms for solving numerical problems with parallel computations, like PETSc, LAPACK, SLEPc; (iii) runtime visualizations, like ParaView Catalyst, VisIt, SENSEI; (iv) parallel graph partitioning, like ParMetis, Scotch; and (v) I/O data management like ADIOS. More recently, we have witnessed the popularity boom of Machine Learning (ML) libraries, like TensorFlow and PyTorch, which also process data in parallel, to *e.g.*, train ML models and are often invoked in codes written by ML specialists. As a result, the software code is a workflow script, meaning that to run the underlying mathematical modeling, it requires invoking functions, components, or APIs from

these libraries or frameworks.

As opposed to the CSE applications that are good cases for WMSs (Sec. 2.3.2), these workflow scripts have intrinsic parallelism, not necessarily embarrassingly, usually implemented by the CSE users. Thus, WMSs, which are designed to control the parallel execution, cannot be adopted to manage the execution of these scripts because there will be conflicts caused by the competition for resources between the parallel programming libraries invoked by the script the WMS engine. Anyhow, these CSE users still need provenance data management.

Despite the several solutions available for making workflow scripts provenance-aware [58–60], capturing provenance data in these workflow scripts in CSE is still an open issue. The challenges are mainly related to performance and provenance granularity. Solutions that are easy to deploy collect provenance in very fine grain and present a significant overhead, while solutions that are based on function calls present low overhead and granularity is controlled by the code instrumentation [58]. One disadvantage of instrumentation is the need to have access to the code, which is not an issue for workflow scripts as very often the code to be instrumented is written by the CSE user, who can assist in instrumenting [36]. In Chapter 3, we discuss the support for steering in scripts. Next, we present a real case of a workflow script.

Computational Fluid Dynamics in Geoscience: libMesh-sedimentation

libMesh-sedimentation [27] is a Computational Fluid Dynamics workflow in the Geoscience domain, used in the O&G industry. It implements a simulation solver built on top of a widely used parallel finite element library, libMesh [61], which supports parallel simulation of multiscale, multiphysics applications. libMesh interfaces with several libraries for Computational Science and Engineering applications (*e.g.*, PeTSc, Metis, Parmetis, LAPACK). Also, scientific visualization tools like ParaView [62], are used to help to gain insight from the computations. The resulting application can be seen as an iterative workflow, implemented as a script. In applications like libMesh-sedimentation, users typically set up the QoIs and several parameters for the numerical methods. Examples of parameters are tolerances for linear and nonlinear solvers, number of levels for mesh adaptation, tolerances for space and time error estimates. These parameters have a direct influence on the accuracy and simulation costs, and bad choices may lead to inaccuracies and even to a simulation crash. As an example, the number of finite elements predicted by the mesh adaptation procedure may exceed the memory available in a processor, and the simulation is halted with an error message. In simulations with complex dynamics, it is often very difficult to set-up *a priori* a maximum number of finite elements per core that will guarantee the necessary accuracy without exhausting

the available resources. Figure 2.2 shows the libMesh-sedimentation solver modeled as an iterative workflow, composed of six data transformations with a dataset between each transformation. Particularly, the `Solver` part of libMesh-sedimentation is modeled as a data transformation that evaluates a time loop.

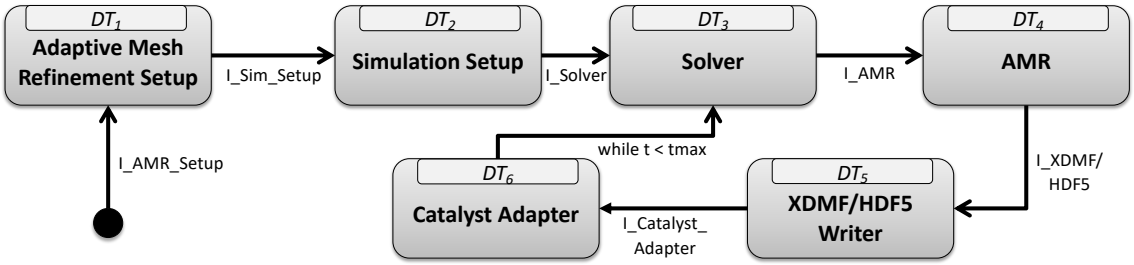


Figure 2.2: libMesh-sedimentation Workflow.

2.3.2 CSE Applications in Workflow Management Systems (WMSs)

Using a WMS is the other way for automating the execution of CSE experiments in large-scale computers. Good cases for WMSs comprise experiments that are characterized by the execution of a same program or chaining of programs to process a very large dataset. Usually, each individual execution of one of these programs is not parallel, so to speed up the processing of the datasets, one can exploit data parallelism on an HPC machine [19]. Applications with these characteristics can highly benefit from the efficient parallel execution control provided by WMSs because since there is no parallelism within workflow activity being managed by the WMS, the competition for computing resources is significantly reduced. Also, WMSs provide mechanisms that facilitate the modeling of the data transformations and the dataflow and many WMSs already provide provenance data management mechanisms. [23].

Another characteristic is that these applications are typically black-boxes, meaning that the CSE users either do not have access to their source code or having the source code access is not relevant for the overall experiment. Also, even if they did have access to the code, these users typically do not want to rewrite complex (already tested, optimized, and stable) scientific software. Thus, although the input and output datasets of the data transformations are known, the internal computation, which contains how the data are transformed, is opaque.

In Chapter 3, we discuss the support for workflow steering in existing WMSs. Next, we describe real-world CSE application examples that can be used within WMSs and how they can be modeled used our dataflow-oriented approach.

Structure Analysis: Ultra-deepwater's Risers Fatigue Analysis

The Risers Fatigue Analysis is an O&G workflow for ultra-deepwater oil production systems. Risers are fluid conduits between subsea equipment and the offshore oil floating production unit. They are susceptible to a wide variation of environmental conditions *e.g.*, sea currents, wind speed, ocean waves, temperature), which may damage their structure. The fatigue analysis workflow adopts a cumulative damage approach as part of the riser's risk assessment procedure considering a wide combination of possible conditions. The result is the estimate of riser's *fatigue life*, which is the length of time that the riser will safely operate. The Design Fatigue Factor may range from 3 to 10, meaning that the riser's fatigue life must be at least 3 to 10 (according to the factor) times higher than the service life [1].

Sensors located at the offshore platform collect external conditions and floating unit data, which are stored in multiple raw files. Offshore engineers use specialized programs (mostly simulation solvers) to consume the files, evaluate the impact of environmental loads on the risers in the near future (*e.g.*, risk of fractures), and estimate the risers' fatigue life. Figure 2.3 shows the Risers Fatigue Analysis workflow, composed of seven piped programs (represented by data transformations) with a dataset in between each transformation.

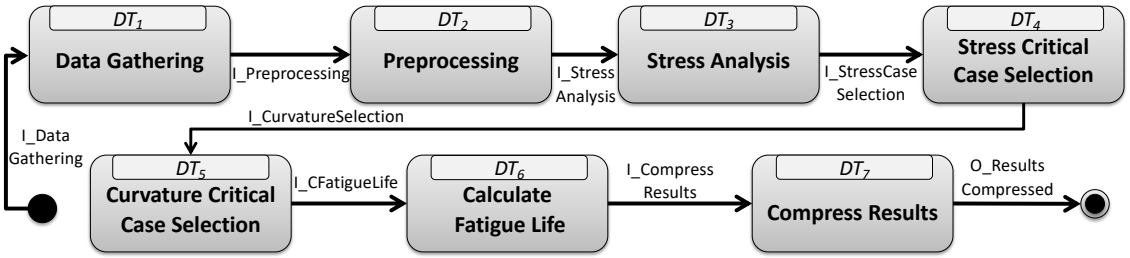


Figure 2.3: Risers Fatigue Analysis Workflow [1].

Risers Fatigue Analysis is modeled as parameter sweep workflow, where each task of `Data Gathering` (data transformation DT_1) decompresses one large file into many files containing important input data, reads the decompressed files, and gathers specific values (environmental conditions, floating unit movements, and other data), which are used by the following data transformations. `Preprocessing` (data transformation DT_2) performs pre-calculations and data cleansing over some other finite element mesh files that will be processed in the following data transformations. Stress Analysis (data transformation DT_3) runs a computational structural mechanics program to calculate the stress applied to the riser. Each task consumes pre-processed meshes and other important input data values (gathered from DT_1) and generates result data files, such as histograms of stresses applied throughout the riser (this is an output file), and stress intensity factors in the riser and principal stress tensor components. It also calculates the current curvature of the riser. Then, `Stress Critical Case Selection` (data transformation DT_4) and

`Curvature Critical Case Selection` (data transformation DT_5) calculate the fatigue life of the riser based on the stresses and curvature, respectively. These two transformations, DT_4 and DT_5, filter out results corresponding to risers that certainly are in a good state (no critical stress or curvature values were identified). Those cases are of no interest to the analysis. `Calculate Fatigue Life` (data transformation DT_6) uses previously calculated values to execute a standard methodology [1] and calculate the final fatigue life value of a riser. `Compress Results` (data transformation DT_7) compresses output files by a riser.

2.3.3 Computational Scientists and Engineers

This work aims at supporting one type of user, *i.e.*, computational scientists and engineers, who are the typical users of a user-steered workflow. However, steering workflows in HPC usually involves several users with different levels of expertise on each of the aspects involved in the process. We consider three types of users: domain specialist, computational scientist or engineer, and computer scientist or engineer.

Domain Specialists

Examples are geologists, biologists, and experimental physicists. They are very familiar with concepts, nomenclature, and semantics of the domain. They are commonly very good at understanding the scientific hypothesis, results and data interpretation. They may not have programming or computational skills. The resulting data of a complex computational simulation are often delivered to them as well organized, cured, and with some aggregations, visualizations, plots, and dashboards. Their main work is typically to give sense to these cured data.

Computational Scientists and Engineers

Examples are bioinformaticians, computational physicists, engineers, and parallel scientific application developers. They are not domain specialists, but have knowledge in the domain, although more focused on the computational aspects. They typically have programming and computational skills. They are more prone to learning new computing technologies and use new systems that support their computational simulations. They know how to analyze domain-specific data and organize large raw data files into analyzed data so they can work together with domain specialists to interpret the data. They know how to chain the different data transformations and design a workflow to attend the main goal of a CSE experiment. They can also operate WMSs or dispatch jobs in an HPC cluster.

Computer scientists and Engineers

They are experts in developing tools, methods, or systems that support large-scale simulations. Examples are HPC, data management, workflow solution specialists. Often, computer scientists or engineers work closely with computational scientists or engineers to obtain the best performance for an HPC simulation and achieve the goal of the CSE experiment. They can analyze performance, linked with domain and provenance data to help to tune the system, debugging, and fixing errors.

2.4 Supporting Workflow Steering in CSE with Data Management Techniques

There are at least six aspects of computational steering in scientific workflows: online analysis, monitoring, adaptation, notification, interface for interaction, and computing model [23]. Despite the importance of them all, the first three are essential and are the ones this thesis focuses on. Online adaptations driven by the user is at the core of user steering. However, users will only know how to fine-tune parameters or which subset needs further focus if they can explore partial result data during a long-lasting execution. The *workflow steering lifecycle* (Figure 2.4) is therefore given by: (i) analyzing the workflow data online; (ii) choosing what, when, how to adapt; (iii) adapting the workflow data online; and (iv) analyzing the workflow data again to investigate the consequences of the adaptation and returning to the first step until the end of the workflow execution.

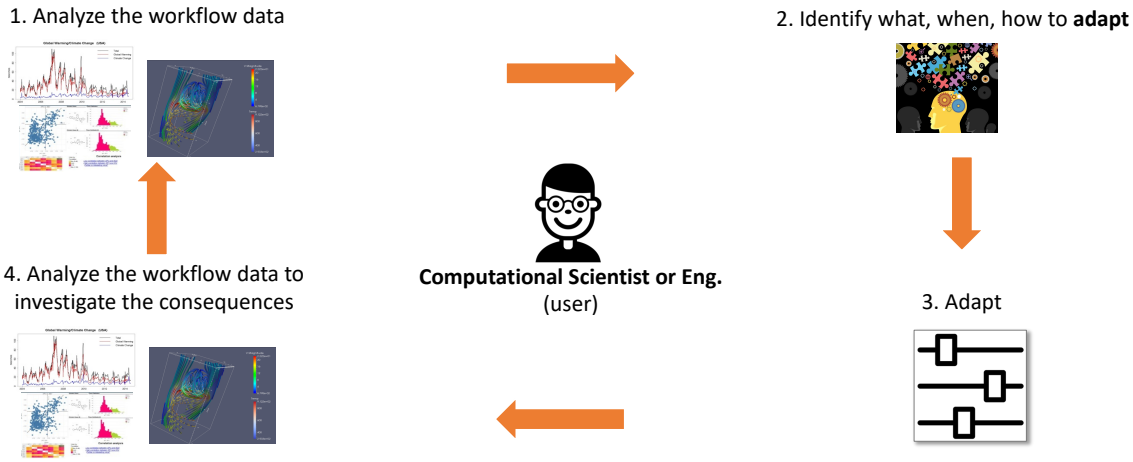


Figure 2.4: Workflow Steering Lifecycle.

In this section, we explain how users can steer the workflow based on data management techniques. A major issue to manage data in CSE is to address the

different types of data that need to be managed. We can group the workflow data into three types: execution, domain, and provenance, explained as follows.

Managing Execution Data

Tasks are the basic unit of control in a workflow execution. Lower level execution engine information, such as physical location (*i.e.*, virtual machine or cluster node) where a task is being executed, can highly benefit data analysis and debugging in large-scale HPC executions. Users may want to interactively investigate how many parallel tasks each node is running. Tasks run domain applications that can result in errors. If there are thousands of tasks in a large execution, how to determine which tasks resulted in domain application errors and what the errors were? Furthermore, performance data analysis is very useful as users are frequently interested in knowing how long tasks are taking, how much computing resources (memory, CPU, disk IO, network throughput, etc.) are being consumed. These workflow execution data are important to be analyzed and can deliver interesting insights when linked to domain dataflow data [24, 56]. When execution data is stored separately from domain and provenance data, these steering queries are not possible or require combining different tools and writing specific analysis programs [35]. To support this, a dataflow-oriented approach allows for recording parallel workflow execution data in a way that they can be linked to domain and provenance data.

During workflow execution, typically there are many tasks to be scheduled or executed. Considering our dataflow-oriented approach (Sec. 2.2), for each execution of a data transformation (also known as *data transformation execution*), there is a *task*. Although tasks are more closely related to the execution control, there are essential metadata that should be captured and coherently related to the domain data so that the user can steer the workflow. Information such as which tasks should execute on which computing nodes, number of tasks per node, tasks' input data, task duration, and how much memory or computing power it consumed are examples of execution data to be managed, which can increase the user's awareness on the CSE experiment being conducted. Execution data can be further divided into three basic categories: (i) list of tasks to be executed along with data about them, (ii) computing nodes to which tasks must be executed, and (iii) performance data [20, 24, 56, 63]. Here, we briefly describe each category:

(i) *List of tasks and their associated data.* This category refers to tasks and relevant data about them. Relevant data include: domain data to be processed by each task; status (ready for execution, running, completed); program to be executed, its version, versioning information; command lines and arguments, if

any; and any other data that can be used to improve the steering capabilities.

(ii) *Computing nodes that will execute the tasks.* This category refers to where each task will be executed, *i.e.*, which computing node will process it. There is a list of available computing nodes in the HPC machine with relevant data about them. Relevant data about a computing node include its IP address or hostname; processing capacity *e.g.*, number of instructions per second); available memory; and the total number of CPU cores. These data are useful for analyzing major execution bottlenecks associated with specific computing nodes.

(iii) *Performance data.* This category refers to the amount of resources each task consumed. Examples are the amount of CPU used in the task, memory, and network; the size of files manipulated in a specific task; and how long a task took to be completely executed. These data can be used by monitoring and debugging tools to analyze the scientific application's performance associated with the domain dataflow. Users can make decisions based on them.

Managing Domain Data

The datasets that are produced and generated in the flow between data transformations are part of the domain data that compose the dataflow. To analyze intermediate data with context, the domain data must be available for steering. Keeping track of the raw data files while keeping their context and relating their content to provenance improves online data analyses [35].

However, this is hard. The data size is large, as in a single file may achieve terabytes of data and there may be multiple files in a single run. The scientific data typically contain huge numerical values with dimensional data, which can be stored as large plain text files or binary formats in proprietary or domain-specific scientific file formats, such as FITS for astronomy, SEG-Y for seismic, NetCDF for fluid simulations, or others like HDF5. Often, the workflows expect that those files are stored at runtime on an HPC shared file system (*e.g.*, GPFS, Lustre) for data communication between data transformations. Capturing data references to these large raw data files and associating the references with relevant, meaningful data values in the dataflow is a significant assist in runtime workflow data analysis [35].

Managing Provenance Data

To manage provenance data, provenance data management systems (including components of WMSs responsible for managing provenance and provenance capture tools that can be coupled to workflow scripts) capture provenance data during workflow

execution to build a workflow database. This requires log files or other data structures. When provenance data are enriched with domain data and the database is made user-accessible for queries, the aforementioned analytical capabilities are highly enhanced [20, 35, 41, 55].

Managing execution, domain, and provenance in a same database

When execution data, domain data, and provenance data are stored in the same database at runtime, the database provides for online data analysis via monitoring, debugging, performance analysis, and interactive domain data analysis. Let us call this database as *"workflow database"*. Any database is often managed by a Database Management System (DBMS), like MySQL, MongoDB or Neo4j. Using the Risers Fatigue Analysis workflow (Sec. 2.3.2) as a real-world example, Figure 2.5 illustrates how the scientific domain data are managed in our dataflow-oriented approach, showing how the data elements flowing between transformations are stored as datasets linked with workflow execution data and provenance data in the workflow database.

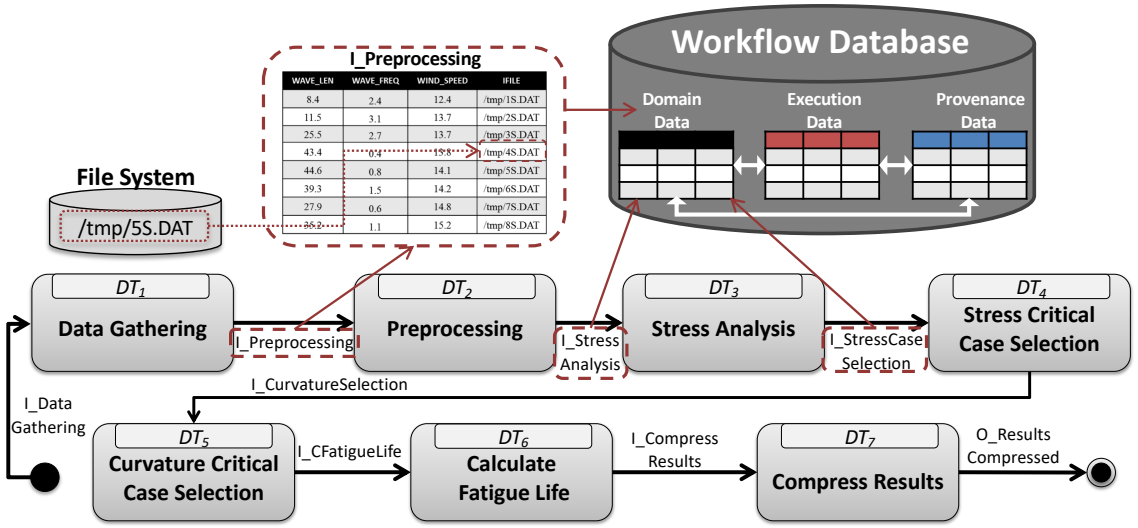


Figure 2.5: Integrating domain, execution, and provenance data in a workflow database.

Users can analyze the data by running queries in the database query interface at any time during execution or using any application that connects to the database to plot data visualization. To exemplify some possible interactive data analyses, Table 2.1 has some typical analyses that are executed for the Riser Fatigue Analysis workflow involving domain and provenance dataflow analysis. For Queries Q1–Q4, there is a history of the data elements generated in DT_4 and DT_5 since the beginning of the flow, linking each element-flow in between. For example, environmental conditions (Q1) and hull conditions (Q2) are obtained in DT_1, and stress- and

curvature-related values are obtained in DT_4 and DT_5, respectively. To correlate output elements from DT_4 or DT_5 to output elements from DT_1, provenance data relationships are required. Table 2.2 shows interactive data analysis that integrates domain, execution, and provenance data. For such, the core idea is to relate task scheduling data with domain data at runtime and make these data available for online queries.

Table 2.1: Examples of interactive data analysis integrating domain and provenance data.

Q1	What is the average of the 10 environmental conditions that are leading to the largest fatigue life value?
Q2	What are the water craft's hull conditions that are leading to risers' curvature lower than 800?
Q3	What are the top 5 raw data files that contain original data that are leading to lowest fatigue life value?
Q4	What are the histograms and related finite element mesh files when computed fatigue life based on stress analysis is lower than 60?

Table 2.2: Examples of interactive data analysis integrating domain, execution, and provenance data.

Q5	Determine the average of each environmental conditions (output of `Data Gathering` — DT_1) associated to the tasks that are taking more than the double of the average execution time of `Curvature Critical Case Selection` – DT_5), grouping the results by the machines (hostnames) where the tasks of DT_5 were executed.
Q6	Determine the finite element meshes files (output of `Preprocessing` — DT_2) associated to the tasks that are finishing with error status.
Q7	List status information about the 5 computing nodes with the greatest number of `Preprocessing` tasks that are consuming data elements that contain wind speed values greater than 70 km/h.

Without such structured query support, users need to look for files in their directories, open and analyze them, and try to do this analysis in *ad-hoc* way. Frequently, they write scripts to search in these result files. They often interrupt the execution to fine-tune input data and save execution time. This user behavior is observed not only in the O&G domain, but also in several other domains, such as bioinformatics, computational physics, and astronomy. More examples exploring how managing domain, execution, and provenance data in a same workflow database enables powerful online data analysis in workflows are found in background work [24, 29, 35, 55, 56].

2.5 W3C PROV and its Extensions

To model domain, execution, provenance data in a workflow database, it is useful to follow existing standard data diagrams to model the data entities and relationships in such a database. A major advantage in following standards is that it enables data interchange, interoperability, and ease of communication among scientists and engineers from different communities or that use different provenance data management systems that follow the same standard. The World Wide Web Consortium (W3C), acknowledged for defining standards and recommendations on the web, recommends the PROV family of documents [2]. Particularly, the Provenance Data Model (PROV-DM) is the general entity-relationship provenance data diagram that gives basis to other PROV documents, such as the PROV-O for ontologies and RDF data. PROV-DM models generic concepts like agents, entities, and activities, and how they relate to each other (Figure 2.6), which are useful concepts for the general management of provenance data.

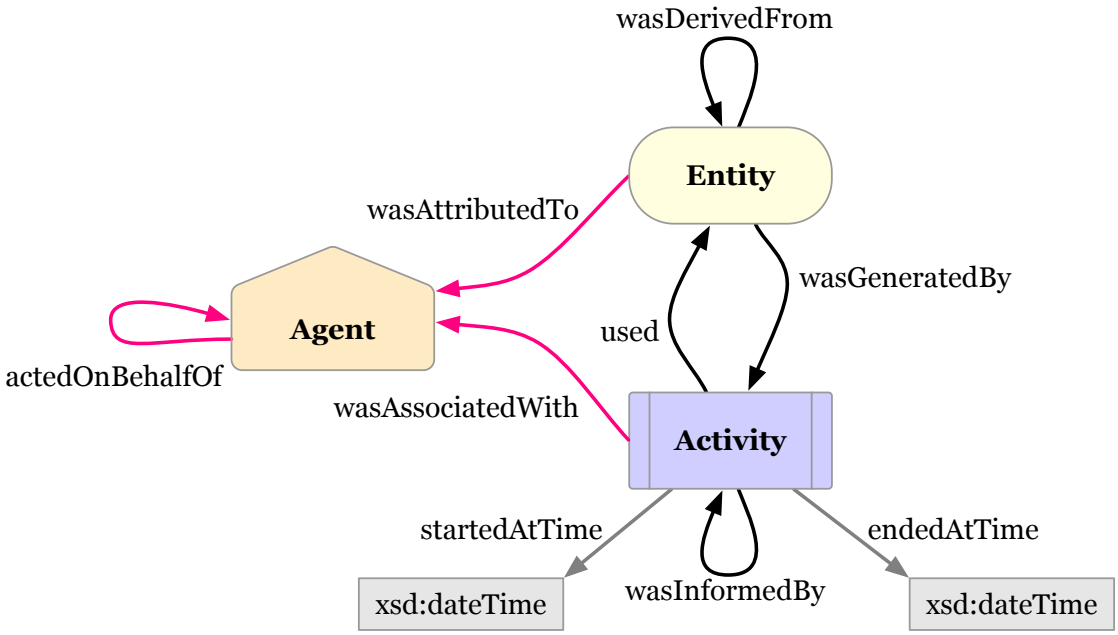


Figure 2.6: PROV-DM General Overview [2].

To model the specific concepts for workflows, there are specializations of PROV-DM that are PROV-compliant provenance data diagrams for workflows, like the ProvONE [64] and further specializations that model workflow execution data and domain data related to provenance data, like the PROV-Wf [29] and PROV-Df [35]. These provenance data entity-relationship diagrams can be implemented by systems that manage provenance data using DBMSs with various data models, varying from triple stores [65] to relational DBMSs [55].

To give a concrete example, d-Chiron [41] is a WMS that manages provenance

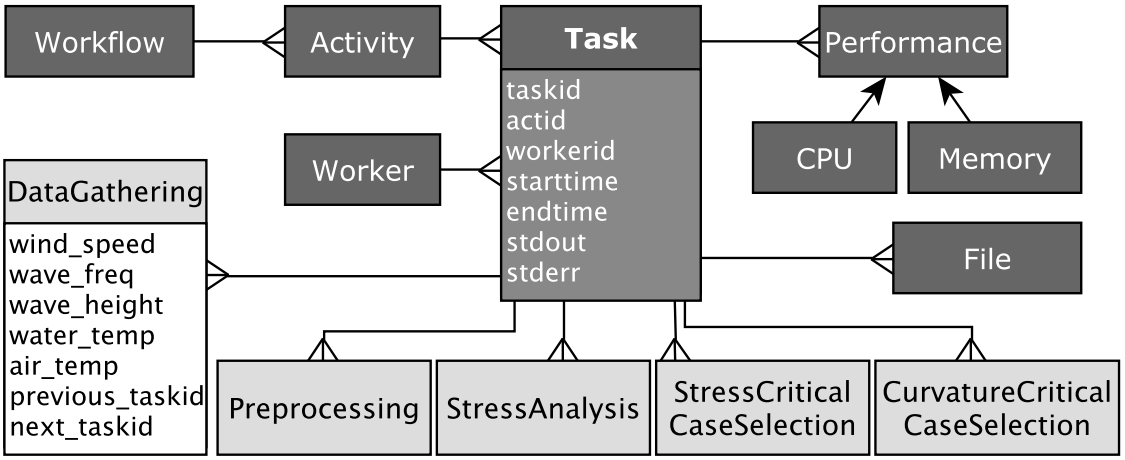


Figure 2.7: An excerpt of a relational database schema that implements PROV-Wf.

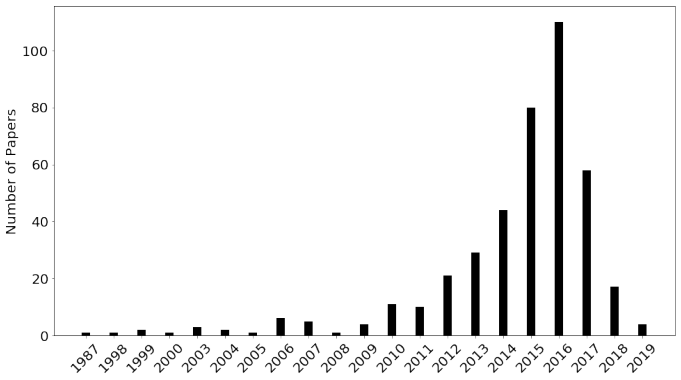
data in a relational database with an instantiation of PROV-Wf. When controlling the execution of a workflow, say the Risers Fatigue Analysis (Sec. 2.3.2), the WMS populates a database with a relational database schema whose excerpt is illustrated in Figure 2.7. Relations (tables) in light gray represent domain data and those in dark gray represent both execution and provenance data. Domain data are part of the schema used for running the workflow. A complete instantiation of the Risers Analysis workflow in a relational database schema used by d-Chiron WMS is available on GitHub, where we also show concrete examples of SQL queries used to steer this workflow [66]. The SQL queries are also in the Appendix A.

For execution data, the main supporting relation is `Task`, which has the list of tasks to be executed and their associated data. `Task` has a relationship with the `Worker` relation that keeps track of which worker (usually a computing node) executed which task. Metadata about the files that were processed or generated by each task execution are maintained in the `File` relation. `Performance` relations store data about resource consumption by each task. Since these data are related, a user can run analytical queries to help users to steer the workflow at runtime [24–26, 55, 56]. In this chapter, we presented the background concepts for our approach, WfSteer. In the next chapter, we survey how existing approaches support workflow steering.

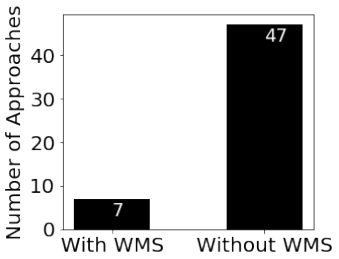
Chapter 3

The State-of-the-Art in User Steering Action Data Analysis

As discussed in the introduction (Chap. 1), steering actions are the individual interactions that the user performs while they are steering a CSE application running on a large-scale machine. Several surveys, both old [5, 67] and recent ones [50, 68], have analyzed multiple approaches to support user steering and others, also recent [19, 23, 32–34], further highlight the importance of dynamic steering in workflows. Particularly, MATTOSO *et al.* [23] analyzed approaches implemented within WMSs analyzing their steering and provenance data management support. In this chapter, we extend the analyses done by these surveys by examining several approaches, within a WMS or not, concerning how they support steering (online analysis and adaptation) and, more importantly, their support for allowing for tracking steering actions. A search on Google Scholar using keywords related to the content of this thesis returned about 450 papers, among more than 300 have been published in the past few years as shown in Figure 3.1a. Among these papers, we investigate in detail over 60 papers related to this work, as we present in the next sections. Among these papers, we identified 54 approaches, where 7 are based on a WMS and 47 are not (Fig. 3.1b). To the best of our knowledge, this is the first state-of-the-art analysis in the context of user steering in large-scale workflows that examines approaches both within and without WMSs. We begin this chapter by investigating the types of steering actions supported in the state-of-the-art. We could identify two main types: online data analysis and online data adaptation. These two can be further classified as shown in Figure 3.2, and detailed as follows.



(a) Amount of papers analyzed in our search in the past years.



(b) Number of approaches that use or not WMSs.

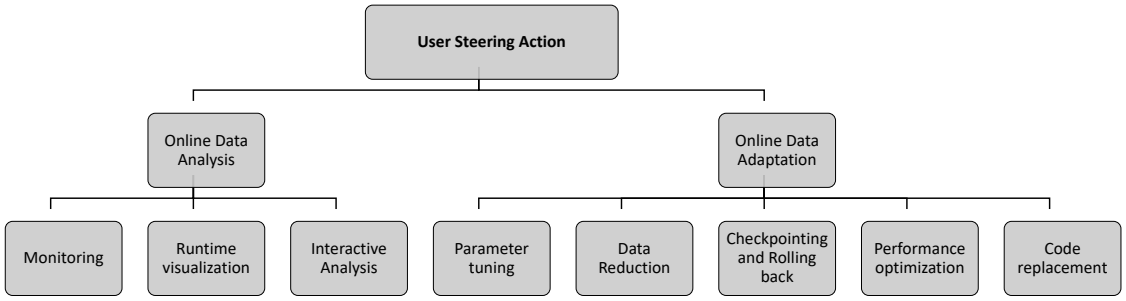


Figure 3.2: A taxonomy for user steering actions.

In case of online data analysis, we can further identify the following actions:

- *Monitoring* — a predefined data analysis plotted as charts in a dashboard or tables, whose results are refreshed in predefined time intervals. Usually, users follow the monitoring results passively, with less interaction. Depending on the monitoring results, users can interact with the workflow;
- *Runtime visualization* — a data analysis where there are scientific data visualization techniques involved, often combined with computer graphics tools and HPC techniques that allow for visualizing the large scientific data at runtime;
- *Interactive analysis* — a data analysis that allows users to perform *ad-hoc* exploratory analysis over the workflow data. This is different from monitoring in the sense that the former requires predefined specification of what is going to be analyzed, whereas the latter enables more interactive queries as the user can change what is being analyzed as they analyze the partial results.

In case of online data adaptation, we can further identify the following actions:

- *Parameter tuning* — dynamic changes of parameter values of data transformations in a dataflow, such as fine-tune simulation parameters of a computational model, change loop-stop conditions;

- *Data reduction* — dynamic removal of input data that the user decided that should not be processed, such as pruning possible solutions of a large solution space being explored in parallel that are no longer interesting (*e.g.*, the solutions are unfeasible or not optimal).
- *Checkpointing and rolling back* — specification of certain points, during workflow execution, that the user marked "safe" or had the expected results. If a failure happens, e.g., program crash, function not con-verging as excepted, or even failure caused by external factors, such as hardware failure, the user would be able to request a rollback to one of the check-pointed states. This facilitates "what-if" exploration [69].
- *Performance optimization* — dynamic tunes in the application data or computing resources at runtime aiming at improving performance or saving resources, *e.g.*, change the load of processor units that are overloaded.
- *Code replacement* — modification of parts of a running code (also known as "live programming").

Then, in the following sections, we analyze the approaches available in the literature, in the context of user steering support.

3.1 Approaches for Online Analysis, Adaptation, and User Steering Action Data Management

Table 3.1 summarizes the approaches on online data analysis support, which online adaptations they support, and whether or not they allow for tracking user steering actions. We particularly investigate, for example, if they manage steering action data or any other data for data analysis or reproducibility.

Table 3.1: Comparison of approaches for user steering support.

Approach	Online Data Analysis	Online Data Adaptation	Tracking Steering Actions
DfAnalyzer [27, 35, 38]	Monitoring, interactive data analyses, runtime visualization	X	X
VASE [70]	Monitoring	Parameter tuning, Code replacement	X

Table 3.1: Comparison of approaches for user steering support.

Approach	Online Data Analysis	Online Data Adaptation	Tracking Steering Actions
SCIRun [71]	Monitoring and runtime visualization	Parameter tuning	X
CSE [8, 10, 72]	Monitoring and runtime visualization	Parameter tuning	X
Progress and Magellan [73]	Monitoring and runtime visualization	Parameter tuning	X
CUMULVS [74]	Monitoring and runtime visualization	Parameter tuning, Check-pointing and Rolling-back	X
VIPER [75]	Monitoring and runtime visualization	Parameter tuning	X
MOSS [76]	Monitoring	Parameter tuning, Data reduction	X
gViz [77]	Monitoring and runtime visualization	Parameter tuning	X
DISCOVER [78]	Monitoring	Parameter tuning	X
MoSt [79]	Monitoring	Parameter tuning and performance optimization	X
GRASPARC [80]	Monitoring	Parameter tuning, Check-pointing and Rolling back	History Tree for maintaining the track of the simulation after a steering action, which is used to go back to a safe state.

Table 3.1: Comparison of approaches for user steering support.

Approach	Online Data Analysis	Online Data Adaptation	Tracking Steering Actions
ParaView Catalyst Live [50, 62]	Monitoring and runtime visualization	Parameter tuning	X
PathFinder [81]	Monitoring	Parameter tuning and performance optimization	X
Extempore [82]	Runtime visualization	Code replacement	X
Cactus [83]	Monitoring and runtime visualization	Parameter tuning, Data reduction	X
EPSN [84]	Runtime visualization	Parameter tuning	X
pV3 [85]	Runtime visualization	Parameter tuning	X
RealityGrid [4]	Monitoring and runtime visualization	Parameter tuning with Check-pointing and Rolling back	X
EPIC [86]	Monitoring and runtime visualization	Parameter tuning	X
CS_Lite [87]	X	Parameter tuning	X
I-WAY [88]	X	Parameter tuning	X
Falcon [11]	Interactive analysis	Parameter tuning	X
Autopilot [89]	Runtime visualization	Parameter tuning	X
WBCSim [12–14]	Monitoring and interactive analyses	Parameter tuning	X
YI *et al.* [90]	Monitoring and runtime visualization	Parameter tuning	X
MA *et al.* [91]	Runtime visualization	Data reduction via down sampling	X

Table 3.1: Comparison of approaches for user steering support.

Approach	Online Data Analysis	Online Data Adaptation	Tracking Steering Actions
HAN and BROOKE [7]	Monitoring	Parameter tuning	X
MATKOVIC *et al.* [92]	Monitoring and runtime visualization	Data reduction of parameter space exploration	X
BUTNARU [93]	Monitoring and runtime visualization	Data reduction	X
KNEZEVIC *et al.* [94]	Runtime visualization	Parameter tuning	X
DANANI and D'AMORA [6]	Monitoring and runtime visualization	Parameter tuning	X
Chiron WMS [26, 39, 40]	Monitoring and interactive data analyses	Parameter tuning, code replacement	X
WorkWays (on top of Nimrod/Kepler WMS) [52]	Monitoring and runtime visualization	Parameter tuning, Data reduction	X
gridMon Steer (on top of Triana WMS) [95]	Monitoring	Parameter tuning	X

Table 3.1 shows that all approaches surveyed provide partial support for online data analyses, as at least monitoring is provided by almost all of them and many provide runtime visualization. However, these approaches provide little support for managing steering action data, as discussed next. Figure 3.3 summarizes the contents of Table 3.1 to quantify the amount of approaches that support online data analysis, adaptation, and steering action tracking.

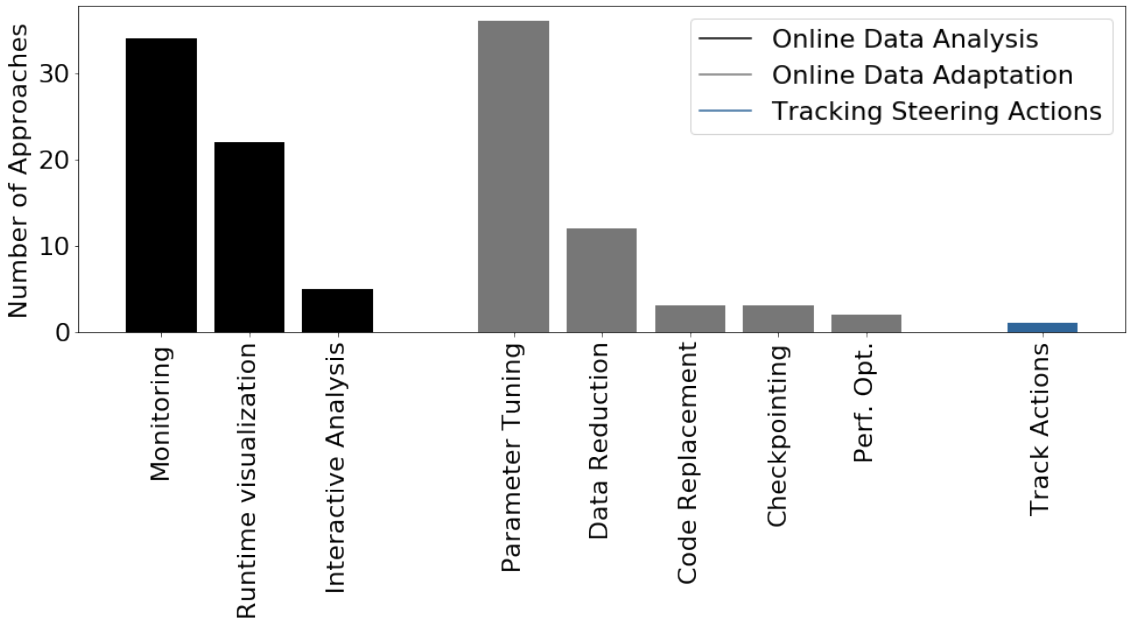


Figure 3.3: Summarization of Table 3.1.

Approaches without a Workflow Management System

GRASPARC [80] maintains a history tree, but it is used for checkpointing and rolling back. Falcon [11] collects trace data and stores in a database. It also has Trace data collector, Trace data analyzer, and a Trace data database. These traces are used for debugging and *post-hoc* analyses. Its runtime data capture is limited, with no integration with domain and performance data. Steering action data are not captured. There is no support for exploratory online data analyses. There are no explicit data relationships between the user's steering actions with the results of a steering action. Furthermore, they show little use of the traces database in their paper, making it harder to evaluate the benefits of having such a database.

DANANI and D'AMORA [6]'s work uses a DBMS for registering user records and generic application data. Despite the data analytical component in their architecture, it is not clear if it can be used for online data analyses to facilitate knowledge discovery in large amounts of scientific data and to relate to user steering data. SHU *et al.* [13, 14] use a DBMS to facilitate development of WBCSim. It stores simulation input and results, thus can be used for analyses.

DfAnalyzer [27, 35, 38] allows for rich dataflow online analysis in large-scale workflows. It contributes by providing workflow provenance data analysis enriched with execution data and domain data through runtime data extraction and indexing techniques. DfAnalyzer materializes dataflows by relating elements from raw data files. Through manual instrumentation of workflow script, users add lightweight library calls with negligible overhead to the workflow execution. It can be combined with ParaView Catalyst to allow for *in-situ* data capture and runtime visualization.

Based on data analysis provided by DfAnalyzer, CSE users were able to steer a running workflow [27]. However, like all the other approaches, DfAnalyzer does not manage user steering action data, thus users cannot understand how their actions are influencing a running workflow, which is a critical aspect that needs to be supported in the workflow steering lifecycle.

Therefore, none of these approaches manage steering action data, *i.e.*, they do not capture steering action data, relate to the workflow data, and store in a database available for analysis at runtime, which would allow users to analyze steering action data online, supporting the workflow steering lifecycle.

Approaches with a Workflow Management System

Data managed by current WMS execution engines are separate from data for provenance and analyses. During the execution, the WMS manages its internal scheduling data to be efficient for parallel performance and generates logs to be structured for queries for provenance data analyses after the execution ends. That is, execution data are not available for online queries when a workflow is running. There are important limitations with this approach of not making execution data available from data analyses at runtime.

A major shortcoming is the lack of support for monitoring queries involving domain data from data structures or repositories, which remain isolated from execution data. This prevents a WMS from providing support for monitoring queries like "how many tasks are still pending to be scheduled for the workflow activity 3?" or "what is the average execution time for the tasks that have already executed?", and debugging queries like "how long was the average execution time for tasks that produced a result value greater than x?". The problem is that the user who models a workflow often needs to fine-tune its configuration. Analytical queries, with performance data (*e.g.*, task duration, memory or CPU consumption by task, etc.) and domain data, can show whether the workflow is in the right direction or needs further adjustments. When these data are separated, submitting a monitoring or debugging query is quite complex, leading to the classic problem of data integration for analysis [96].

Besides, if performance data are not adequately registered during workflow execution, they will no longer be available for analyses when the execution finishes. Another problem is the difficulty for providing runtime data analysis since provenance is only registered in log files, which are much harder to be queried than structured or semi-structured data. To facilitate dealing with data management issues, WMS solutions often make use of DBMS. In this section, we briefly discuss how DBMS take part in existing WMS architectures and which role they play in terms of user steering capabilities.

Swift/T [97, 98] is a highly scalable approach that uses a distributed key-value store to enforce data dependencies between tasks during scheduling execution, but it does not allow for tracking user steering action. To keep its high scalability, Swift/T stores analytical data in log files, which are loaded to a DBMS only when the workflow execution finishes. Still, there are two databases, which use different DBMS to manage them and the user does not have access to the database used by Swift's engine for scheduling.

Pegasus [21] is also a scalable WMS that has been used for many real-world workflows, like LIGO for gravitational wave discovery [99]. Pegasus uses a DBMS to store execution data, which is available for the user to do runtime execution monitoring, but does not provide for data analyses based on runtime *ad-hoc* queries. Like Swift, Pegasus also stores provenance data in log files to be made available for user queries only after the workflow finishes.

Stampede [100] is a DBMS-based execution monitoring tool that can be plugged into WMSs. Stampede adopts a centralized DBMS solution and has been evaluated with two different WMSs, Pegasus and Triana, to show its monitoring facilities. However, it is also a solution that does not integrate monitoring to domain or provenance data, and does not allow for tracking steering actions.

FireWorks is a scalable WMS [101] that also has a DBMS-driven workflow execution engine. FireWorks has a JSON-based approach to state management and uses MongoDB to support scheduling queues and queries JSON documents to monitor workflow execution. FireWorks's approach shows scalable results when monitoring concurrent workflows' executions. Even though MongoDB is a distributed document-oriented DBMS, FireWorks communicates with MongoDB as a centralized database server, which suffers from many concurrent accesses. Also, as a document-oriented DBMS, MongoDB has limited query capabilities for relating different workflow data and limited *ad-hoc* analytical queries.

Chiron [20, 26] collects provenance data and uses a centralized, relational DBMS to integrate and manage data required for parallel execution and data for analytical queries. It is the only WMS that manages execution, domain, and provenance data in the same database, making it an exception among the approaches within WMSs. SciCumulus [63] is a lightweight middle on top of Chiron that adds capabilities to exploit cloud computing environments. d-Chiron [41, 102] is a version of Chiron that employs a highly efficient in-memory distributed DBMS to accelerate the parallel execution control in Chiron. However, as the other WMSs, neither Chiron or any of its versions can manage steering action data without the concepts and techniques introduced in this thesis.

Therefore, we are not aware of any WMS approach that manages steering action data or allow the users to track their steering actions, compromising the workflow

steering lifecycle support.

Other Approaches for User Steering

Other approaches [103–105] provide some user steering support, like parameter tuning, reduction of parameter space, performance tuning, and monitoring. However, they provide little or none data analytical support, no provenance data management, and no steering action data management.

Moreover, while most approaches that enable data reduction aim at avoiding data to be generated, *e.g.*, by enabling users to prune useless solutions in a large solution space, there is an intense area of data reduction research focused on reducing data already generated by the simulation. For example, initiatives like CODAR [106], Melissa [107], DimStiller [108], and the work by JIN *et al.* [109], propose data reduction strategies such as dimension reduction, outlier detection and compression also based on online data analyses, which are complementary to our approach. Other approaches provide steering support but are too application or domain-specific, such as Computational Fluid Dynamics [110], whereas we aim to design a domain and application-independent solution.

SPINUSO [111] has investigated the importance of capturing provenance data at runtime for steering in scientific workflows, in an approach called *Active*. Among several contributions, he proposes S-PROV, a provenance data diagram extension of W3C PROV, to represent workflow abstractions and concretization in the execution of real use cases. The main difference between *Active* and our approach *WfSteer* is that we aim at supporting computational steering in large-scale workflows, such as the ones found in CSE applications that require large HPC machines, which adds new challenges as already discussed. Additionally, another difference is that we addressing the challenge to enable efficient capture and query of provenance data considering use cases both with and without WMSs, which adds a new complexity to the problem. Therefore, we believe our detailed characterization of steering actions, along with the notion of provenance of the steering action data and our proposed W3C PROV-compliant representation of these concepts, and the design principles for efficient steering action data capture in HPC considering both with and WMSs is complimentary to the *Active* approach.

Recent works [112–115] have explored user steering concepts for parameter tuning; data capture for analysis, monitoring, and debugging; and data reduction applied to Machine Learning life cycle (*e.g.*, data preprocessing, training, models, result analyses).

Figure 3.4 [3] shows a conceptual interactive optimization approach, such as human-guided search, for Operations Research. Because an optimization model "may underestimate or ignore some aspects of the real problem", resulting in com-

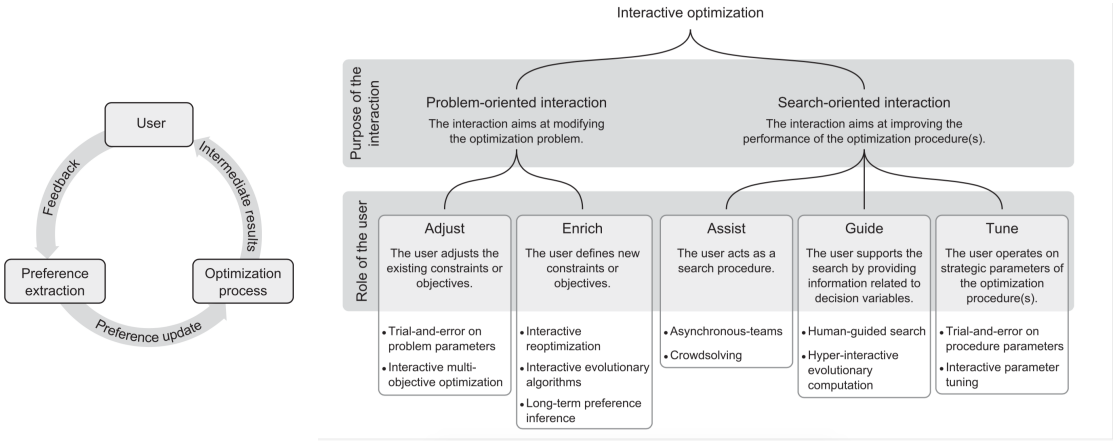


Figure 3.4: User steering approach for optimization [3].

putation of either "unrealistic or unfeasible solutions, or solutions that do not capture domain-related characteristics", several contributions in Operations Research have been made to overcome this. For example, robust optimization to deal with the inherent uncertainty of a decision context. However, even with such improvements in Operations Research, there are still limitations such as to find an efficient optimization method or, regardless the optimization model previously designed, there may be specific decision criteria that remain unknown and will only be found during the decision-making process. For this, interactive optimization can be used and they share the same motivations as ours. However, most of the approaches that provide some support for interactive optimization do not use an online approach, requiring the simulation to stop, tune, and re-submit. Also, making sense of very large raw datasets is not a focus.

ADIOS [116], GLEAN, Damaris/Viz, Qiso, Nessie, Strawman, Decaf [117] enable strong monitoring and online data visualization features, but no adaptation, nor steering action tracking. WMSs like WINGS/Pegasus [21], Swfit [97, 98], Nimrod/Kepler [118], VisTrails [119] have monitoring and data analyses support, but no adaptation nor steering action tracking. In the next section, we analyze these WMSs in detail regarding their data analysis and provenance management support.

3.2 Making a CSE Application Steerable

In Table 3.2, we summarize how each analyzed approach makes a CSE application steerable and Figure 3.5 shows the amounts of approach per category. Here, we want to analyze how they expose the data to be steered by the user. For instance, if they use message passing-based or data-oriented implementation for communication between the running parallel application and the user who is steering. The impor-

tance of this in the context of this thesis is that to be able to capture steering action data, our solution needs to be aware of how the CSE application is made steerable.

Table 3.2: Comparison of implementation on how the approaches make a CSE application steerable.

Approach	Making Application Steerable	Communication Implementation
DfAnalyzer [27, 35, 38]	Manual source code instrumentation.	HTTP RESTful API calls.
VASE [70]	Manual source code instrumentation.	File-based communication.
SCIRun [71]	User writes C++ abstract program using the system's libraries	MPI
CSE [8, 10, 72]	User writes "satellites" and plug to their application to communicate with the "Data Manager"	MPI
Progress and Magellan [73]	Manual source code instrumentation, user specifies only relevant data structures (called "steering object model") to be exposed for steering.	Separate threads in the same memory space
CUMULVS [74]	Manual source code instrumentation	NI
VIPER [75]	Manual source code instrumentation.	NI
MOSS [76]	Manual source code instrumentation	NI
gViz [77]	Manual source code instrumentation	NI
DISCOVER [78]	Manual source code instrumentation	MPI between the running code and the steering server; and Java RMI between the server and the UI.
MoSt [79]	Dynamic source code instrumentation	NI
GRASPARC [80]	Manual source code instrumentation	NI
ParaView Catalyst Live [50, 62]	Manual source code instrumentation	File-based
PathFinder [81]	Manual source code instrumentation	NI

Table 3.2: Comparison of implementation on how the approaches make a CSE application steerable.

Approach	Making Application Steerable	Communication Implementation
Extempore [82]	User writes in a specific programming language.	Just-in-Time compilation for "hot-swapping" or "live programming" of a main loop.
Cactus [83]	Manual source code instrumentation	NI
EPSN [84]	Manual source code instrumentation	NI
pV3 [85]	Manual source code instrumentation	NI
RealityGrid [4]	Manual source code instrumentation	SOAP for messages between client and server.
EPIC [86]	Manual source code instrumentation	NI
CS_Lite [87]	Manual source code instrumentation	Socket-based
I-WAY [88]	Manual source code instrumentation	NI
Falcon [11]	Manual source code instrumentation	NI
Autopilot [89]	Manual source code instrumentation	NI
WBCSim [12–14]	Manual source code instrumentation	NI
YI *et al.* [90]	Manual source code instrumentation	File-based
MA *et al.* [91]	Manual source code instrumentation	File-based
HAN and BROOKE [7]	Manual source code instrumentation	NI
MATKOVIC *et al.* [92]	Users develop in a monolithic system	NI
BUTNARU [93]	Both non-intrusive and manual source code instrumentation	NI
KNEZEVIC *et al.* [94]	Manual source code instrumentation	MPI

Table 3.2: Comparison of implementation on how the approaches make a CSE application steerable.

Approach	Making Application Steerable	Communication Implementation
DANANI and D'AMORA [6]	Modification of RealityGrid source code to use the UI inside a HPC machine network	MPI
Chiron WMS [26, 39, 40]	Dataflow-oriented WMS; Non-intrusive	MPI and DBMS-oriented
WorkWays (on top of Nimrod/Kepler WMS) [52]	Science Gateway; Non-intrusive	HTTP RESTful API calls.
gridMon Steer (on top of Triana WMS) [95]	Non-intrusive.	NI

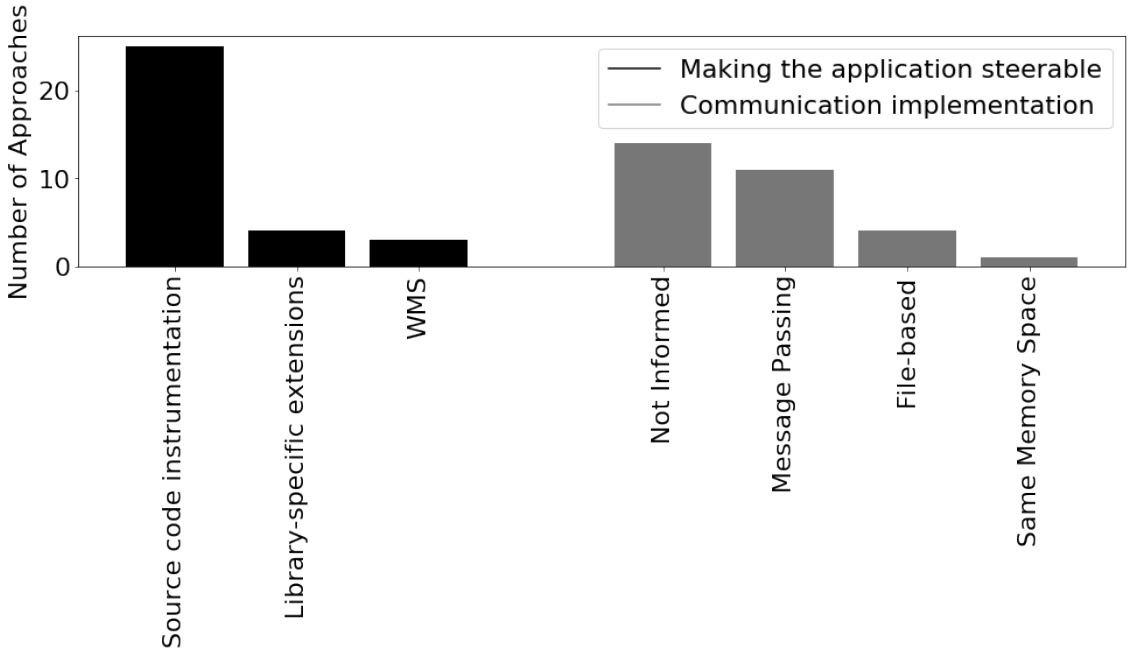


Figure 3.5: Summarization of Table 3.2.

Users can develop their own *ad-hoc* way to implement an adaptable application,

such as using a simple file-based approach [27]. Such implementations are usually lightweight but unsuitable for reuse, as argued by BAUER *et al.* [50]. General-purpose systems or frameworks or libraries that provide user Steering support are more suitable for a universal reuse [50]. We investigate how such approaches can modify a running code. Approaches that ease data parallelism control, like WMSs with steering support [25, 26, 120], provide non-intrusive strategies to support parameter sweep workflows, but cannot support applications that run as workflow scripts, like the ones we describe in Section 2.3.1. Other approaches and libraries enable users to manually instrument their source code for external invocations to a steering library [5, 50, 68]. Although "manually" may be an issue, in a way that there are solutions that aim to enable autonomous ways to instrument a source code [60], manual instrumentation is a typical routine in the daily life of CSE application developers [36, 50]. Users that develop such applications are used to writing external calls to parallel libraries in their application source code. They know very well where to add such calls in their code so they know which aspects should be steered during execution. Manual instrumentation allows users to customize the solution to their specialized needs, as argued by GU *et al.* [11], who developed steering support in Falcon.

Solutions that provide autonomous code instrumentation to capture data at runtime may pollute the source code and the online data analysis as they may collect more data than the data that are actual relevant for the users, and they can add unnecessary overheads by collecting data that the user will likely never use [36, 58]. These approaches that support steering by allowing the user to manually instrument their application in strategic "steerable" points of their application code are suitable for CSE applications, which is a limitation for the WMSs that support steering. However, if the added code for instrumentation is overwhelmingly complex for the user, too many lines need to be added, or significant performance overheads are added, this may be a problem for adoption of the solution, as discussed by the authors of CS_Lite [87].

3.3 System Design of an Approach that Supports User Steering

By surveying these approaches, we identify two main strategies to implement steering: data-oriented or message passing. In the data-oriented strategy, there is a shared data space between the running parallel application and the system supporting steering. The majority of these approaches using the data-oriented strategy adopt a file-based implementation, *i.e.*, files in the file system are the "shared data

space". The file is accessible both by the user steering the running application and by the application itself. If the user changes the file, the application reloads its settings based on the new values in the file. The advantages of this implementation are that it is very simple and lightweight, and it does not require extra software or libraries [4]. However, if there are multiple checks in this file in a very short time, or if there are multiple modifications at once, or if a high performance shared disk (*e.g.*, Lustre, GPFS) is not available, this may impact the overall performance of the simulation. In addition to performance, another issue that may arise is consistency. If a user modifies an application while it is running, how to guarantee that the adaptation will not cause an error or lead to an inconsistent state? For instance, during execution often there are thousands of tasks to run in parallel. Some of them are already running, others have already executed, and others are waiting to be executed. If the user tries to steer the execution affecting an already executed task, the steering action may not take effect; or if the user tries to modify a task that is executing, like performing calculations or writing data files on disk, this may lead the execution and data (*e.g.*, overwriting files that were being written) to an inconsistent state. Moreover, when designing a steerable application, one needs to be concerned with a "steering lag" [121], which is the elapsed time between the user interaction and the steering action to take place in the running parallel application.

In addition to data-oriented, other implementations use a client-server architecture, via Message Passing Interface (MPI), sockets or HyperText Transfer Protocol (HTTP) for message passing between the user interface, the running application, and the system providing steering support. We explain which implementation each approach uses in Table 3.2. In these implementations, there are no disk accesses for steering, which may be a better strategy for *in-situ* steering. However, we found no performance and consistency issues being further addressed in those works. As mentioned by AYACHIT *et al.* [68], there was no focused effort to study the scalability of these approaches nor the overhead they add to parallel workflow script (*i.e.*, the simulation code). Besides, some approaches [67, 73] use the terms "sensors" and "actuators" to signify runtime relevant data capturers and adapters of steerable data.

3.4 Further Discussion on the Analyzed Approaches

We analyzed several approaches and their support for steering. The publications of three of them, RealtyGrid [4], Magellan [73], and Mulder's [5], contain high-level illustrations (Figures 3.6, 3.7, and 3.8, respectively) of an approach that provides traditional user steering support. In common, they have three main components: the user interface (from where users analyze and steer the parallel application); the

system that supports steering; and the parallel application that is going to be steered by the user.

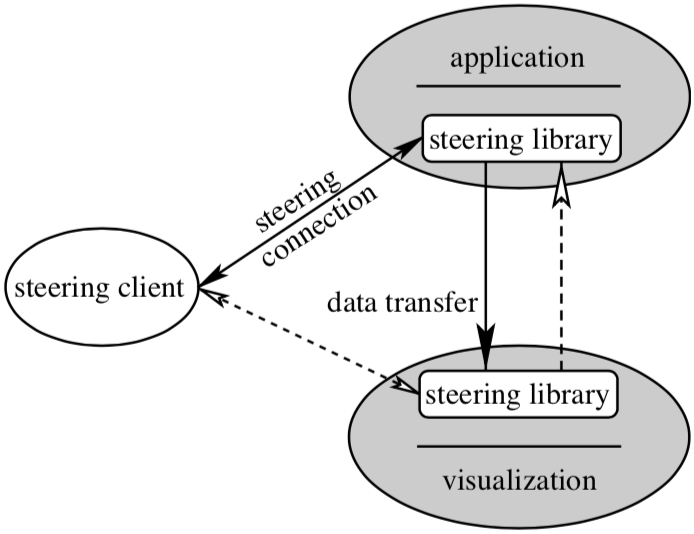


Figure 3.6: Basic components of an approach that supports user steering, according to PICKLES *et al.* [4].

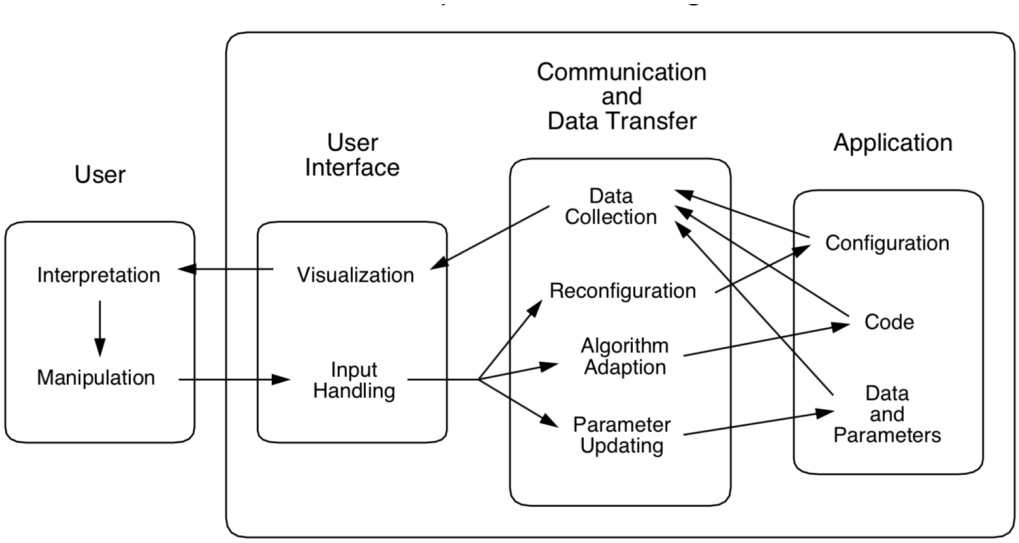


Figure 3.7: User steering concepts, according to MULDER *et al.* [5].

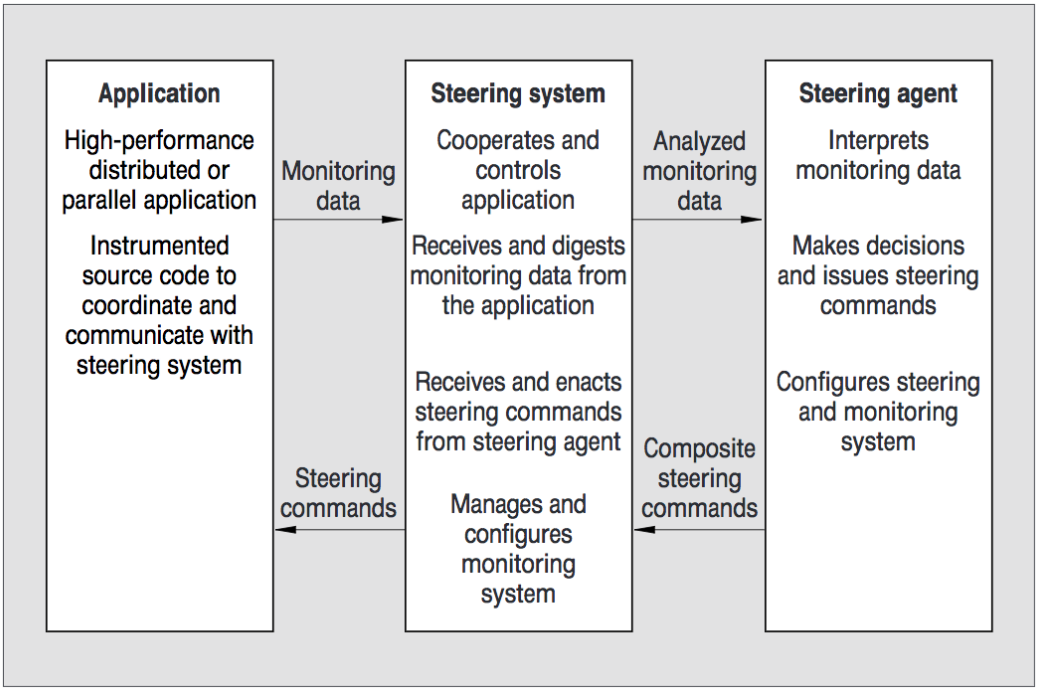


Figure 3.8: Conceptual view of a approach that supports steering according to MULDER *et al.* [5].

Figure 3.9 shows a general architecture for interactive optimization, which considers human feedback for building a preference model to help users to steer [3].

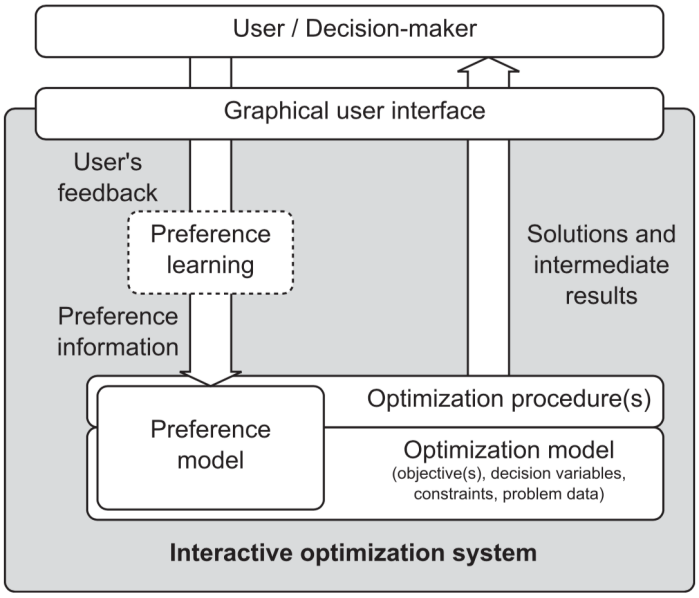


Figure 3.9: User steering for Operations Research [3].

In Figure 3.6 [4], we also see that they provide a library to be used as external calls in the running code for visualization and steering. According to DANANI and

D'AMORA [6], who deployed RealityGrid in Blue Gene/Q HPC machine, RealityGrid has a client-side that runs on the user's desktop whereas the server runs on the same machine that the parallel simulation runs. This is a limitation for Blue Gene/Q and several other HPC machines, like Lobo Carneiro[1], which disallows connections form external networks for security reasons. Thus, they modified RealityGrid's open source code [122], in the client component, to run it on Blue Gene/Q. Then, they were able to run RealityGrid in OpenFOAM, a highly parallel CFD toolbox that contains numerical solvers. They were able to control two parameters using RealityGrid: *deltaT* and *nCorr*, which are used in the inner loop of the solver. They provide a general architecture (Figure 3.10) for computational steering and data analyses for large volumes of raw data, which they call "Future of Steering Workflows".

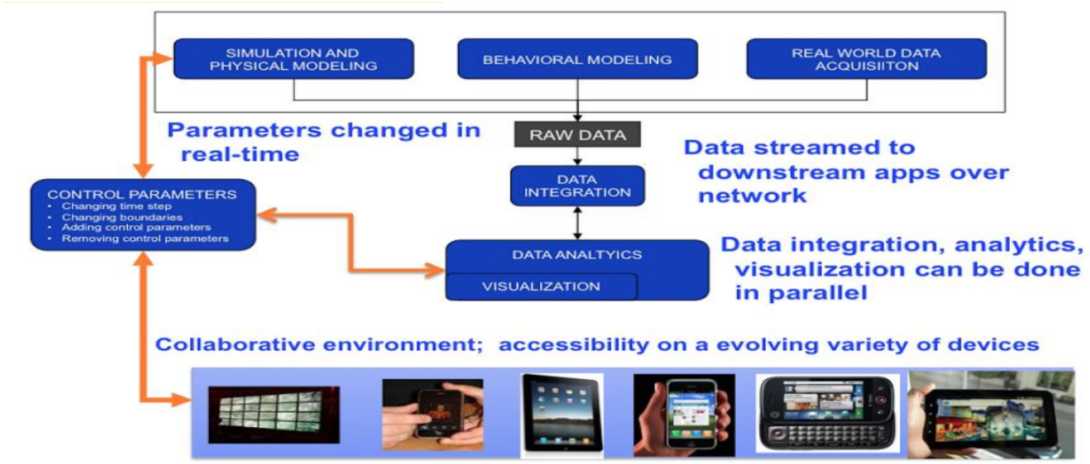


Figure 3.10: DANANI and D'AMORA [6]' view on "the future of steering workflows".

Figure 3.11, extracted from HAN and BROOKE [7]'s work, shows a dynamic data-driven architecture for steering. There is a Calibrator component that is controlled by humans and can change the parameters and variables at runtime of a legacy source code previously instrumented manually. Data are captured at runtime for Machine Learning-based prediction and monitoring. They use a DBMS to store data for the Machine Learning models. Based on analysis of the Predictor and Monitoring, users can steer. HAN and BROOKE are co-authors of another application [123] that runs on Blue Gene/Q. This application has a similar architecture to the one that uses RealityGrid, which we previously discussed. This application demonstrates an interesting modern use of this Computational Steering: users using a mobile device, such as a smartphone, to steer a CFD simulation running on the IBM Blue Gene/Q HPC machine.

[1]http://www.nacad.ufrj.br/recursos/sgiicex

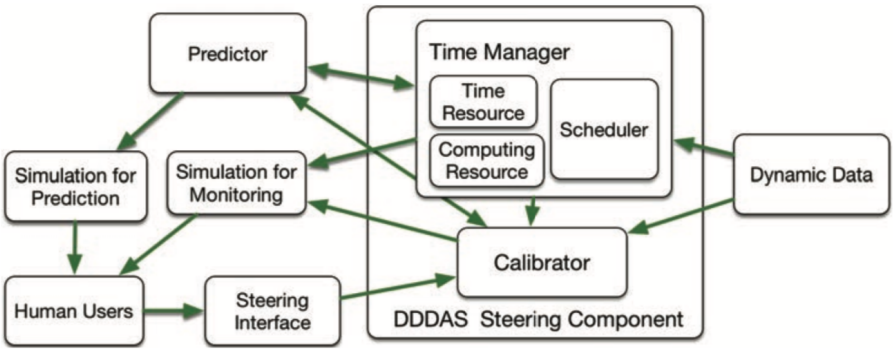


Figure 3.11: Dynamic data-driven steering architecture [7].

Another classic approach that supports steering is the so-called "Computational Steering Environment", developed in the nineties [8–10], whose architecture is shown in Figure 3.12. It is one of the first approaches with user steering support. It also relies on the idea of manual application source code instrumentation. It uses a data-driven approach for steering. "Satellites" are interfaces between the user and the Data Manager. Satellites can read and write variables (scalars or arrays) from a database managed by a Data Manager component. When a variable is changed as a result of a satellite command, the Data Manager sends back a notification to confirm the change. The database managed by the Data Manager is stored as files on disk. In this way, users of parallel applications develop their mechanisms for reading/writing variables to this file.

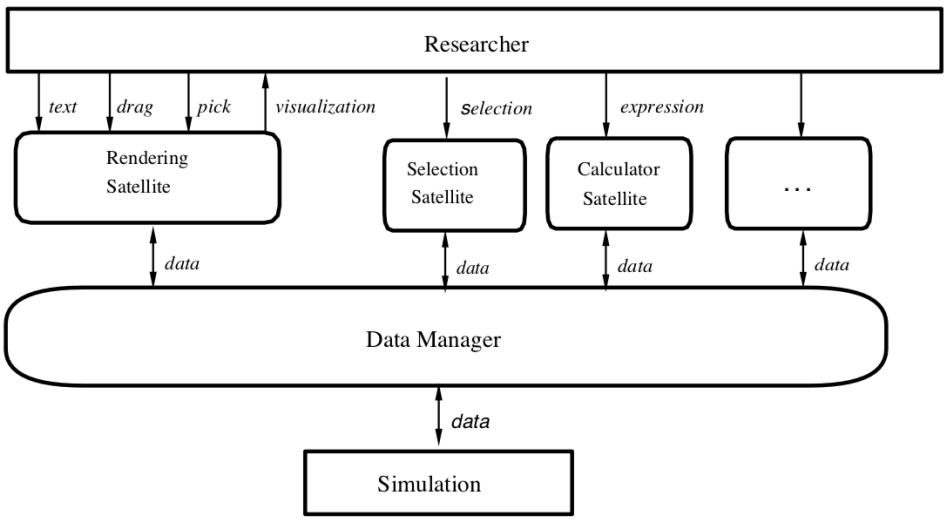


Figure 3.12: "Computational Steering Environment" system architecture [8–10].

Falcon [11] is a lightweight toolkit that supports monitoring and steering. Its architecture is shown in Figure 3.13. It has runtime libraries for information cap-

ture, collection, filtering, and analysis, and a graphical user interface. In Falcon, the application code is instrumented. The user specifies monitoring to expose relevant simulation parameters to be monitored and steered. Falcon also captures performance data. Partially processed monitoring information is used by the user to help to decide on what to steer. One of the main functionalities in Falcon is to allow users to control at runtime any added functionality to the user's running application. Users can dynamically turn on and off any runtime data capture or steering capabilities previously added during source code instrumentation.

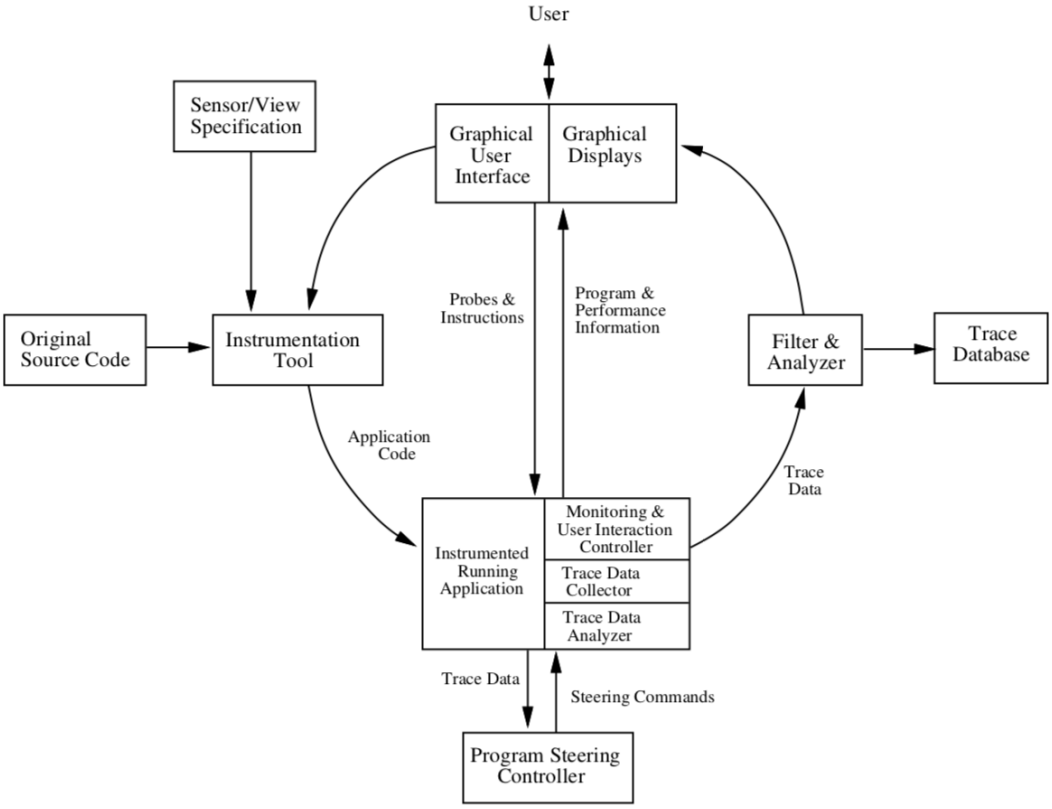


Figure 3.13: Falcon's architecture [11].

WBCSim (Figure 3.14) has been developed since 1999 [12]. It is another approach that supports user steering in legacy workflow script codes. It provides a lightweight library for source code instrumentation. It has a monitor that alerts users when it is time to steer. It has a data visualization of simulation results. Analyzing the behavior of the execution after a steering action, the addition of multiple steering points, and the addition of support of a database in an idea that is similar to the track of steering action data to compare steering results to other results are pointed as future work [13]. Despite the important contributions for general concepts of user steering, WBCSim's papers [12–14] promote deeper specialized discussions on how to support wood science and legacy FORTRAN application codes instead of a wider variety of scientific domains. Moreover, one of its design goals is to provide

user steering for small-scale Problem Solving Environments, which differs from our motivations as we aim at large-scale parallel applications.

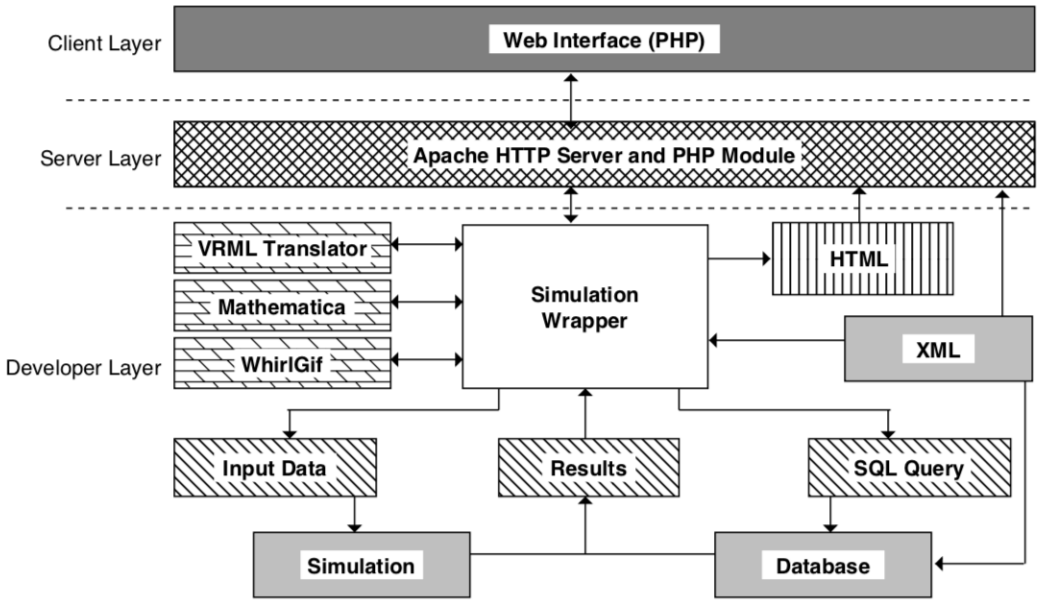


Figure 3.14: WBCSim approach architecture [12–14].

VIPER (VIsualization of Parallel numerical simulation algorithms for Extended Research) [75] is mainly designed for parameter space exploration. Users annotate the program to identify data and input parameters as steerable objects. When a steerable object is found in the code at runtime, VIPER server is notified. It extracts data from the application or restores input parameters. They test steering in a parallel CFD application.

MoSt (The Monitoring and Steering Environment) [79] provides powerful data visualization and enables users to adapt the running code. It also allows for steering performance optimization. One interesting feature is its ability to instrument a running application. This allows dynamicity even without pre-defined code instrumentation.

Extempore [82] is a live-programming environment. Users add calls to Extempore in their main simulation loop. Then the loop is controlled by Extempore. Thereafter, the user can interactively send statements to the Extempore compiler. They are "just-in-time" compiled (C, C++, Fortran) into native machine code, including MPI, and immediately executed.

ParaView Catalyst Live [62] is designed to run synchronously with the simulation. There is MPI communication between simulation and analysis and it is the application developer's responsibility to write an additional communication routine during code instrumentation. It has efficient *in-situ* data visualization capabilities as analysis and simulation run alongside, in the same address space. ParaView Cat-

alyst Live enables monitoring and steering the simulation workflow. The simulation data structures are re-mapped at each time step [50].

Therefore, as surveyed by BAUER *et al.* [50], many of these approaches are no longer maintained. As discussed by FIGUEIRA and BUI [87], some of them are very hard to be used, as there are several lines of code that need to be added in the user's application code. HART and KRAEMER [121] mention that consistency in modifying a running parallel code needs to be considered by the steering systems developers, but we found that most of these approaches do not address consistency problems, as mentioned in Table 3.2. Thus, although many approaches provide steering support and some of them were proposed over twenty years ago, there are issues such as performance, consistency, and enabling multiple steering actions that are not yet addressed. Furthermore, none of these approaches allows for tracking steering actions, compromising the workflow steering lifecycle support. Except for DfAnalyzer and the WMS approaches, provenance data management techniques, which could be a solution for this, are not mentioned at all in the publications describing the approaches surveyed in this chapter. In the next chapter, we present our approach for tracking steering actions in workflows.

Chapter 4

WfSteer: An Approach for Managing User Steering Action Data

We informally defined steering actions as interactions that the user performs during workflow steering, like online data analysis and online data adaptation, in Section 2.1 and presented formal definitions on the dataflow-oriented approach in Section 2.2. Here in this chapter, we introduce WfSteer, our approach for managing user steering action data in large-scale workflows. We begin by motivating the need for management of a new type of data, steering action data, to support steering in CSE (Sec 4.1). Then, we extend the dataflow-oriented approach to formally define *user steering action* and *steering action data* (Sec. 4.2) and define instances of steering actions following these definitions (Sec. 4.3). Finally, in Section 4.4 we introduce the notion of provenance of steering actions and present a new W3C PROV-compliant data diagram to model it. The conceptual parts that encompass the two typical ways of conducting CSE experiments on large-scale computers (*i.e.*, workflow scripts and WMSs) have been unified here in this chapter. Then, in Chapters 5 and 6, we introduce the specific details on how WfSteer is instantiated to manage steering action data in workflow scripts and in a WMS, respectively.

4.1 User Steering Action Data: a new type of data that needs to be managed

After users have analyzed partial data and gained insights, they may decide to adapt the workflow execution online. In this thesis, an *online data adaptation*, or adaptation for short, is a steering action performed by a user that causes a change in the flowing data elements in the dataflow. Adaptations bring powerful abilities to users, putting the human in control of a scientific workflow execution. They can significantly reduce overall execution time, since users are able to identify a

satisfactory result before the programmed number of iterations. The adaptable aspects range from computing resources involved in the execution (*e.g.*, adding or removing nodes), to check-pointing and rolling-back, reducing datasets, tuning of filter thresholds, loop-stop conditions, and parameter values.

Populating a workflow database (Sec. 2.4) during workflow execution enables both online data analysis and adaptation. For example, it is possible to analyze or monitor the data being continuously generated as the workflow executes [35], change filter conditions during execution [124], and adapt loop conditions of iterative workflows (*e.g.*, modify number of iterations or loop stop conditions) [26].

In a long-running execution, many interactive data analysis and adaptations may occur. When a user adapts the dataflow, new data type of data, *steering action data*, are generated and, hence, must be managed, *i.e.*, captured, related and stored in a database, integrated with the rest of the workflow data (*i.e.*, execution, domain, and provenance). Since provenance data are so beneficial, we consider that when a user interacts with the workflow execution, the provenance of the generated steering action data must be stored in a database as well. By bot doing so, users can easily lose track of what and how they have steered in the past, negatively impacting results reliability, validation, and reproducibility. For example, if a user removes subsets of a dataset (data reduction-kind of adaptation), the tasks (execution data) that would consume them will not need to be executed, leading to inconsistencies or spending unnecessary time if the tasks are executed [25]. In another example, if a user fine-tunes parameters of a program, the overall result may be changed.

Furthermore, enriching the workflow database steering action data, jointly with provenance, execution, and domain data, enables future interaction analysis. In addition to reliability and reproducibility, having such data enables users to learn from their own adaptations: they may find that when they tune certain parameters to a given range of values, the convergence of the solver improves by a certain amount. Finally, these steering action data allow for building Artificial Intelligence (AI)-based systems that help users while they are steering simulations [125], as they can extend their training database with provenance of adaptations. Thus, the computational steering systems, including WMSs with steering support, should manage steering action data, which has never been done before.

4.2 User Steering Action and Data Definitions

As defined in Definition 2.5, a dataflow Df is represented as $Df = (T, D, \Phi)$. Given this, we define User Steering Action as follows.

Definition 4.1. User Steering Action. A user steering action SA is an interaction between a user who analyzes or monitors or dynamically adapts one or more

elements of DS:

$$DS' \leftarrow SA_{\alpha}(DS)$$

where $DS \in D$ is a dataset in the dataflow that the user can steer, α is a steering action clause that delimits the scope of the steering action resulting in DS'.

For instance, in a parameter sweep workflow steered by a user, an input dataset I_{DS} can have its composing data elements dynamically adapted by the user. In this case, α contains the criteria to specify which elements would be affected by the action, resulting in the dynamically modified dataset I'_{DS}. This example is explored in the next section.

To allow for tracking a user steering action SA, steering action data need to be managed, *i.e.*, captured, related to the rest of the workflow data, and stored in a database ready for online data analysis. User steering data consist of data informing: *when* the action happened, *why* the user decided to act, *how* the action occurred, *which* workflow data were analyzed or adapted, *what* was happening before and after the action, *who* acted, and the type of the action itself (*e.g.*, parameter tuning, data reduction, query).

Definition 4.2. User Steering Action Data. User Steering Action Data SD are important information that helps to understand the steering actions and their influence on the workflow data. User Steering Action Data SD are represented as:

$$SD = (DS, DS', \alpha, U, \Gamma, \mu, \psi)$$

where:

- DS, DS' and α are the datasets and the clause, respectively, involved in the action SA;
- U contains data about the user who performed SA;
- Γ is a set of data transformation executions (tasks) related to SA;
- μ contains metadata of the steering action SA itself, such as the type of the action, wall time during which the action lasted, descriptions informed by the user at the moment of the action; and
- ψ is an optional argument representing any other data, in the same context of the steering action SA, which can benefit the register of the steering action.

4.3 Definitions for Online Data Adaptations

Adaptations have a characteristic of either update (we call them *U-adaptation*), or insert or delete (called *I/D-adaptation*). U-adaptations are steering actions that update data values, like adjustments, fine-tunings, or any modification of one or more

data elements in the dataflow. Parameter tuning is an example of U-adaptations. I/D-adaptations are steering actions that cause addition or deletion of data elements in the dataflow. Data reduction is an example of I/D adaptations. In this section, we define these two examples of adaptations as special cases of the Steering Action general Definition 4.1.

Definition 4.3. Tune. *Tune* is a steering action for tuning parameters, represented as follows:

$$I'_{DS} \leftarrow Tune_{(\eta,C)}(I_{DS})$$

where:

- I_{DS} contains old values of attributes being tuned into I'_{DS} with the new values. I'_{DS} follows the same schema $\mathbb{S}(I_{DS})$ and semantics $\Sigma(I_{DS})$ of the input dataset I_{DS};
- (η, C) is the steering action clause;
- η is a set of ordered pairs (p, v), where $p \in P_I \subset \Sigma(I_{DS})$ is the parameter being tuned and v is its new value; and
- C expresses a predicate to address a specific data element that is having its parameters tuned. In case of an I_{DS} that contains a single data element, C is optional.

To register a *Tune* action, the steering action data SD_{Tune} managed are:

$$SD_{Tune} = (I_{DS}, I'_{DS}, (\eta, C), U, \Gamma, \mu, \psi)$$

Definition 4.4. Cut. *Cut* is a steering action for data reduction. It cuts data elements of an input dataset I_{DS}. It is represented as follows:

$$I'_{DS} \leftarrow Cut_C(I_{DS})$$

where:

- I_{DS} is the input dataset that is having its data elements being removed by the cut, resulting into I'_{DS}, s.t. $|I'_{DS}| < |I_{DS}|$, which is the new dataset after the cut. I'_{DS} follows the same schema $\mathbb{S}(I_{DS})$ and semantics $\Sigma(I_{DS})$ of the input dataset I_{DS} ;
- C is the criteria that addresses the slice of input data elements that are removed. C may either be a simple predicate (*e.g.*, $a_1 = FATIGUE$) or a min-term predicate (*e.g.*, $a_2 > 38 \wedge a_3 > 0.1$).

To register a *Cut* action, the steering action data SD_{Cut} managed are:

$$SD_{Cut} = (I_{DS}, I'_{DS}, C, U, \Gamma, \mu, \psi).$$

4.4 Modeling Steering Action Data using W3C PROV Concepts

In Section 2.5, we discussed W3C PROV and its specializations for workflows, like PROV-Wf [29] and ProvONE [64]. SILVA *et al.* [35] have modeled the dataflow concepts presented in Section 2.2 using another PROV specialization, called PROV-Df. In this thesis, instead of creating a completely new provenance model, we first begin by consolidating a base model using several past contributions to the W3C-compliant provenance data diagrams PROV-Wf and PROV-Df [25, 29, 35, 55] to introduce a W3C PROV-compliant provenance data diagram that specializes PROV concepts to model user steering actions. We call it *PROV-DfA*. By adhering to the well-established W3C PROV standard, our approach aims at allowing for interoperability among other provenance databases. In addition, another important principle is that the W3C PROV and its extensions PROV-Wf and PROV-Df are abstract and flexible enough to be used in different domains or applications, such as Astronomy, Bioinformatics, and O&G [25, 29, 35, 126]. A general overview of PROV-DfA using an UML class diagram is illustrated in Figure 4.1. For the sake of comprehension of the figure, we only show the classes and their relationships. The relevant classes' attributes are not shown in the figure, but discussed in the text.

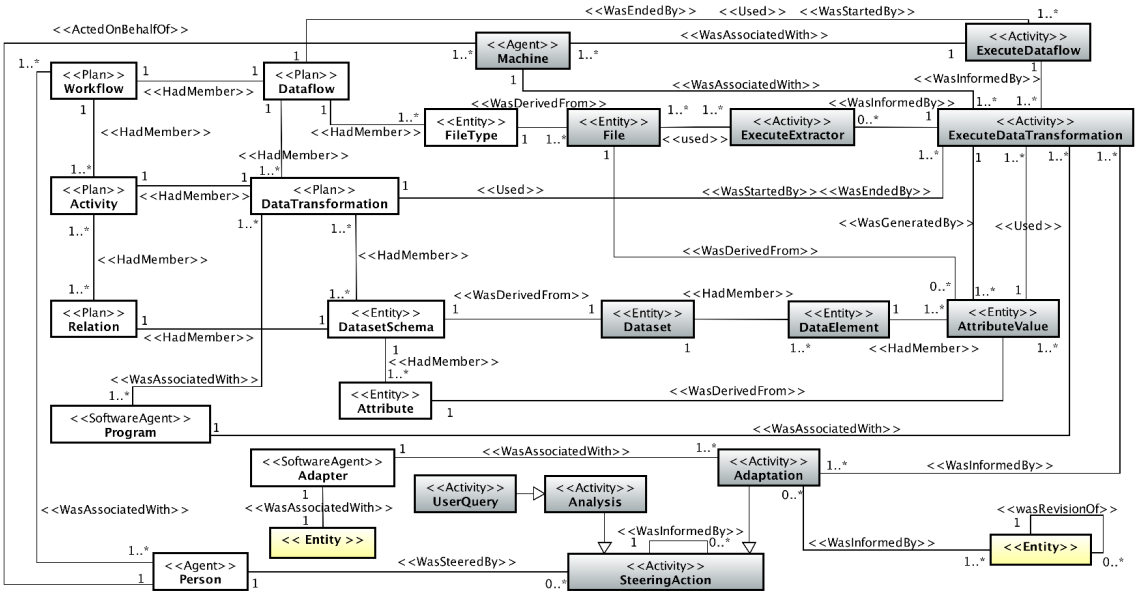


Figure 4.1: PROV-DfA overview. A larger visualization is on GitHub [15].

PROV-DfA's General Concepts

We use `prov:` namespace to indicate the PROV classes or relationships. Each `ExecuteDataTransformation` consumes (`prov:used`) and produces (`prov:wasGeneratedBy`) `AttributeValues`. These values may have been extracted

by an `ExecuteExtractor` [35]. Data elements compose the dataset (`Dataset`). For prospective provenance, the dataset has an associated `DatasetSchema`, which is composed of `Attribute`. `Attributes` describe the `AttributeValues` generated during execution. They have a data type (*e.g.*, integer, text) and may have extra fields in the `Attribute` class to allow for attribute specification (*e.g.*, determine if the attribute semantics is in $\{F_I, F_O, V_I, V_O, P_I, L_I, L_O, C_O\}$ — Def. 2.6). Data about execution, such as task's (*i.e.*, data transformation execution) wall time and performance data (CPU, memory), linkage to subsequent and previous tasks, and their related prospective and retrospective provenance can be stored relating to instances of `ExecuteDataTransformation`.

Next, PROV-DfA adds specialized classes and relationships to represent steering actions and steering action data. PROV-DfA introduces the classes `SteeringAction`, `Analysis`, `Adaptation`, and `Adapter`; and the relationship `WasSteeredBy`. We use a UML class diagram, where the `«stereotypes»` in classes specify PROV superclasses (mainly `Agents`, `Entities`, and `Activities`); and the `«stereotypes»` between classes specify relationships. Classes in white background represent prospective provenance, whereas in gray represent retrospective provenance. `prov:Entity` in yellow means that classes in PROV-DfA that are subclasses of `prov:Entity` (prospective or retrospective) can be used in place, as explained next.

In PROV-DfA, adaptation is represented by `Adaptation`, a subclass of `Steering Action`, subclass of `prov:Activity`. An adaptation was steered by (`wasSteeredBy`) a `prov:Person`, occurred at a specified time (class attribute `prov:startedAtTime`), had an adaptation characteristic (class attribute `adaptationCharacteristic`) that can be "update" or "insert/delete", in case of U-adaptations or I/D-adaptations, respectively. Users may add a description to the adaptation to describe, *e.g.*, what was going on in the experiment when they decided to perform a specific change. Also, as inherited by `SteeringAction`, an `Adaptation` may have been informed by another `Adaptation`, hence the auto-relationship `prov:wasInformedBy`. This is the case, for example, of a rollback adaptation, requested by a user, that happened right after the user modified parameters in a simulation, which is another adaptation.

Since adaptations in the dataflow occur while the workflow is executing, it is important to manage the execution state data. The most representative PROV-DfA activity that represents the execution state is `ExecuteDataTransformation`. When an adaption occurs, these instances carry information about time, pointers to domain data being consumed or produced, and computational resources being consumed. Thus, being able to track which specific data transformation was running at the moment of the adaptation may be very useful for analyses that integrates adaptation with provenance, domain, and execution data. For this, we relate which

ExecuteDataTransformation instances were influenced (prov:wasInformedBy) by adaptations. How adaptations relate to ExecuteDataTransformation, as well as how prov:Entities are affected depend on characteristic of the online adaptation, as explained next.

Adapter is a software service (it can be implemented, *e.g.*, as a function in a workflow script or a method in the WMSs code) that knows the communication protocol capable of adapting the running workflow following the command issued by the user. That is, it knows how to adapt the elements of the dataflow in a running workflow. Since it is a software service, it is a subclass of prov:SoftwareAgent. When the user decides to adapt an element of the dataflow, the Adapter is responsible for modifying the requested element. Any information that describes the Adapter (*e.g.*, which element of the dataflow it adapts, where the program can be located in the file system, how it can be invoked) may be stored relating to the Adapter class. Adapter relates to classes that are subclasses of prov:Entity and to the adaptation itself (via prov:wasAssociatedWith).

When the user performs a U-adaptation, a new instance of Adaptation is created. Also, a new instance of one of the prov:Entity subclasses in PROV-Df is created (*e.g.*, AttributeValue, DataTransformation) containing the new data, which will replace the old data in the dataflow. The newly created entity is related (prov:wasInformedBy) to the adaptation. Moreover, the newly created data is related to the old one via prov:wasRevisionOf, so that the association between the new and old data is maintained. Additionally, to relate the adaptation with execution state, PROV-DfA relates (prov:wasInformedBy) the ExecuteDataTransformation instances that were in "running" state at the moment of the adaptation. Finally, Adapter is related to the prospective entity (*e.g.*, Attribute, DataTransformation) that specifies the entity adapted.

When the user performs an I/D-adaptation, a new instance of Adaptation is created and there is a relationship (prov:wasInformedBy) between the Adaptation and the added or deleted instances of a prov:Entity subclass. In case of deletions, the entity is not physically deleted from the workflow database, for the sake of provenance. Rather, it is assumed that when an Adaptation is a deletion, the deleted instance is logically deleted from the dataflow. This enables tracking entities deleted online. Since adding or deleting elements affects the execution, the instances of ExecuteDataTransformation directly affected by the added or deleted elements of the dataflow are related (prov:wasInformedBy) to the Adaptation instance. For example, in a data reduction, data transformations that are supposed to execute are not executed because of a adaptation. These not executed instances of ExecuteDataTransformation are related to the Adaptation instance. Finally, Adapter and Adaptation are related similarly as in U-adaptations.

In summary, in PROV-DfA, an `Adaptation` is a `prov:Activity` steered by a `prov:Person`, which influenced instances of classes that are subclasses of `prov:Entity`, and influenced instances of `ExecuteDataTransformation`. The `Adapter` software relates to the prospective entity being adapted and to the adaptation.

Modeling Specific Steering Actions using PROV-DfA

In this section, we specialize PROV-DfA concepts to represent online parameter tuning, changes in loop control, and data reduction as PROV-DfA's *U* and *I/D-adaptations*.

Parameter Tuning

Parameter tuning refers to the action of steering parameters of a data transformation in a dataflow, like numerical solver parameters or machine learning model hyperparameters. In PROV-DfA, ParameterTuning is a specialization of Adaptation. Parameter tunings are adaptations in attribute values (AttributeValue) that are related to data elements (`DataElement`) related to I_{DS} (`Dataset`) of a certain data transformation (`DataTransformation`). The attribute value modified must have been derived from (`prov:wasDerivedFrom`) an `Attribute` whose attribute specification is P_I. As a U-adaptation, a new instance of `ParameterTuning` is created and related to the new instance of its adapted entity, *i.e.*, `AttributeValue`, with the new value for the parameter. The new value is related to the old one via `prov:wasRevisionOf`. `ExecuteDataTransformation` instances running at the moment of the adaptation are related to the `Adaptation` instance. Finally, since users tune parameters of data transformations, the `Adapter` relates to the `DataTransformation` associated to `DatasetSchema` that had the `Attribute` modified.

Online Adaptation of Iterative Simulations

Workflows with an iterative workflow execution model have data transformations that evaluate loops. Using the dataflow-oriented approach concepts, values for these loop-stop conditions are modeled as an attribute in L_I of a data transformation that evaluates a loop and the iteration counter is modeled as an attribute in L_O of the data transformation. Moreover, each iteration generates an instance in `ExecuteDataTransformation` for the loop evaluation. During execution of each iteration, a relationship between the output of this data transformation, containing the current iteration value, and the `ExecuteDataTransformation` instance is particularly useful for such workflows, as it identifies a specific part of the workflow

execution, and often users can analyze results as the workflow iterates. Such control information is important for the adaptation, as users can associate their specific actions with execution data, such as which point in workflow elapsed time that action happened or what memory or CPU consumption were.

In PROV-DfA, such adaptations are represented as `LoopAdaptation`, a subclass of `Adaptation`. Similarly to `ParameterTuning`, its instance is related to the new instance of `AtrributeValue`, containing the new value for the loop control condition, relating (`prov:wasRevisionOf`) to the old one. The adapted instance of `AttributeValue` must be derived from an `Attribute` whose attribute semantics is L_I. Additionally, the generated `ExecuteDataTransformation` instance related to the output of the last iteration (*i.e.*, last execution of the data transformation for loop evaluation) is related to the `LoopAdaptation` instance. Finally, the adapter must be able to dynamically modify the data transformation that represents the loop evaluation. That is, `Adapter` in this case relates to `DataTransformation`.

Data Reduction

Online user-steered data reduction is very useful for reducing execution time and amount of data to be processed during a simulation [25]. `DataReduction` is a subclass of `Adaptation`. In the dataflow-oriented approach, data files are represented as pointers in F_I, whereas V_I contain extracted domain values from those files specified in F_I. An approach to reduce data is to specify a criteria based on V_I values to eliminate files in F_I to be processed, enabling the adapter program to logically delete data elements in I_{DS}. This makes the application not to execute the data transformations for the removed elements. Analogously, in PROV-DfA, reducing data means to logically remove instances of `DataElement` (and consequently `AttributeValues`) of a `Dataset` (I_{DS}). This can be the result of an I/D adaptation. Thus, there is a relationship (`prov:wasInformedBy`) between the removed instances of `DataElement` and `AttributeValue` and the adaptation. The `ExecuteDataTransformation` instances that would use (`prov:used`) the removed `AttributeValue` instances are related to the `DataReduction` instance. Additionally, the criteria to remove data elements is stored within the adaptation instance. Finally, as users remove data elements in I_{DS}, the adapter is related to the `DataTransformation` associated to the `Dataset` that had the `DataElement` and `AttributeValues` removed.

Applying the Dataflow-oriented Approach with PROV-DfA

Now we apply the concepts defined in Sections 2.2 and 4.2 to put the PROV-DfA provenance data diagram into practice to illustrate with a concrete example of its use in the libMesh-sedimentation workflow.

In Figure 4.2, we show large raw input files (with mesh data) stored on disk, with pointers in the solver's input dataset I_{DS}. I_{DS} has over 70 parameters, among which only two are displayed in the figure (flow linear and non-linear tolerance). These solver parameters are extracted from a configurations file, which is read at each iteration. Yet, the maximum number of iterations (t_{max}) is a L_I attribute of the data transformation solver. Metadata are extracted (V_I) from input raw files at runtime to allow for tracking their contents while they are processed. Elements of O_{DS} of each data transformation are also collected (via raw data extractors and source code instrumentation) and stored in the database. For example, the solver O_{DS} contains calculated values, such as linear and non-linear results, as well as the current time iteration value. In addition to online data analyses, the user performs several adaptations during the simulation.

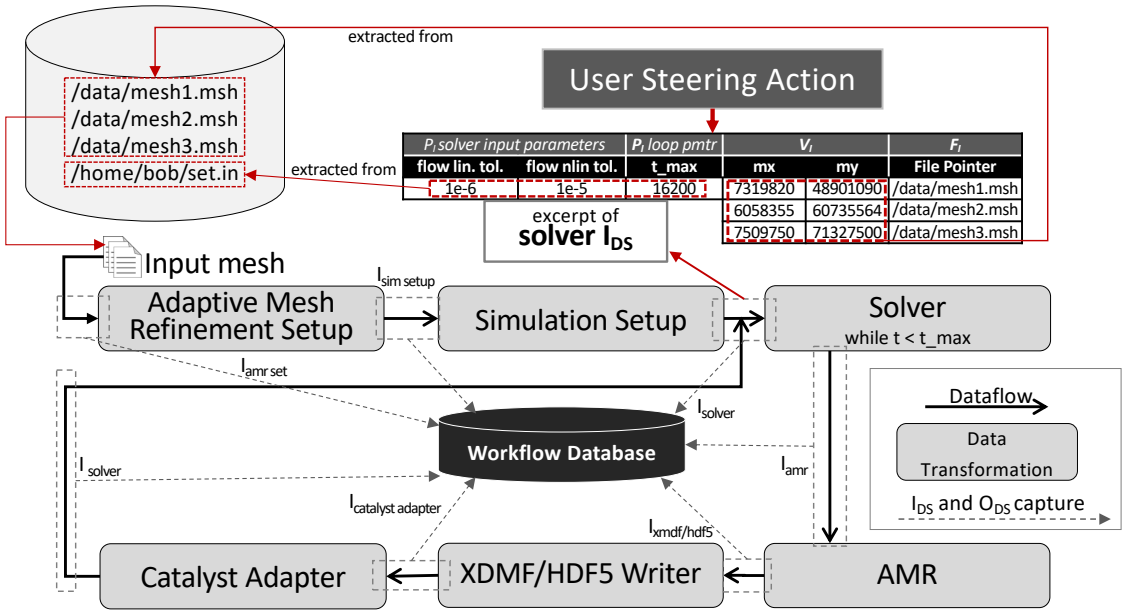


Figure 4.2: Dataflow in the libMesh-sedmentation simulation using the dataflow-oriented approach with PROV-DfA.

In Figure 4.3, we present a visualization of an excerpt of the data in a workflow database implementing PROV-DfA. It shows a user tuning the flow linear tolerance parameter from 1e−5 to 1e−3 and a data reduction with criteria $mx < 7e6$.

By using data in a relational workflow database implementing PROV-DfA, users can run the following queries (their SQL codes are on GitHub [15]).

- *Inspecting parameter tunings ("who", "when", "what")*

  How many tunings did the user do? Which parameters did the user change? What were the values when the user changed and what values did the user change into? When did each adaptation happen?

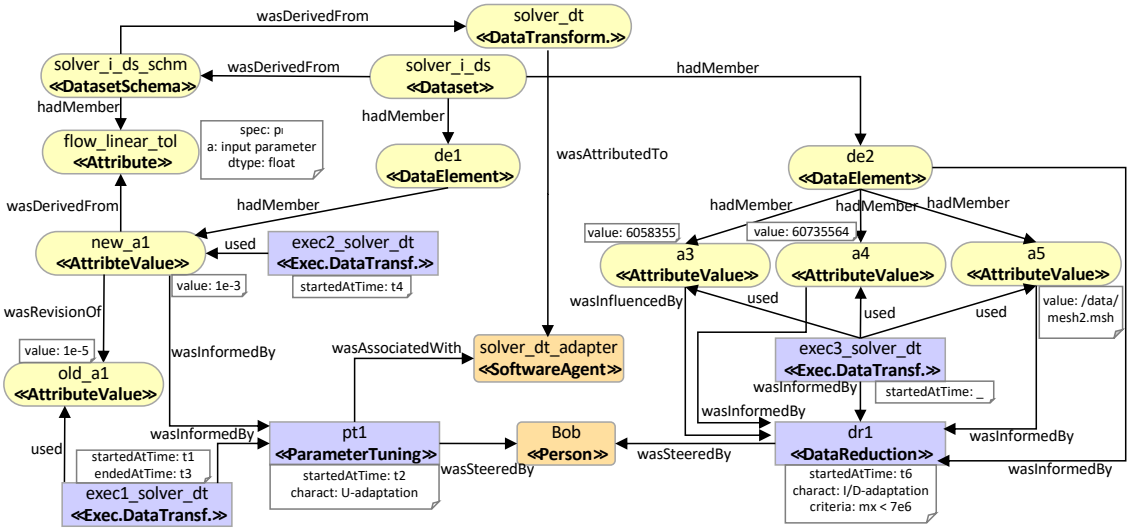


Figure 4.3: Visualization of data using PROV-DfA.

- *Inspecting parameter tunings ("how")*

  In parameter tuning 3, how was the main solver output values 10 iterations before and after?

- *Data reduction ("how", "which")*

  On average, how long iterations were lasting before and after the user reduced input files from the input data? Which files were affected?

These queries show the potential of PROV-DfA for workflow databases allowing for tracking user steering action data in large-scale workflows.

4.5 Adaptive Monitoring Concepts

In this section, we present an adaptive monitoring approach that combines monitoring and adaptation. It helps users following the evolution of the workflow data being generated during execution, interesting parameters, QoIs, and result data. Since what users find interesting may change over time, this approach allows the user to adapt the monitoring definitions, such as which data should be monitored and how. The adaptive monitoring relies on online queries to the continuously populated workflow database. Users can set up queries (such as the ones in Tables 2.1 and 2.2) to monitor the data, analyze monitoring results, and adapt monitoring settings.

Monitoring works as follows. There is a query set QS composed of monitoring queries q_i, $0 \leq i \leq |QS|$, each one to be executed at each $\Delta t_i > 0$ time intervals. Users do not need to specify queries at the beginning of the execution, since they do not know everything they want to monitor. This is why QS starts empty.

After users gain insights from the data, after interactive data analyses, they can add monitoring queries to QS in an *ad-hoc* manner. Each Δt_i can be adapted, meaning that users have control of the time frame of each q_i during execution. The monitoring queries and settings are stored in the workflow database.

Each q_i execution generates a monitoring query result set qr_{it}, $t = \{k\Delta t_i | k \in \mathbb{N}_{\geq 0}\}$, at each time interval Δt_i. This result set is also stored in the workflow database. The users have the flexibility to adapt the monitoring during workflow execution. To do so, at each time instant t after each monitoring query result qr_{it} has been generated, the values for Δt_i and q_i are reloaded from the workflow database. If any change has happened, it will be considered in the next iteration $t + \Delta t_i$. Moreover, at each certain time during execution (also configured by the user), there is a check to verify if the user has added new monitoring queries in QS. This approach takes advantage of the data stored online in the workflow database to enable users to adapt monitoring settings, including which data will be monitored and how.

In the next chapter, we present the first instantiation of WfSteer to support workflow scripts.

Chapter 5

Managing User Steering Action Data in Workflow Scripts

The first instantiation of WfSteer, called DfAdapter, aims at allowing for tracking steering actions with low execution overhead in large-scale workflow scripts. DfAdapter is a lightweight tool whose goal is to capture provenance of online steering actions in dataflows and storing the related dataflow provenance to enable understanding of the impacts of the actions. Section 5.1 presents DfAdapter's methodology and general conceptual architecture; Section 5.2 has the main system design principles followed by DfAdapter; Section 5.3 has implementation details; Section 5.4 shows how DfAdapter is utilized; and Section 5.5 concludes with a methodology to analyze the incurred overhead.

5.1 Methodology and General Architecture

Methodology

In this section, we briefly describe a methodology that defines the steps that need to be followed to enable steering action data management in an adaptable application. Some of these steps occur offline, before the execution starts, whereas others occur online. The offline steps are mainly related to manually inserting the APIs calls in the workflow to capture data when the workflow executes. We extend our previous methodology [36] to add steering action data management support. The methodology presented in this thesis has been adapted from our paper SOUZA *et al.* [37]. Table 5.1 summarizes the high-level steps for enabling tracking of steering actions.

Table 5.1: Methodology for workflow steering.

1	Identify the workflow	Before execution
2	Add analysis points in the workflow	
3	Add adaptation points in the workflow	
4	Online data analysis	During execution
5	Execute a steering action	
6	Steering action data management	
7	Steering action data analysis	

In Step 1, users identify inputs and outputs of relevant parts of their application that form a workflow. That is, they identify and specify the data transformations, the datasets, and the data derivation paths. In Step 2, API calls are inserted in the workflow to add *analysis points*, which are the regions in the workflow that contain relevant (for the users) data for analysis at runtime. These data consist of relevant input and output data elements in the dataflow, to be analyzed online, either for monitoring or interactive analysis. In Step 3, users identify parts of the workflow that can be dynamically adapted at runtime and add *adaptation points* in those parts. Adaptation points should be added to safe points of the workflow to avoid execution or data inconsistencies. Usually, users know where to add adaptation points. A typical example occurs in iterative workflows where each new iteration is an opportunity to redefine parameters or input datasets preset beforehand. In this case, an adaptation point is added at the beginning of the iteration. Each iteration is often executed as a whole. When a user adapts, the adaptation will take effect only at the next iteration, rather than changing values during an iteration. This helps to make data and execution consistent with what the user decided to adapt during the iteration. After these three initial offline steps, the workflow is submitted to parallel execution in an HPC machine. In Step 4, the workflow data specified in the analysis points at Step 2 are captured and can be analyzed online. In Step 5, based on the analyses, users may decide to execute a steering action. In Step 6, the system captures the steering action data and relates the steering action to the rest of the workflow data being captured, and stores in the database. Finally, in Step 7, users analyze the consequences of their actions relating to domain-specific relevant data and execution data. In our experiments (Sec. 7.1), we provide concrete examples of how these steps are carried out in a real workflow.

General Conceptual Architecture

In this section, we present DfAdapter's general conceptual architecture (Figure 5.1). We describe its components as follows.

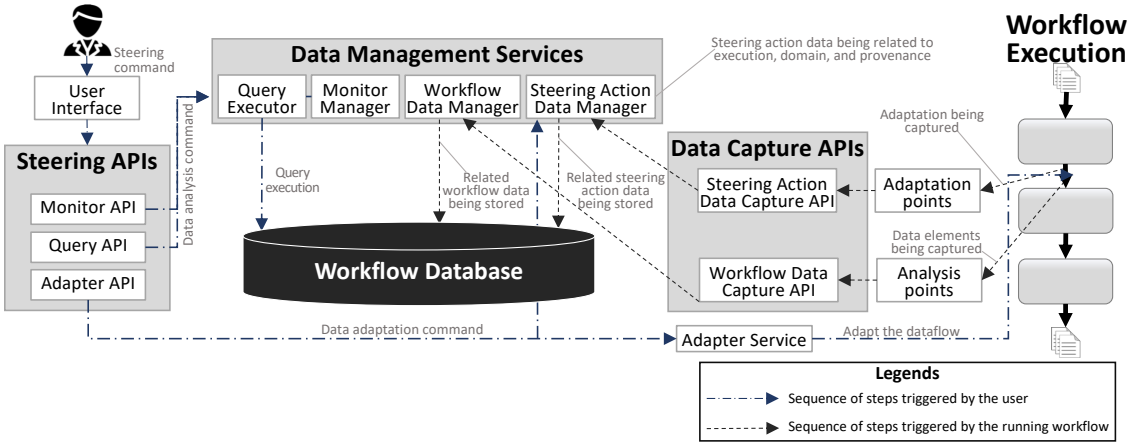


Figure 5.1: DfAdapter's General Conceptual Architecture.

Data Capture APIs

The Data Capture APIs are *Workflow Data Capture API* and *Steering Action Data Capture API*. The Workflow Data Capture API has the programming interfaces containing the methods to insert workflow data (mainly domain data values) into the workflow database. Calls to this API are inserted in the analysis points in the workflow. Steering Action Data Capture API has the programming interfaces with the methods to capture steering action data, when a steering action is issued by the user. Calls to this API are inserted in the adaptation points. These APIs are inserted in the workflow, called by workflow at runtime, and call the steering data management services.

Data Management Services

There are three main services composing the Steering Data Management Services: *Query Executor*, *Monitor Manager*, *Workflow Data Manager*, and *Steering Action Data Manager*. The Workflow Data Manager is responsible for receiving the calls from the Workflow Data Capture API with the captured data during workflow execution. Then, it provides the data relationships to the other data in the workflow database, integrating provenance, domain, and execution data, and finally stores the related workflow data in the workflow database. The Steering Action Data Manager listens to calls from the Steering Action Data Capture API. This service is responsible for consistently adding the data relationships between the captured steering action and the remainder of the workflow data, and storing the related steering action in the workflow database.

The Query Executor is only called by the Steering API. When the client, *i.e.*, the user or a data visualization application, performs a data analysis command,

like analysis or interactive analysis, the Query Executor is issued. It then builds the needed queries according to the client's requests and executes the query to the DBMS managing the workflow database to retrieve the requested data. The Monitor Manager is to implement the adaptive monitoring concepts (Sec. 4.5). The user programs the monitoring query by using the interface that calls the Monitor API, which calls the Monitor Manager service. This service is responsible for managing the monitoring queries, for submitting the queries to the Query Executor at each time interval programmed by the user, and for returning the query results to the requesting client, *i.e.*, the Monitor API.

Steering APIs

The Steering APIs are composed of three APIs: *Monitor API*, *Query API*, and *Adapter API*. The Monitor and Query APIs call the Query Executor for data analysis, either for workflow analysis or for interactive analysis, respectively. They contain programming interfaces that allow users to submit queries to the Query Executor. The Adapter API has the interfaces for calling the Adapter Service, which is the component in the adaptable application that knows how to adapt the workflow (see the definition of the Adapter software service in Sec. 4.4). The interfaces in the Adapter API are implemented according to the data communication implemented for the adaptable application, as we see in the next section (Sec. 5.2). The Steering API's submodules are called by the user using the *User Interface*, which concentrates the available commands the user can issue using the Steering API. It can be exposed as a command line interface, graphical user interfaces, or as a RESTful API.

5.2 Design Principles

In this section, we explain the core design principles followed by DfAdapter.

Dataflow-oriented Data Capture and Analysis. DfAdapter relies on a dataflow-oriented approach for workflows, as the one presented in Section 2.2, as this is the fundamental basis for WfSteer to manage steering action data, as discussed in Chapter 4. DfAdapter needs that the domain data flowing in the dataflow are captured and stored in a workflow database, during execution, jointly with execution and provenance data (Sec. 2.4). Considering workflow scripts, DfAnalyzer [27, 35, 38] is a tool that can be coupled with scripts to capture runtime data, aware of the dataflow, and stores these captured data in a database ready for analysis during execution. In the implementation section (Sec. 5.3), we explain how DfAdapter is combined with DfAnalyzer to provide rich runtime data analysis that integrates

provenance, execution, domain, and steering action data.

Asynchronous Service Calls. DfAdapter can be coupled to adaptable applications, like the ones proposed by approaches that allow for online adaptation (Table 3.1) or adaptable workflow scripts. In either case, to enable steering action data analysis, the Steering Action Data Capture API is used so it can be called from the adaptation points in the adaptable application. Similarly, to enable online workflow data analysis, Workflow Data Capture API calls are placed in analysis points in the workflow script to capture data elements flowing in the dataflow and execution data of data transformations.

Since CSE users typically discard any tool that can add significant overhead to their already long workflow executions, attaining low performance overhead is a basic requirement in DfAdapter. For this, calls to DfAdapter are asynchronous, meaning that when the user adapts the running workflow, the steering action data capture is done in such a way that the main computational process will not wait until the steering action data are completely stored. The same strategy is followed for any added Workflow Data Capture API in the workflow. Also, the most computationally costly components in DfAdapter, such as the ones that relate and store steering action data in the database during workflow execution, are deployed in separate hardware, different from where the main computational process runs, but in the same internal network (*e.g.*, the nodes in the HPC machine has local access to the node that runs the Data Management Services) to reduce communication costs, following *in-situ* and *in-transit* approaches [50]. This avoids making DfAdapter and the main computational process compete for resources.

Data Communication between DfAdapter and the Running Workflow. To be able to capture steering action data, DfAdapter needs to access the data being adapted in a steering action. For this, we characterize here the data communication between the Adapter Service and the adaptable application. By analyzing the state-of-the-art (Chapter 3), we identified three ways to implement such data communication. To access the data being adapted, DfAdapter needs to intercept this communication. We characterize these implementations as follows.

i. File-based checks — This is a simple yet widely used way to implement data communication [50]. In this case, there are files in a storage resource that are accessible both by the Adapter API and by the Adapter Service. That is, the Adapter API modifies a file that is accessible by the running workflow. When the program pointer in the running workflow reaches an adaptation point, the Adapter Service verifies if files were modified based on the last-modification date metadata provided by the file system API, in case of modification, the application is adapted. Then, the Steering Action Data Manager service is called to relate and store the adaptation. Although file-based checks are a simple approach, it is widely used

especially by users that implement their own *ad-hoc* way to make their simulation steerable. However, it requires that the Adapter API and the Adapter Service share access to files in a storage resource, which may not be always possible. This is the way libMesh-sedimentation (Sec. 2.3.1), used in our experiments (Sec. 7), implements the data communication.

ii. Message passing — In this case, the Adapter API sends a message to the Adapter Service in the running workflow. When the adaptation point is achieved, the Adapter Service verifies if a message has arrived and effectuates the adaptation accordingly, and DfAdapter is called to relate and store the steering action data. MPI, sockets, or RESTful HTTP messages can be used as communication protocol to implement this. Many systems that enable adaptation use message passing to implement the data communication [4, 87, 94]. This is an alternative to file-based checks, as it does not require files to be shared in a storage resource by the adapter's front and back ends.

iii. DBMS-driven — It is similar to file-based checks in the sense that there is a DBMS that is accessible both by the Adapter API and the Adapter Service in the running workflow. It is similar to message passing in the sense that it does not require files to be shared in a storage resource. Rather, data that can be modified at runtime are managed by the DBMS that can even run in-memory, depending on the DBMS. In this implementation, when the user uses the user interface to adapt, the Adapter API modifies the data in the DBMS. When the program pointer achieves the adaptation point in the running workflow, the Adapter Service checks if the data have been modified, effectuates the adaptation accordingly, and DfAdapter is called to relate and store the steering action data. We tested this implementation in a synthetic workflow example using the parallel framework Apache Spark and Redis, a lightweight in-memory Key Value store, as the DBMS between the workflow and DfAdapter. The source code is available on GitHub [127] and in Appendix B.

DBMS and Data Model for the Workflow Database. DfAdapter needs a DBMS to manage the workflow database. Several data models can be used for workflow databases, such as graph and relational data models. The usage pattern in DfAdapter is that the running workflow only produces insertions to the database, while the user typically runs provenance queries for online data analyses to support decision-making, *i.e.*, Online Analytical Processing (OLAP) queries. This usage pattern, both by the running workflow and by the user, is benefited from column-oriented relational DBMSs, as shown in previous works [27, 35]. Moreover, since there may be many appends to this database during execution, the DBMS must be able to handle parallel requests from clients. One available option for this is the open source DBMS MonetDB[1], which is the one currently in use by DfAdapter.

[1]https://www.monetdb.org

5.3 Implementation Details

This section describes the implementation details of DfAdapter. The steering action currently implemented in DfAdapter is the user-steered parameter tuning, which is by far the most supported one by user steering approaches, as we surveyed in Chapter 3. Here we describe the sequence of steps that are executed to manage steering action data, as illustrated in the sequence diagram in Figure 5.2. First, during the workflow execution, (0) the data specified in the analysis points are sent to the Workflow Data Manager Service via Workflow Data Capture API calls. Then, (1) Workflow Data Manager service relates and stores the data in the Workflow Database. While the workflow runs, the user can call Query or Monitor APIs to follow the intermediate data results and decide for an adaptation action. If the user decides for an adaptation action, (2) the user sends a steering command using DfAdapter's steering command line interface (implemented as a program called `WfSteerCtl`), which (3) calls the Adapter API, which (4) calls the Steering Action Data Manager service, waits for the service callback, then (5) registers the beginning of an adaptation intention. The Adapter API also (6) communicates with the Adapter Service, which (7) effectuates the adaptation. When (8) the running workflow notices that an adaptation occurred (*e.g.*, it verifies that a file or a data structure, depending on the data communication implementation, has been changed because of a steering action), the (9) Steering Action Data Capture API inserted in the adaptation point is called to send the captured adaptation data to the Steering Action Data Manager service. (10) The service receives the call, relates the steering action with the workflow data being processed, and stores in the Workflow Database. After that, the workflow continues to run, and finally (11) the user can run user steering action analysis.

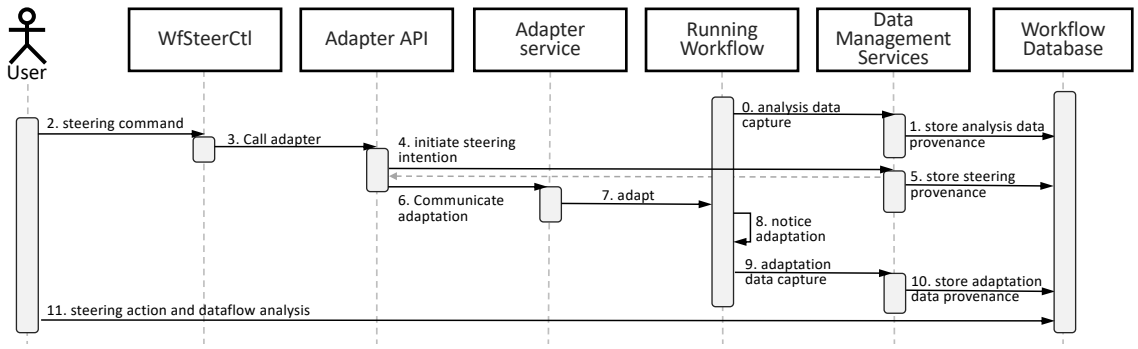


Figure 5.2: Sequence diagram for managing steering action data with DfAdapter.

The *Tune* operator (Def. 4.3) is implemented to call the Steering Action Data Capture API. Before the call, the steering action data need to be prepared. Algorithm 1 defines how the steering action data need to be prepared before sending them to the Data Management Services. The iteration data at the moment of the

action is denoted by ξ; the wall time at the moment of the action is denoted by t; and the set of ordered pairs with the old values for the tuned parameters are denoted by θ. The algorithm registers new domain data that were modified in the adaptation as well as their corresponding old values. It captures current execution data, iteration counter values (in case of iterative workflows), and user data. Then, it stores user steering data relating to the rest of the workflow data being continuously captured during workflow execution.

Algorithm 1: Capturing User Steering Action Data for *Tune*.

Input:
I_{DS}: The input dataset in the dataflow containing the parameters to be tuned.
η: key-value data structure containing the parameters and their new values.
$C <$ optional $>$: criteria to specify the data element that is being adapted.
$\mu <$ optional $>$: metadata about the steering action, such as annotations.

```
1  import data_capture_api
2  θ ← ∅
3  ξ ← ∅
4  Γ ← data_capture_api.get_running_tasks()
5  U ← data_capture_api.get_user()
6  t ← data_capture_api.get_current_wall_time()
7  current_data_element ← data_capture_api.get_element(I_DS, C)
8  attribute_semantics ← data_capture_api.get_semantics(I_DS)
9  for all key-value pairs (p, current_value) in current_data_element do
10     if p ∈ keys(η) then
11         θ[p] ← current_value
12         if p ∈ attribute_semantics[L_I] and ξ = ∅ then
                       // Tuning a loop attribute.  Get iteration data
13             ξ ← data_capture_api.get_current_iteration_data(I_DS)
14 data_capture_api.send_steering_action_data(I_DS, η, C, U, Γ, μ, ξ, t, θ)
```

Workflow Database Schema

To implement the PROV-DfA provenance data diagram presented in Section 4.4, we use the relational data model. An excerpt of the relational database schema in use by DfAdapter is in Figure 5.3, whereas a complete figure can be found on GitHub [127]. Whenever a user issues a steering command to tune parameters, a new instance of parameter tuning action is stored in the `ParameterTuning` table. Since a parameter tuning may modify one or many attributes, and the same attribute may be modified by many steering actions, there is a many-to-many relationship between `ParameterTuning` and `Attribute` tables. The associative table, `ParameterTuned`, has fields to store old and new values. The I_{DS} is a specialization of the table `Dataset`. Each tuple in the `Dataset` table is a data element. Each `ParameterTuning` instance may directly affect one or many data elements

in I_{DS} and a same data element in I_{DS} may be affected by many parameter tuning actions, hence there is a many-to-many relationship between `ParameterTuning` and `Dataset` tables, via the `ModifiedDataElement` associative table. Moreover, as O_{DS} is also specialization of `Dataset`, we use `InfluencedDataElement` associative table between another many-to-many relationship between `ParameterTuning` and `Dataset` tables to store output data elements directly influenced by a tuning, such as iteration counter data in case of parameter tunings in data transformations that evaluate loops. Finally, we relate execution data about the current state of the execution when a tuning action happened via the associative table `InfluencedTask`. Tasks are directly mapped to `ExecuteDataTransformation` in PROV-DfA, and execution data are further extended with performance data via the relationship between `Task` and `Performance` tables. The `Person` who steered and the `Adapter` service used in that specific steering action are related and stored to `ParameterTuning`. Thus, because of these entities and relationships are populated during the workflow execution in a user-accessible database, users can drive their analyses and decisions at runtime using these data.

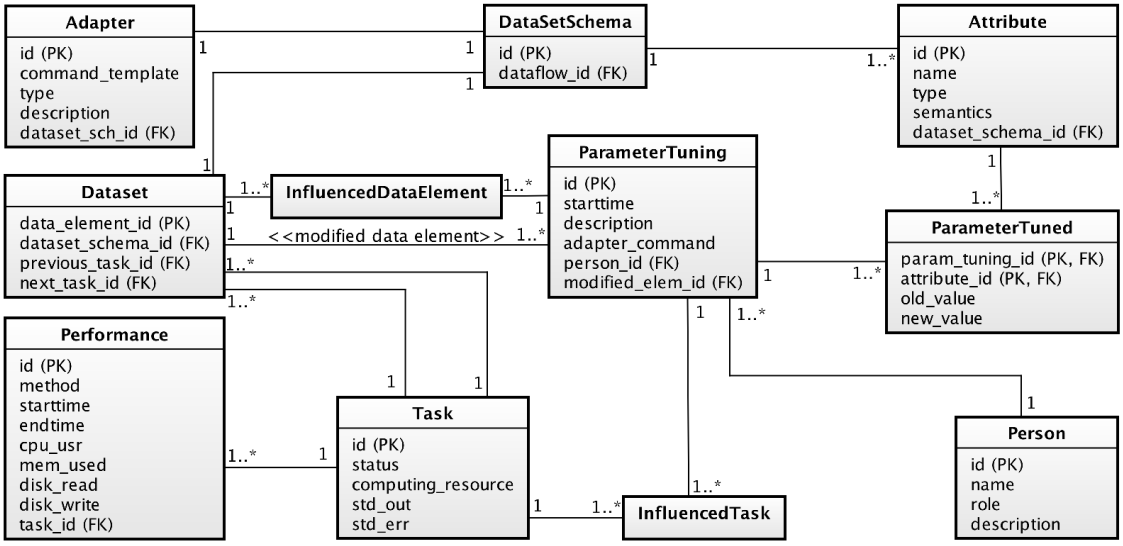


Figure 5.3: Workflow Database Schema for DfAdapter.

Combining DfAdapter with DfAnalyzer

DfAdapter needs a Workflow Data Manager to provide the data that will give the context of the steering actions, by relating the actions with the overall workflow data being generated during execution. DfAnalyzer [27, 35, 38] follows the dataflow-oriented approach (Sec. 2.2), which is the basis for DfAdapter. Also, DfAnalyzer follows the same design principles presented in Section 5.2 to make data capture efficient, with low execution overhead, and with efficient analytical queries. Therefore, DfAdapter uses DfAnalyzer as its Workflow Data Manager service with its corre-

sponding Workflow Data Capture API. In DfAnalyzer, the Workflow Data Manager service is called Provenance Data Extractor and it is accessible via calls from its RESTful data capture API inserted into the workflow script in the analysis points. Together with the Steering Action Data Capture API inserted in the adaptation points, DfAdapter and DfAnalyzer capture steering action data and data elements in the dataflow from the running workflow, respectively. Both use the same workflow database, although we modify its schema to add the steering action data-related tables, following the PROV-DfA as in Figure 5.3. Finally, during workflow execution, DfAdapter relates and stores the steering action data captured by its Steering Action Data Capture API to the workflow data captured by DfAnalyzer's Workflow Data Capture API in the same integrated workflow database, ready for provenance analyses of the steering actions.

5.4 Utilization

To describe how DfAdapter is used, we resort to the methodology in Section 5.1. We explain how it can be added to dynamic workflows before execution and how it can be used to manage steering action data.

Before execution. The user identifies a workflow by specifying parts of the workflow that can be modeled as data transformations and their datasets, and the data derivation paths. Analysis and adaptation points are added into the workflow script, with their corresponding calls to capture workflow data and steering action data. Listing 5.1 shows an example using an excerpt of libMesh-sedimentation workflow script (Sec. 2.3.1) with the added data capture calls. In this example, a programming library to capture the data is imported into the workflow script. To capture workflow data, it calls DfAnalyzer's Workflow Data Capture API and to capture steering action data it implements the Algorithm 1, which calls the Steering Action Data Capture API. The underlying implementation details, *e.g.*, usage of a programming library, will depend on the workflow script being instrumented with our API calls. This step is usually done in collaboration between the workflow specialists and the CSE users, who know how the code of their workflow script and know which data should be captured and how the calls should be placed in their script.

```
...
for (unsigned int t_step = p.init_tstep;
    (t_step < p.n_time_steps) && (time < p.tmax); t_step++) {
    data_capture_lib.init_time_loop();
    if (parameters_modified()) {
        p = reload_parameters();
        data_capture_lib.capture_steering_time_loop();
    }
    ...
    for (unsigned int nonlinear_step = 0;
        nonlinear_step < p.max_nonlinear_steps; ++nonlinear_step) {
            data_capture_lib.init_fluid_solver();
            flow_system.solve();
            ...
            data_capture_lib.finalize_fluid_solver();
    }
    ...
    data_capture_lib.finalizeTimeLoop();
}
```

Listing 5.1: Excerpt of libMesh-sedimentation code with Data Capture API calls.

During execution. When the workflow is running, the user can send an adaptation command using DfAdapter's user interface, which implements the calls to the Adapter API. DfAdapter's command line-based interface, `WfSteerCtl`, to be used in a terminal connected to the HPC machine where the workflow runs. In the command line, users only need to inform the input dataset I_{DS} to be adapted, and a simple key-value data structure containing the parameters and their new values. For flexibility, the key-value data structure can be passed directly using the argument `--p` or can be written into a file to be passed as an argument. We provide an optional argument `--reason` to allow users to annotate that specific steering action. Listing 5.2 shows an example of DfAdapter's command line interface.

```
$> WfSteerCtl --user="Bob"
$> WfSteerCtl --tune
      --dataset="I_Iteration_Params"
      --p='{"max_linear_iterations": 500}'
      --reason="checking how linear iterations affects
                convergence"
$> echo '{
    "flow_initial_linear_solver_tolerance": 1.0e-6,
    "amr/c_fraction": 1.0e-5
   }' > new-values.json
$> WfSteerCtl --tune
       --dataset="I_Solver"
       --p="new-values.json"
```

Listing 5.2: Command lines to use DfAdapter.

5.5 Methodology to Analyze the Overhead

The adoption of our approach depends on how much execution time overhead it implies. The overhead depends on the data needed for analysis and adaptation. For analysis, it depends on the workflow data identified in the workflow script that needs to be captured. That is, which input and output data values, for each data transformation, should be monitored during execution. For adaptation, which adaptation points should be added and how many adaptation actions happened during execution. In both cases, the overhead depends on data collected for analysis and adaptation actions, always based on user decisions.

We use the dataflow-oriented concepts (Secs. 2.2 and 4.2) to express the overhead. Let γ be a data transformation execution, *i.e.*, a task. When a task γ is executed to perform a data transformation DT_y, the execution cost of γ, $c(\gamma)$, is given by its actual computational cost $comp(\gamma)$ (*i.e.*, the inherent cost of executing DT_y) plus the overhead $o(\gamma)$ introduced because of the utilization of our approach:

$$c(\gamma) = comp(\gamma) + o(\gamma) \tag{5.1}$$

Let the overhead $o(\gamma)$ of a task γ be expressed as a function of analysis $anl(\gamma)$ and adaptation adp $s(\gamma)$ overhead introduced in the analysis and adaptation points in the workflow script code, respectively:

$$o(\gamma) = anl(\gamma) + adp(\gamma) \tag{5.2}$$

The overall overhead is given by the sum of $o(\gamma)$ for all tasks γ, of all data transformations $DT_y \in T$. Next, we detail the analysis and adaptation components.

Analysis overhead. Analysis overhead $anl(\gamma)$ is defined by the data capture overhead $anl_{point}(\gamma)$ and raw data extractions $ext(\gamma)$ during each data transformation execution identified by the user as relevant for analysis:

$$anl(\gamma) = anl_{point}(\gamma) + ext(\gamma) \tag{5.3}$$

Provenance data capture overhead $anl_{point}(\gamma)$ depend on the number of data values of each data element captured at a task execution γ. Each execution γ of a data transformation DT_y consumes input data elements in I_{DS} and produces output data elements in I_{DS}. In DfAdapter, data elements are stored at once in the beginning (input data elements) and at the end (output data elements) of each task γ.

Raw data extraction overhead $ext(\gamma)$ depends on the number of data values the user wants to extract from raw data files at each execution of a data transformation DT_y. Let V_γ be the set of all data values extracted when γ is executed. Each extracted data value $v_i \in V_\gamma$ has an associated data attribute a_i has semantics in V_I or V_O, depending on if v_i is in a data element in I_{DS} or O_{DS}, respectively. Therefore, the extraction overhead $ext(\gamma)$, for each γ to execute a data transformation DT_y is therefore given by the summation of costs to extract each $v_i \in V_\gamma$:

$$ext(\gamma) = \sum_{v_i \in V_\gamma} ext(v_i) \tag{5.4}$$

The cost to extract a data value depends on application-specific raw data extractors [35].

Adaptation overhead. The adaptation overhead occurs in data transformations that have an adaptation point. Adaptation overhead also depends on when an adaptation action happens. When an adaptation action happens, all those operations presented in the sequence diagram in Figure 5.2 are triggered. Let $T' \subset T$ be the subset of the data transformations that have adaptation points. Then,

$$adp(\gamma) = adp_{point}(\gamma) + action(\gamma) \tag{5.5}$$

where $adp_{point}(\gamma)$ is the overhead associated to adding adaptation points to DT_y, $action(\gamma)$ is the overhead associated to computing that an adaptation action happened, and $adp_{point}(\gamma) = action(\gamma) = 0, \forall DT_y \notin T'$.

Putting it all together. The overall cost $c(Df)$ to compute a dataflow Df is given by the sum of costs to compute the actual computation $comp(Df)$, provenance capture $anl_{point}(Df)$, raw data extractions $ext(Df)$, adaptation points $adp_{point}(Df)$,

adaptation actions $action(Df)$ for the entire dataflow Df. That is:

$$\begin{aligned} c(Df) &= comp(Df) + o(Df) \\ &= comp(Df) + anl(Df) + adp(Df) \\ &= comp(Df) + anl_{point}(Df) + ext(Df) + adp_{point}(Df) + action(Df) \end{aligned} \tag{5.6}$$

where $c(Df) = \sum_{\gamma} c(\gamma)$, for all tasks γ, for all $DT_y \in T$. Analogously, the components of $c(Df)$ can be obtained by the summation of each individual component for all tasks. That is, $anl_{point}(Df) = \sum_{\gamma} anl_{point}(\gamma)$, $ext(Df) = \sum_{\gamma} ext(\gamma)$, and so on.

In CSE applications, tasks are often complex, meaning that for a task γ, $comp(\gamma)$ takes at least a few seconds, but often minutes long [128]. Experimentally analyzing the individual elapsed time of the components, $anl_{point}(\gamma), adp_{point}(\gamma), adp_{action}(\gamma)$ of the overhead $o(\gamma)$, we observe that, on average, they are close to constant and typically milliseconds-long. Therefore, we can assume that in typical CSE applications $comp(\gamma) >> o(\gamma)$, which leads to the generalization that overhead of capturing user steering action data is negligible. Also, because such operations occur asynchronously and in a different computing resource, the time for the individual components of $o(\gamma)$ is "hidden" by the actual computation, which is significantly higher. This contributes to reduce the impact on the workflow execution performance. If we consider $ext(Df)$ which depends on the user settings, it is still typically very much smaller than the raw data that is being generated and stored on files. As we show in our real case study (Sec. 7.1.3), the overall $o(Df)$, including the costs for $ext(Df)$, is less than 2%, which is still negligible.

In the next chapter, we present the second instantiation of WfSteer to support a data-centric WMS.

Chapter 6

Managing User Steering Action Data in a WMS

The second instantiation of WfSteer aims at allowing for tracking steering actions with low execution overhead in a large-scale execution in a WMS. The main difference from the implementation for workflow scripts is that WMSs are responsible for controlling the parallel execution of the workflow on the HPC machine. Because of this, an important part of a WMS philosophy is to provide services that alleviate the burden to the users concerning parallel execution control issues, such as efficient exploitation of the parallel hardware and consistent execution. The importance of this for steering action data management is that the WMS must not only provide services to capture, relate, and store steering action data, but must also address issues related to keeping the execution consistent when the user is performing an adaptation and, as any WfSteer implementation, keep the introduced overhead low. In this chapter, we present our implementation in d-Chiron. We discuss design principles (Sec. 6.1), implementation details related to consistency control when the user is adapting the workflow (Sec. 6.2), then how the user utilizes the WMS for steering (Sec. 6.3).

6.1 Design Principles

In this section, we present the design principles that drive the architectural, technological and implementation decisions of WfSteer in a WMS.

A Data-centric WMS. A data-centric WMS follows a similar philosophy proposed by our dataflow-oriented approach in the sense that the dataflow, rather than the execution flow, is a first-class-citizen (Sec. 2.2). Chiron [20, 26] is the only existing WMS that implements a data-centric approach for workflow management. As any WMS, it must manage the parallel execution control, however its main pur-

pose is to provide for powerful online data analytical capabilities to users so they can steer a running workflow. Chiron always has the most up-to-date data for analysis during an experiment run, ready for joint analysis of domain, execution, and provenance data because it uses a single, unified workflow database both as its only source of data for parallel task scheduling data management and to provide for data analysis. d-Chiron [41] is a highly distributed, scalable version of Chiron. Its main differentiator compared to any other WMS is that it relies on an efficient in-memory distributed DBMS to deal with distributed and parallel concurrency control in a large-scale execution. It is a good alternative to keep the execution overhead low while allowing for workflow steering. For these reasons, we choose d-Chiron to implement WfSteer concepts.

Fast Hybrid Transactional and Analytical Workloads. When the user sends an adaptation command, the WMS needs to react and respond to the user call as soon as possible. The steering action is delimited by a clause α (Def. 4.1), which often delimits the subset of the dataset in the dataflow that is being steered. Thus, to implement the steering action in a WMS, the DBMS managing the workflow database needs to implement efficient indexing and Online Transactional Processing (OLTP) querying capabilities to search for specific subsets in large sets of data. Additionally, after the users adapt, they can analyze the consequences of the adaptation. In this case, efficient analytical queries to perform joins over multiple datasets, aggregations, sorting, and filters are required to enable users to perform complex data analyses. Therefore, the DBMS must allow for both OLTP and OLAP workloads.

Consistency Control during a Steering Action. It is essential that the data remains consistent within a user steering action. It is quite complex to guarantee a consistent execution when a user decides to adapt the workflow, in particular in a large HPC execution and without stopping the workflow execution. On the other hand, distributed relational DBMS natively provide atomicity, consistency, isolation, and durability (ACID) transactions [129]. The WMS can take advantage of this capability and implement the steering actions in a way to outsource to the DBMS complex transaction control that guarantees consistency.

The in-memory distributed DBMS in d-Chiron is the MySQL Cluster[1], a relational DBMS that ensures strong-consistent ACID transactions, has a high throughput to deal with multiple concurrent tasks, and it supports both transactional and analytical workloads. Thus, MySQL Cluster is a good choice to follow the presented principles.

[1]https://www.mysql.com/products/cluster

6.2 Implementation Details

In this section, we discuss the implementation details for user steering action data management in d-Chiron. The steering action we implemented in d-Chiron is user-steered data reduction. Since it is a WMS and as such it follows the philosophy to alleviate the burden to the users concerning execution control issues, a significant part of the implementation efforts are dedicated to addressing consistency problems when the user is adapting the workflow (Sec. 6.2.1). Then in Section 6.2.2 we discuss further implementation details on steering action data management in d-Chiron. Finally, in Section 6.2.2 we present our implementation of the adaptive monitoring concepts.

6.2.1 Addressing Steering Action Data Consistency Issues

This section describes how a consistent data reduction is implemented in a data-centric WMS.

Consistency Issues in a User-steered Data Reduction

The data reduction steering action follows the Cut definition (Def. 4.4). Cut can only operate on input data elements that are waiting to be processed in the workflow. When a Cut happens, the dataset I_{DS} that will be reduced is a shared resource between the WMS engine that is normally processing the workflow in a batch job and the user who wants to remove a slice, delimited by the criteria C, from I_{DS}; hence, race conditions can occur. Suppose, for example, that at a given instant t in time, the WMS finishes processing a set of data elements and then needs to get new data elements that were waiting to be processed. If at the same time t, the user decides to cut off some of those input data elements that were waiting to be processed, the WMS may go to an inconsistent state because it could try to process elements that were removed. Or, the user may try to remove a slice that the WMS already considered to process, thus generating errors. These inconsistencies are even more likely to occur in a highly concurrent execution, such as executions on large HPC clusters with thousands of computing cores, as the ones we use for typical CSE workloads.

To address this problem, we define a *safe* subset of an input dataset I_{DS} to which the data reduction is applied. We split I_{DS} into two subsets $G \subset I_{DS}$ and $H \subset I_{DS}$, where G has the input data elements that have already been processed and H has the elements waiting to be processed. That is, $I_{DS} \leftarrow G \cup H | G \cap H = \varnothing$. H is the subset of I_{DS} that is safe to remove a data slice from. To guarantee this, the WMS must provide lock controls so that only the subset H will be reduced, as we show

next. Figure 6.1 illustrates the separation of the safe subset and the general view of how a data-centric WMS executes a data reduction using the *Cut* operator, using an excerpt of the Risers Fatigue Analysis workflow (Sec. 2.3.2) as example.

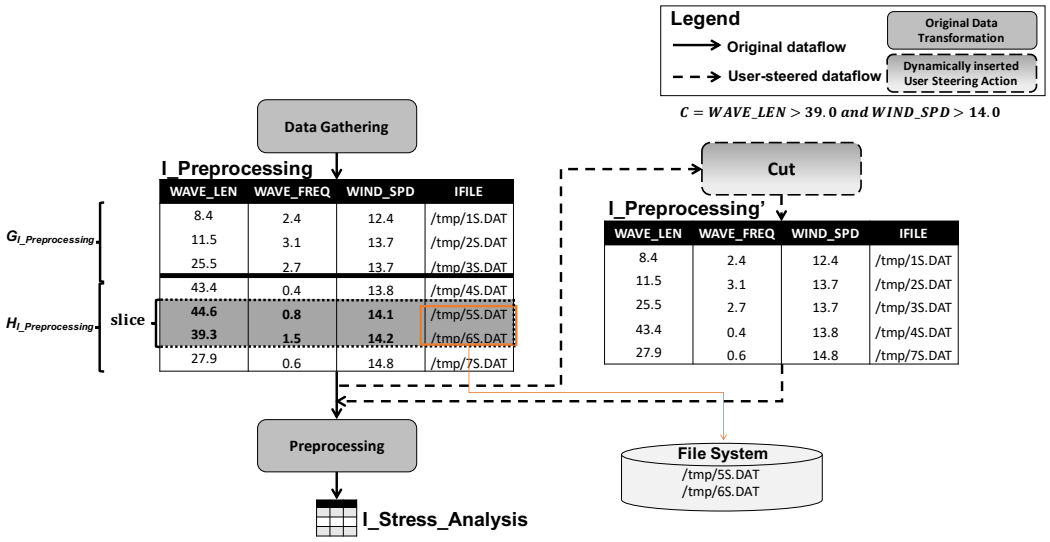


Figure 6.1: User-steered data reduction using the *Cut* operator. The input dataset $I\_Preprocessing$ is split into subsets $G_{I\_Preprocessing}$ and $H_{I\_Preprocessing}$. A slice following the criteria $C = WAVE\_LEN > 39.0 \wedge WIND\_SPD > 14$ is cut off from $I\_Preprocessing$ transforming it into $I\_Preprocessing'$.

Addressing Consistency Issues using a Relational DBMS

Before diving into the details of consistency issues when effectuating a data reduction at runtime, we explain how the slice delimited by the criteria C is defined in d-Chiron, using its relational DBMS, so it can later be safely removed while guarantying consistency after reduction.

Input data elements are consumed by the many parallel tasks (usually thousands in Many-Task Computing workflows [128]) that need to be scheduled by the WMS engine. To represent the dataflow-oriented approach (Sec. 2.2) in a relational database schema, the datasets $DS \in D$ map to input and output specializations of `Dataset` relations (*i.e.*, tables), according to the domain modeling of the managed application. The tuples of the `Dataset` relations map to the data elements $e_i \in DS$. The data transformation executions (*i.e.*, tasks) are mapped to a `Task` table, where each task is related to one or more data elements of a dataset.

Thus, for a certain input dataset $I_{DS} \in D$, the join $I_{DS} \bowtie Task$ returns a set containing tasks with their input data elements in I_{DS}. Moreover, among other attributes, each task has an important state attribute that determines if a task is READY to be executed (already knows its input data to start, but is waiting for a free CPU so it can be scheduled), `RUNNING`, `COMPLETED` (already been successfully executed), `BLOCKED` (even though there may be free CPUs, the task does not have

the input to start yet), or any other state a task may assume. Depending on the data transformation, each task may consume one or more input data elements. Therefore, we distinguish between (i) data transformations in which each task consumes one data element (we denote such tasks as $tasks^{1:1}$, and (ii) data transformations in which each task consumes more than one data element (we denote them as $tasks^{1:n}$).

(i) In data transformations with $tasks^{1:1}$, removing input data element means "informing" the WMS not to execute the tasks that would consume them, hence reducing overall execution time. To implement the separation of the input dataset I_{DS} into G and H, a semi-join relational operation [129] is used to join input data elements from the input dataset I_{DS} with tasks in `READY` state to only select the domain data elements that still need to be processed. Then, after having H, the WMS can obtain the elements in H that follow the criteria C. Since a set containing both the input data elements together with the related tasks that will consume them is important for the implementation of data reduction, we denote this set as $\S^{1:1}$:

$$\S^{1:1} \leftarrow \sigma_C(I_{DS}) \ltimes \sigma_{state=READY}(Task)$$

where the ratio 1:1 means that the tasks in this set are $tasks^{1:1}$ and the criteria C is defined in the Cut operator. Finally, the WMS will know that tasks in $\S^{1:1}$ should not be processed.

(ii) In data transformations with $tasks^{1:n}$, a data reduction in an input dataset I_{DS} can only occur if the task that will consume them is in a `BLOCKED` state. The task has not started yet because the needed input data for it to start is still being generated by a running task in a previous data transformation. When this running task finishes, it signals that the `BLOCKED` task can start. While it is still blocked and the input data elements are being generated, the user can analyze them and identify data values that can be removed. In this case, we denote the set $\S^{1:n}$ similarly to the previous one, but it rather returns the input data elements in I_{DS} that are being consumed by the tasks in `BLOCKED` state:

$$\S^{1:n} \leftarrow \sigma_C(I_{DS}) \ltimes \sigma_{state=BLOCKED}(Task)$$

where the ratio 1:n means that the tasks in this set are $tasks^{1:n}$. Finally, the WMS will know that tasks in $\S^{1:n}$ should not be processed. This is different for $tasks^{1:1}$ because they cannot be executed if their input datasets are removed. The WMS will know how to handle the tasks in sets $\S^{1:1}$ or $\S^{1:n}$. For this, it needs to know beforehand, using prospective provenance, whether the data transformation corresponding to the tasks that would consume the elements defined by the criteria C is expected to consume one element per task or many elements per task. Such verification is important to guarantee consistency during reduction.

6.2.2 Further Details

Implementing Steering Action-related Consistency Control in d-Chiron

To ease slice removal in d-Chiron, we developed a program to be used as a command line-based user interface to steer the workflow in d-Chiron. Similarly to what we did for DfAdapter, we call this program as `WfSteerCtl`. With `WfSteerCtl`, users can issue command lines to inform the name of the input dataset I_{DS} and the criteria C. The slice delimited by C is added to the `where` clause in the SQL query that will form the `select` expressions. As an implementation decision, instead of physically removing the input data elements (either in $\S^{1:1}$ or $\S^{1:n}$) from the workflow database, we move them to a `Modified_Elements` table, maintaining the relationships. Likewise, the tasks in $\S^{1:1}$, which cannot be executed, are not physically removed, but they have their state marked as `REMOVED_BY_USER`. By doing so, we enable these tasks and data elements to be later analyzed with provenance queries.

To guarantee consistency, we take advantage of d-Chiron's DBMS with ACID transactions. In a user-steered data reduction, both d-Chiron's engine and the `WfSteerCtl` program need to concurrently update shared resources: `Task` and `Dataset` tables in the workflow database. The `WfSteerCtl` program knows if it is about to reduce data elements within a slice of the type $\S^{1:1}$ or type $\S^{1:n}$, since it depends on the dataset being reduced, which is a parameter to the module. Considering the input data elements (either in $\S^{1:1}$ or in $\S^{1:n}$), while d-Chiron's engine gets the input data elements to execute, the `WfSteerCtl` program needs to concurrently move the cut off input data elements to the `Modified_Elements` table. Considering the tasks in $\S^{1:1}$, the `Task` table is a shared resource because while d-Chiron's engine updates the runnable tasks (select them, update their status to `RUNNING`, execute them, and mark them as completed), `WfSteerCtl` needs to update the `Task` table to mark the tasks as removed by the user, so that the engine will not get them for execution. These concurrent actions make concurrency control critical. Figure 6.2 illustrates these steps with a sequence diagram. The `WfSteerCtl` program acts concurrently with the WMS engine on the shared resources, which are in red in the figure. The steps 1–3 in `WfSteerCtl` are put together in a single DBMS transaction, *i.e.*, it is atomic.

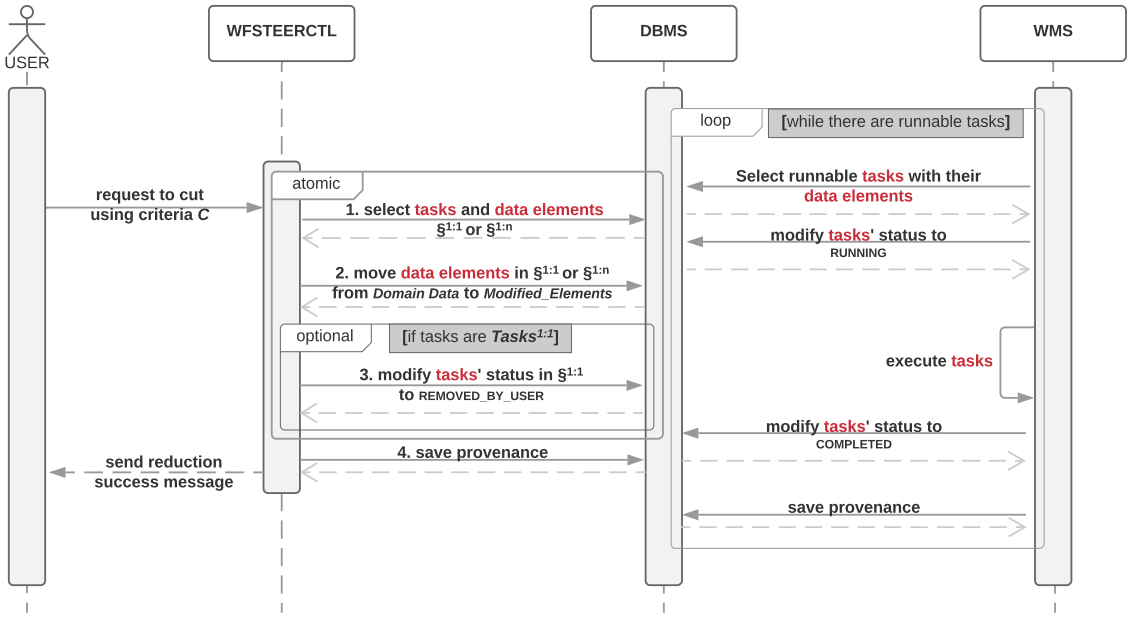


Figure 6.2: Sequence diagram showing what happens in a user-steered data reduction.

The workflow database tables are distributed, thus making concurrency control of the tables' partitions even more complex. In d-Chiron engine, distributed concurrency control in these tables is outsourced to the DBMS that guarantees the Atomicity, Consistency, Isolation, Durability (ACID) transaction properties [129]. We developed the `WfSteerCtl` program in d-Chiron to also exploit the DBMS in a way that the concurrency caused by the aforementioned updates is controlled by the DBMS. Therefore, we implement our approach such that both d-Chiron engine and the `WfSteerCtl` rely on the DBMS to outsource those complex distributed locks and releases of shared resources to guarantee that both execution and data remain consistent before and after a user-steered reduction.

Workflow Database Schema

The data schema that governs the data organization in the workflow database follows PROV-Wf [29], but d-Chiron has extensions to PROV-Wf to accommodate the steering action data. Particularly, `User_Query`, `Monitoring_Query`, and `Monitoring_Result`. Using PROV nomenclature, `User_Query` is a `PROV Activity` that stores the slice that represents sets of data elements that will be removed. `Monitoring_Query` is a `PROV Activity` that contains the monitoring queries submitted by the user in specific time intervals. The monitoring queries generate `PROV Entity Monitoring_Result` that stores the query results. These extensions, presented in 2017 [25], are the first extensions to a PROV data diagram for managing steering action data and are the basis for what we defined later for the PROV-DfA

data diagram [42] (Sec. 4.4).

To store provenance of removed data elements, we extend the workflow database schema with the table `User_Query` to store the queries that select the slice of the dataset to be removed. The description for each `User_Query` column is in Table 6.1. The removed data elements are stored in table `Modified_Elements`, which is a table that represents a many-to-many relationship between `User_Query` and `Dataset`.

Table 6.1: `User_Query` table description.

Column Name	Description
`query_id`	Auto increment identifier
`slice_query`	Query that selects the slice of the dataset to be removed.
`tasks_query`	Query generated by the WMS to retrieve the ready tasks associated.
`issued_time`	Timestamp of the user interaction.
`query_type`	Field that determines how the user interacted. It could be "Removal", "Addition", and others.
`user_id`	Relationship with the user who issued the interaction query.
`wkfid`	To maintain relationship with the rest of workflow execution data.

To store the queries in set QS of our adaptive monitoring approach (Sec. 4.5), we create the `Monitoring_Query` table, shown in Table 6.2. The main advantage of storing monitoring results in the workflow database (and adequately linking the results with the remainder of the data already stored in this database) whenever a monitoring query result is executed is that users can query the results immediately after their generation. The workflow database can also serve as data source for data visualization applications. We add another table: `Monitoring_Query_Result`, shown in Table 6.3, to store monitoring results in the workflow database.

Table 6.2: `Monitoring_Query` table description.

Column Name	Description
`monitoring_id`	Auto increment identifier.
`interval`	Interval time (in seconds) between each monitoring query (d_i).
`monitoring_query`	Raw SQL query to be executed.
`wkfid`	Relationship between the monitoring queries and the current execution of the workflow. In d-Chiron's workflow database, there may be data from past executions for a same workflow.

Table 6.3: `Monitoring_Query_Result` table description.

Column Name	Description
`monitoring_query_id`	Auto increment identifier.
`monitoring_id`	Relationship with the monitoring query that generated this result.
`monitoring_values`	Results of the `monitoring_query`.
`result_type`	Data type of the result values of both queries. Currently, "Integer", "Double", "Array".

Adaptive Monitoring Implementation

Now we provide implementation details for the adaptive monitoring concepts presented in Section 4.5. We implemented a Monitor Manager service that can access the same workflow database used by d-Chiron. A command line starts the Monitor Manager service that runs in the background. Connection settings are provided in a configuration file. Then, the Monitor Manager keeps querying the `Monitoring_Query` table at each s time units to check if a new monitoring query was added. The default value for s is 30 seconds, as the time interval to check if monitoring queries were added or removed. After the service has started, users can add (or remove) monitoring queries to (or from) the `Monitoring_Query` table. Currently, users can add monitoring queries using a command line to inform which SQL query will be executed at each time interval and the time interval itself. Whenever the Monitor Manager service identifies that the user added a new monitoring query, it launches a new thread. Each thread is responsible for executing each monitoring query $q_i \in QS$, stored in table `Monitoring_Query` at each defined time interval Δt_i. A thread is finished when a monitoring query is removed or when the workflow stops executing (in that case, all threads are finished). Figure 6.1 shows the steps executed at each time interval.

```
1 Execute the monitoring query q_i.
2 Store query results in the workflow database.
3 Reload information for q_i from the workflow database for the next time
    iteration t+Δt_i. The user could have adapted any of this
    information.
4 Wait for Δt_i seconds.
```

Listing 6.1: Steps executed by each thread within a time interval.

To enable these steering capabilities, three of these steps represent queries to the workflow database, including reads and writes. The stored results can be further

analyzed *a-posteriori* or, more interestingly, used as input for runtime data visualization tools, since results are immediately made available after they are generated. In our experiments, Section 7.2.2, we show how users can interact with d-Chiron to add monitoring queries and use the `WfSteerCtl` program.

6.3 Utilization

The ultimate goal of this work is to contribute with user-steered workflows in HPC. As discussed, MATTOSO *et al.* [23] explain that there are at least six aspects that need to be considered for this: interactive analysis, monitoring, user-steered adaptation, notification, interface for interaction, and computing model. In this work, we mostly focus on the first three, considering user-steered adaptation as the core of user steering and the one we mostly contribute with. As most related contributions to putting the human in the loop of HPC workflows [23, 26, 52], we focus on the efforts for designing and implementing concepts behind the backend enabling technology for user steering in an HPC workflow. More specifically, we contribute with allowing users to steer data reductions in scientific workflows online, focusing on providing a consistent execution within a data reduction, managing provenance data of user steering actions, and minimizing performance overheads in the HPC system. Enabling such features without jeopardizing performance in an HPC environment is hard. However, besides engineering the backend enabling technology, the interface for interaction is another important aspect to be considered.

Designing good interfaces requires usability studies to determine whether the interfaces are in fact good for the target user profile *i.e.*, computational scientists in our case. This would need a comprehensive user experience test to understand user behavior while interacting with their workflows, then we would develop interfaces based on the gathered design insights, and evaluate the usability. For a valid and comprehensive usability evaluation, we would need to ask multiple users to use the system and the modules developed, observe how they use, and interview them. However, the general context that this work is inserted in very complex. Our target-user profile is quite rare (compared with general business applications) and the results depend on the domain and the application in the domain. For example, if we want to measure the time a user takes to identify that a certain slice will not contribute to the final results and then remove it, a valid evaluation would require analyzing multiple users of the same application, in the same domain. Finding users of a same specific application is so rare that it makes a comprehensive usability test extremely hard. Also, many other questions need to be addressed. For instance, "does the user expertise in the domain-application interfere in the results? — perhaps the more experienced the user is, the faster she will find which slice to remove and the better

she understands the consequences of a reduction"; or "what if the tests were carried out on a different application for the same domain?"; or "what about a different domain?". Besides, using an HPC cluster requires scheduling. For an usability test, the analyzer needs to observe the user while she is interacting with the HPC workflow, and thus the analyzer's and the user's scheduling must match the HPC job scheduling time, for each user. In other words, a valid and comprehensive usability test would require observing users of the same application, of different domains, of different expertise levels, and matching scheduling times with the HPC cluster. Combining these requirements makes it very hard and out of the scope of this work, which focuses on enabling backend technology for steering an HPC workflow.

Therefore, instead of usability tests, in this section, we show how users can use `WfSteerCtl` in d-Chiron. Before developing, we interviewed a few computational scientists. We found that they are very used to command line interfaces and they frequently have to learn new computational tools. They often browse logs in terminals and follow the execution status of their simulations. To reduce data, they need to stop their workflow process, modify the input datasets by hand, and restart execution. For some users, this means resubmitting a job to an HPC cluster subject to scheduling. This may take a long time (even weeks). Therefore, developing a technology that allows them to reduce data online, based on provenance data analysis through structured queries (rather than Unix-like shell commands to filter multiple logs in the file system) is very desirable. Thus, we developed simple command line interfaces to enable them to steer monitoring queries and reduce data online. Since developing the best interface and analyzing its usability is out of the scope, the command line interfaces are our current best effort to make the technology usable. As we show in the experiments (Sec. 7.2.2), for validation purposes, a user is able to use the system, after a d-Chiron specialist trains him and provided support.

In d-Chiron, computer scientists who are experts in operating d-Chiron work closely with CSE users, the target-user of this work. However, computational scientists can operate d-Chiron and steer the running workflow. Before steering a workflow, the workflow has to be modeled. The user identifies the input and output data elements of each of those activities and gives a name to the dataset that contains those elements (following the dataflow-oriented approach — Sec. 2.2). When this is done, d-Chiron creates tables corresponding to those datasets, where each table column is an element produced or consumed by a data transformation. If needed, application-specific extractor scripts are built to collect output data to be stored in the workflow database [35].

The aspects of computational steering workflows tackled in this work are strongly related to how d-Chiron manages the data and the dataflow. It is all about the workflow database being populated online by the WMS. The workflow execution

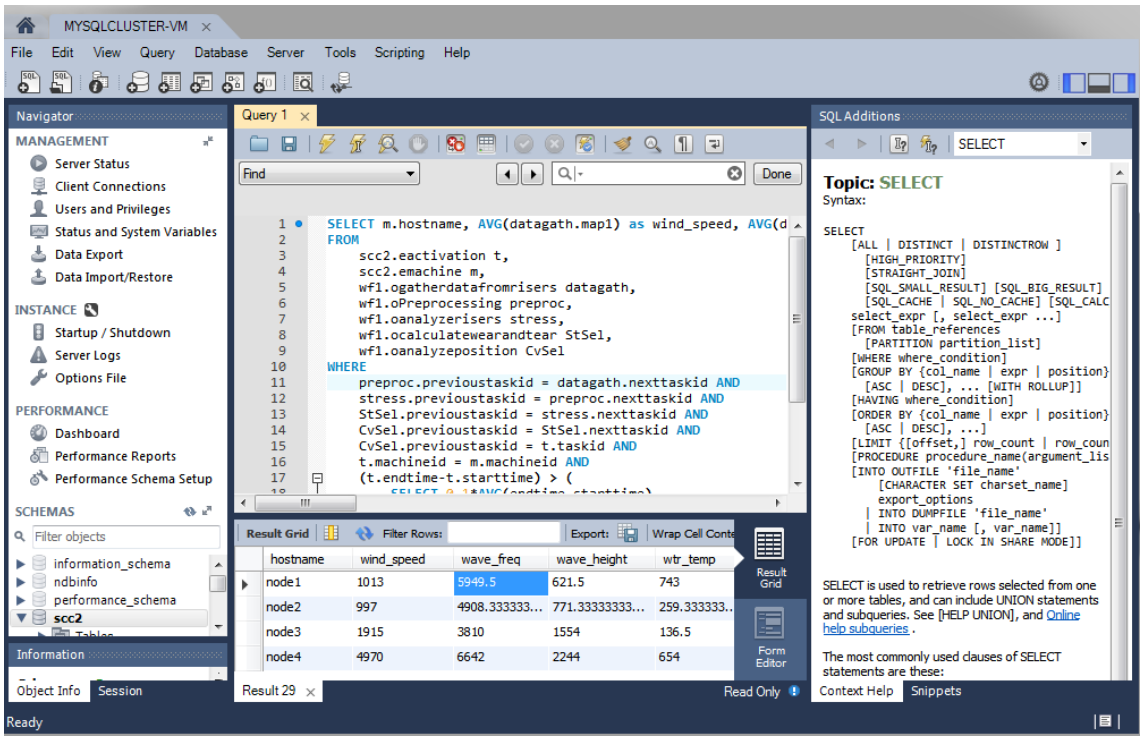


Figure 6.3: Using MySQL Workbench to query the workflow database at runtime.

plan depends on the data in this database (hence can be adapted at runtime) and the workflow database is available for user queries immediately after the workflow has started to run and data elements in the domain-dataflow are stored while they are generated. Then, they are linked to execution and provenance data in the workflow database to enable queries that integrate all these data.

Therefore, the main way users can interact with a workflow execution in d-Chiron is through query interfaces, generally provided by the DBMS, to query the workflow database. It helps to run the queries when the user understands the database schema that logically organizes data in d-Chiron. Computational scientists or engineers can work with computer scientists (d-Chiron specialists in this case) so they can build complex analytical SQL queries to interactively analyze the dataflow. From our experience, computational scientists do not take much time to learn how to write simple queries to a relational database and they later learn how to write complex analytical queries on their own. d-Chiron uses MySQL Cluster to manage its workflow database. MySQL users are accustomed to using MySQL Workbench as a visual interface to the DBMS. They can see the relational database schema, build and run SQL queries, and get their tabular results in the interface as the workflow runs. Figure 6.3 shows how MySQL Workbench can be used to write Q5 (described as natural language in Table 2.2), which integrates domain, provenance and execution data in a same query. More queries with their natural language descriptions are on GitHub [66] and in the Appendix A.

However, such user interactivity does not need to use SQL queries only. Some users prefer graphical user interfaces, so there are many other ways to interact with a DBMS: graphical interfaces with drag and drop boxes to help building queries, Natural Language to Database solutions to translate regular English sentences into SQL queries or dashboards that plot results from a query. In this work, discussing the usability of interfaces is not the focus. However, we show how users currently use command line interfaces for the `WfSteerCtl` program.

For adaptation, a user uses `WfSteerCtl` program (Figure 6.3) to inform who is going to interact (this information is stored in the workflow database for provenance) and passes a configuration file that contains information about the workflow, the HPC cluster, and the DBMS connection settings. After that, a user can run as many dataflow steering commands as necessary informing the input dataset I_{DS} in the workflow that will be reduced and the C criteria to select the slice (operands from *Cut* Definition 4.4). These steering action data are stored in the workflow database for provenance. The output messages allow the user to understand what is happening after a command line is issued. In particular, after a *Cut* action, the output message informs the user of the number of data elements that were removed from the dataset to be processed. For more complex analyses on the consequences of those reductions, users can query the workflow database using the tables introduced in this work to verify, for example, if there were files and their sizes to quantify the number of bytes that were not processed. We show these analyses in the experiment section (Sec. 7.2.2).

```
1 $> WfSteerCtl --user="Peter"
2     Output: Next workflow interactions will be issued by user Peter.
3 $> WfSteerCtl --cut --dataset="opreprocessing" --criteria="wind_speed <
      12.0 and wave_freq > 2.0"
4     Output: 177 data elements were cut off from OPREPROCESSING dataset.
5 $> WfSteerCtl --cut --dataset="opreprocessing" --criteria="wind_speed <
      11.3 and wave_freq > 1.8"
6     Output: 55 data elements were cut off from OPREPROCESSING dataset.
```

Listing 6.2: `WfSteerCtl` command line interface for data reduction.

For the Monitor Manager service (Figure 6.2), a user runs a command to start it as a background service on any cluster node that has access to the DBMS, usually the same node from which the WMS execution was launched. Then, users can add monitoring queries at any time. The monitoring query results are also properly stored in the workflow database, as they are generated. Dashboard graphic visualization applications can query these results to deliver better data visualization for the user.

```
1  $> WfSteerCtl --monitor --start
2      Output: Ready to accept new monitoring queries.
3  $> WfSteerCtl --monitor --add --mq="`cat q1.sql`" --label="q1" --
       interval=30
4      Output: Monitoring query q1 will be executed every 30 seconds.
5  $> WfSteerCtl --monitor --add --mq="`cat q2.sql`" --label="q2" --
       interval=20
6      Output: Monitoring query q2 will be executed every 20 seconds.
7  $> WfSteerCtl --monitor --update --label="q2" --interval=5
8     Output: Monitoring query q2 was updated. It will be executed every 5
        seconds.
9  $> vi q1.sql
10 $> WfSteerCtl --monitor --update --label="q1" --mq="`cat q1.sql`"
11     Output: Monitoring query q1 was updated.
```

Listing 6.3: `WfSteerCtl` command line interface for monitoring.

In this example, after the user starts the monitoring service (line 1), two monitoring queries are added with intervals 30 and 20 seconds, respectively (lines 3 and 5). The user wrote the queries in text files (`q1.sql` and `q2.sql`), which are loaded in the `WfSteerCtl --monitor --add` commands, using `cat` Unix command. Those query files are only to facilitate the command lines and they are not a requirement. A user could write the query string directly in the command line. After some time, the user decides to decrease the time interval in the monitoring of query with label "q2" by issuing the command at line 7. In line 9, the user decides to modify a specific query aspect (*e.g.*, increase the result limit) by editing the query text file and in line 10 he modifies the monitoring query. As for the management of steering actions, these user interactions are properly stored in the workflow database for provenance.

In the next chapter, we present the experimental evaluation of WfSteer.

Chapter 7

Experimental Evaluation

In this chapter, we design and conduct the experiments to support the validation of our approach, WfSteer, to address the problem and hence the validation of this thesis's hypothesis. Our hypothesis encompasses WfSteer's instantiation both for workflow scripts and for WMS, and within each instantiation we further distinguished between allowing for tracking steering actions and keeping the execution overhead low. This chapter follows this separation. Section 7.1 presents the validation of WfSteer for workflow scripts, showing a qualitative analysis on allowing for tracking steering actions and a quantitative evaluation on the execution overhead, using real-world use cases in the O&G industry. Analogously, Section 7.2 has the validation of WfSteer's instantiation into a WMS, following this same organization.

7.1 Managing Steering Action Data in Workflow Scripts

In this section, we present the experiments to aid the validation of one of the instantiations of WfSteer, DfAdapter, in a real-world workflow script, libMesh-sedimentation. DfAdapter has been introduced in Chapter 5 and libMesh-sedimentation has been briefly introduced in Section 2.3.1. We show how users can monitor and understand, at runtime, the impact of their steering actions by relating steering action data with provenance, domain, and execution data, then we evaluate the added overhead in a workflow script. We begin by providing implementation details of how DfAdapter is coupled with libMesh-sedimentation (Sec. 7.1.1), afterwards we present steering action data analysis in libMesh-sedimentation workflow (Sec. 7.1.2), and conclude with an overhead evaluation (Sec. 7.1.3).

7.1.1 Use case: Computational Fluid Dynamics in Geoscience with libMesh-sedimentation

libMesh-sedimentation provides a real and rich case for parameter tuning for the following reasons. First, it is a CSE application that requires HPC to run a simulation with over 70 parameters, which may be modified by the user for better performance and accuracy of results [27]. Second, as this simulation may last for weeks, the user does several tunings and there is no tracking for them. Third, there is a strong potential for richer online data analyses with steering action data by correlating the steering data to domain-specific values (mainly QoIs) and other data in the workflow database.

To use DfAdapter in libMesh-sedimentation, we follow the utilization guide described in Section 5.4. The first step is modeling libMesh-sedimentation simulation as a workflow and identifying analysis and adaptation points. Workflow data are captured by DfAnalyzer. Application-specific data are modeled as new tables of the relational database schema for the workflow database (Fig. 5.3). The main input dataset that the user adapts is the input for the loop evaluation data transformation, named `I_Iteration_Params`, which contains input parameters for the numerical solvers. The users specify parameters in a setup configuration file. The workflow script checks, at every time step, if any modification has been made to this file. If a modification occurred, the parameters are redefined according to the new values. That is, libMesh-sedimentation implements a file-based checks approach for the adapter service (Sec. 5.2). Modifications in this file happen within an implementation of the Adapter API. The implementation receives parameters and new values and modifies the file according to the inputs. The last step is to insert steering action data capture API calls in the adaptation points. In libMesh-sedimentation code, it is inserted immediately after the parameters are reloaded when there is a modification in the configuration file. Finally, when the user steers, DfAdapter captures provenance, domain and steering action data every time it detects user steering actions. Figure 7.1 shows the instrumentation of libMesh-sedimentation with workflow data capture calls (using DfAnalyzer's API — Sec. 5.3) and steering action data capture calls.

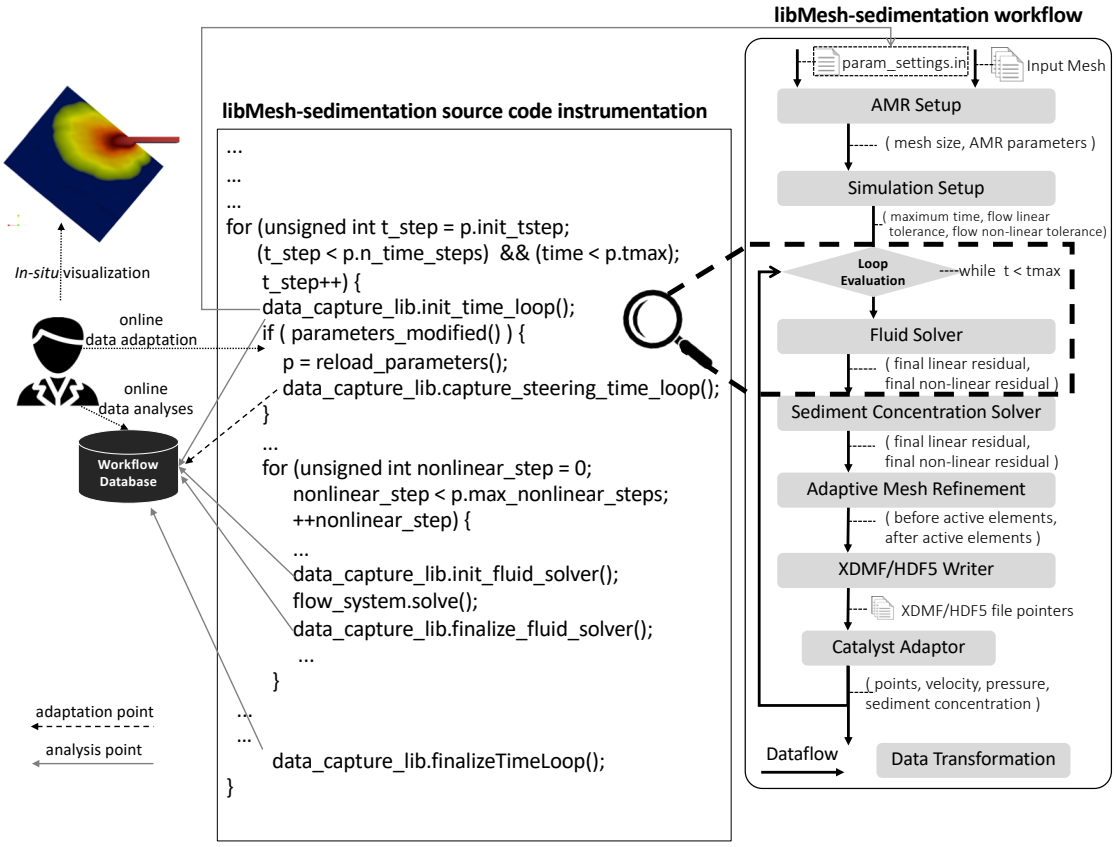


Figure 7.1: libMesh-sedimentation workflow script code with added API calls along with its workflow representation.

HPC Environment and Deployment. The experiments in this section were conducted on Lobo Carneirno cluster[1], an SGI ICE X with 252 nodes, each with a 24-core processor and 64 GB RAM, summing 6,048 cores and 16 TB RAM. The nodes are interconnected via FDR InfiniBand and share a Lustre file system with 500 TB. The Data Management services and MonetDB are deployed on a separate node in the cluster, different from the ones used by the main computational process for libMesh-sedimentation. libMesh-sedimentation is implemented in C++ and its code with instrumentation for analysis and steering is available on GitHub [130] along with with DfAdapter's code [127].

7.1.2 User Steering Action Data Analysis

Small-scale case

The small-scale experiment is used by scientists as a benchmark to evaluate sedimentation solvers. It simulates the laboratory test carried out by DE ROOIJ and DALZIEL [131] with a lock-exchange configuration. The objective of this experiment

[1]http://www.nacad.ufrj.br/recursos/sgiicex

is to show the data analytical potential of our solution, how we record structured parameter-tunings, and how users can query the steering action data to enhance their analyses.

The computational setup used in this test case consists of a plane channel with dimensions $20 * 2$ filled with sediments in suspension and clear fluid at rest. In the laboratory, a lock-gate is used to separate the fluids before the beginning of the experiment. When the gate is removed, a mutual intrusion flow develops in which the particle-laden front travels along the bottom to the right. In this simulation, the lock-gate is located at $x = 0.75$. The non-dimensional parameters used are $Grashof\ number = 5.0e{-}6$, $Schmidt\ number = 1.0$, and $Settling\ velocity = 0.02$. Adaptive mesh refinement is used to track the interface between sediments concentration and clear water. Figure 7.2 shows the concentration of sediments in suspension and the adapted mesh at simulation time $t = 10$.

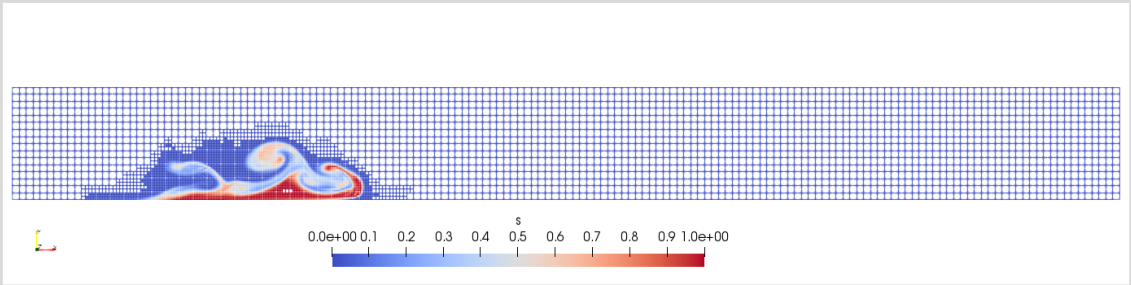


Figure 7.2: 2D visualization of the tank and the concentration of sediments. This figure was generated at simulation time $t = 10$.

In this simulation, the user is interested in analyzing possible performance gains when the number of nonlinear and linear (in this case, GMRES) iterations is tuned at runtime. Specific fine-tunings on different input parameters may impact the solvers and hence the simulation time considerably. During the execution, the user submits analytical queries. Based on the analyses of nonlinear and GMRES iterations, the user decides to fine-tune the solver's parameters. In total, the user chooses to do six fine-tunings in 10 hours of simulation. Figure 7.3 shows a query that tracks the steering action. The query lists the parameters tuned by a user (say, Bob), correlated to the time steps. By running this query, other researchers are aware that Bob adapted this workflow execution six times. The times and values are well-structured and recorded in the workflow database.

Query: List all user tunings correlating with time step.

This query does a join on tables: `ParameterTuning`, `ParameterTuned`, `InfluencedDataElement`, and `Attribute`, filtering by tunings made by 'Bob'. The result is:

Parameter Tuning	t_step	Parameter Tuned	Old Val	New Val
1	1401	flow_initial_linear_solver_tolerance	1e-8	1e-6
2	1474	minimum_linear_solver_tolerance	1e-8	1e-6
3	1484	flow_initial_linear_solver_tolerance	1e-6	1e-4
4	1755	max_linear_iterations	500	300
5	10061	amr/c_fraction	0.01	0.05
6	10128	max_linear_iterations	300	200

Figure 7.3: Query analyzing the track of the steering actions.

Figure 7.4 shows the query results of the average values of strategic quantities ten iterations before and after each of the fine-tunings. The results include nonlinear and linear (GMRES) iterations, which are output values of the solver, and the number of finite elements, which is an output of the mesh refinement process and depends on other inputs of the solver. This query shows an integration of provenance, domain, and execution data, and the new steering action data we introduced.

Query: Average of domain values (QoIs) and simulation time estimate 10 iterations before and after the tunings.

This query does a join on tables `ParameterTuning`, `ParameterTuned`, `Attribute`, `InfluencedDataElement`, `O_Sedimentation_Solver`, `O_Fluid_Solver`, `Task`, and `Performance`. It also does an average on the output values of `O_Sedimentation_Solver` and `O_Fluid_Solver`, and on `Execution Time` in `Performance` table. The result is:

Parameter Tuning	Avg Time (s) Bef	Avg Time (s) Aft	Avg nonlin. Bef	Avg nonlin Aft	Avg gmres Bef	Avg gmres Aft	Avg Elems Bef	Avg Elems Aft
1	17.3	18.5	3.8	3.9	2.03e3	2e3	5.32e3	5.38e3
2	16.9	18.1	4.1	4.3	2.05e3	2.03e3	5.44e3	5.41e3
3	17.4	13.2	4.2	4.3	2.02e3	1.54e3	5.45e3	5.43e3
4	12.7	9.6	3.9	4.2	1.49e3	1.01e3	5.51e3	5.49e3
5	14.4	14.8	4.3	4.0	1.06e3	1.01e3	6.28e3	5.58e3
6	15.6	11.2	4.05	4.1	647	445	5.72e3	5.62e3

Figure 7.4: Query integrating execution, domain, provenance, and steering action data.

The results in Figure 7.4 (we highlight the main findings) show that the Tunes #3, #4, and #6 impacted the average elapsed time and the average number of GMRES iterations, which are of high interest to the user. Tune #5 barely changed the other values but reduced the number of mesh elements by about 11.15%. This reduction is important because when there are too many elements, out-of-memory errors may happen (see the large-scale case next). In Figure 7.5, we plot the evolution of these variables over time and annotate the tunings (Tune #1 to Tune #6) so the user can evaluate the adaptations.

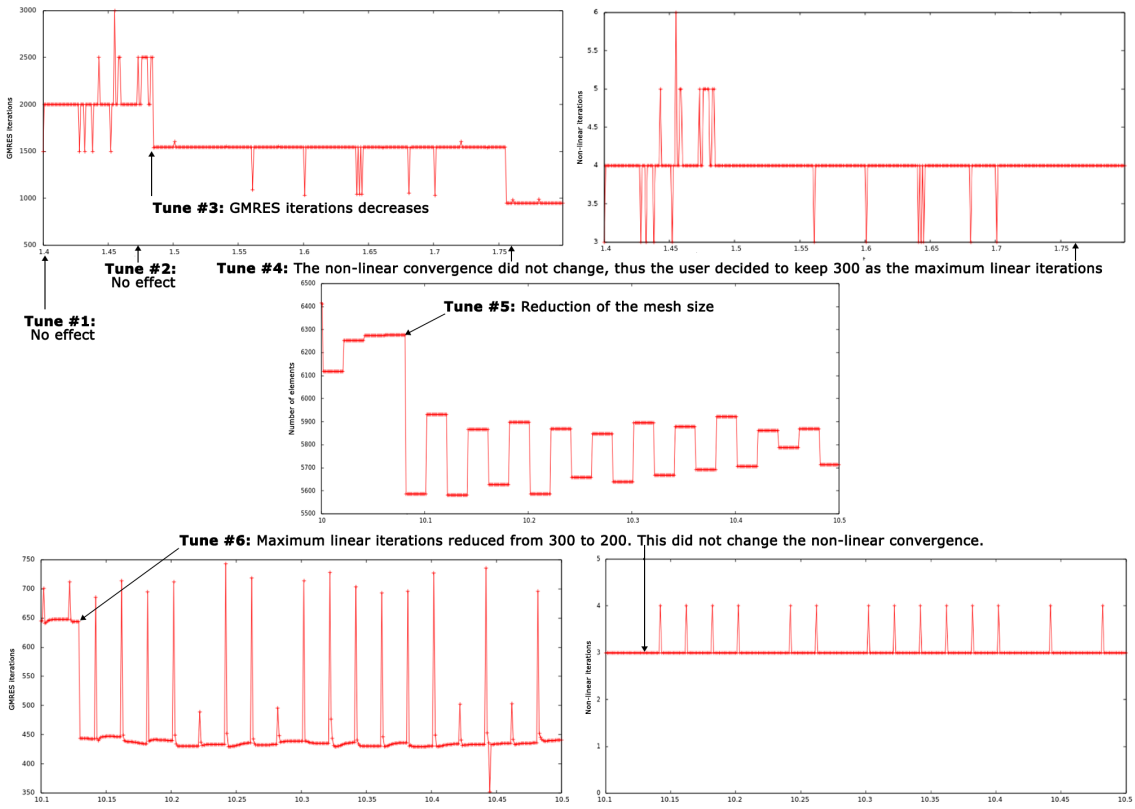


Figure 7.5: Plots of monitoring queries for number of GMRES iterations, non-linear iterations, and mesh elements over time. We highlight the tune actions.

Based on the online analyses of user steering action data, the user decides whether new tunes are needed. Moreover, considering a scenario where another research team analyzes the provenance of the results, the team sees the abrupt changes in the results and can correlate these results with Bob's steering actions through SQL queries in the workflow database. They can check if sudden changes are related to one of the adaptations Bob did. Thus, they will have a better understanding of the results.

Large-scale case

In a large-scale experiment, using 480 cores of Lobo Carneiro cluster, the user sets up the libMesh-sedimentation workflow with a simulation of the deposition of sediments carried by a turbidity current over a real experimental channel. A mixture of sediments is continuously injected into a channel that deposits sediments in the tank. The tank has $length = 135$, $width = 40$, and $height = 50$ (dimensionless units).

The dimensionless simulation parameters are $Settling\ velocity = 5.36e{-}6$, $Grashof\ number = 3.42e7$, $Schmidt\ number = 1.0$, and fixed $time\ step = 0.01$. It uses a 3D simulation with a spatial discretization using an initial unstructured mesh with 1.2 million tetrahedra. AMR/C is employed and three levels of uniform

refinement are applied before the time loop. The user specifies input parameter values for the sedimentation solver (*i.e.*, linear and non-linear tolerances, the maximum number of linear iterations, tolerances for AMR/C error estimation and refinement and coarsening fractions) aiming at attaining a high-fidelity simulation. One strategic simulation data that quantifies such level of detail is the number of elements obtained in the mesh refinement data transformation (the second one in the time loop). Although a large number of elements in the mesh means a high level of detail, it also means more memory and time consumed by the simulation. Depending on the parameter values specified for the solver, the simulation may run out of memory. Thus, the user does not know beforehand which range of parameters is best for a good level of detail with acceptable memory consumption.

To support the user in following the evolution of strategic values, we use our monitoring approach by setting up queries to the workflow database at specified time intervals (each simulation's time step). One query shows linear and nonlinear iterations, residual norms, and the number of elements in the mesh at each time step. Additionally, ParaView Catalyst is set up to plot 3D visualizations of the channel and the sediment deposits over time. Then, the user sees, for example, that the number of elements generated by the AMR/C is close to a maximum preset number of elements. At that rate, the simulation may crash, running out of memory. The user knows that by changing some of the solver parameters, the number of elements tends to decrease. Thus, the user issues a command to adapt the solver parameters and DfAdapter automatically tracks and registers this tuning.

In Figure 7.6, we show the plot of the monitoring query for the number of elements. We see how the number is increasing when the user decided to fine-tune the input parameters online aiming at reducing the number of elements. This action prevented the simulation to result in an out-of-memory error, which would interrupt the simulation, requiring offline tunings and job resubmission to the HPC machine.

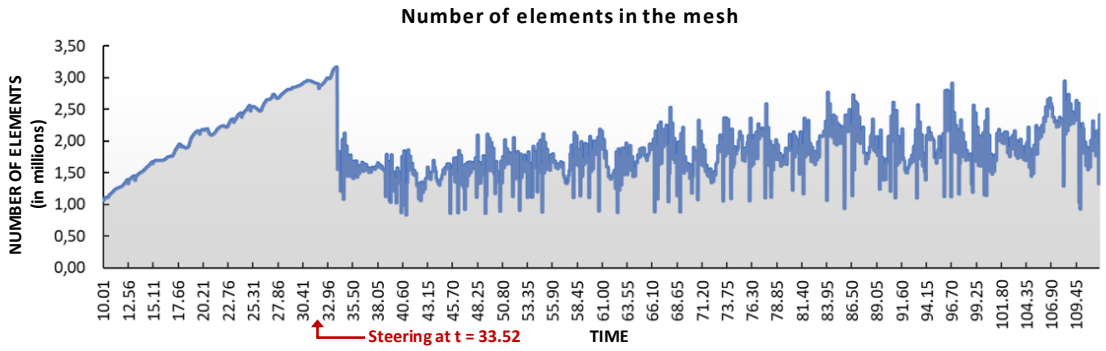


Figure 7.6: Plot of monitoring query showing number of elements over time.

In Figure 7.7, we show the 3D visualizations and the evolution of the strategic

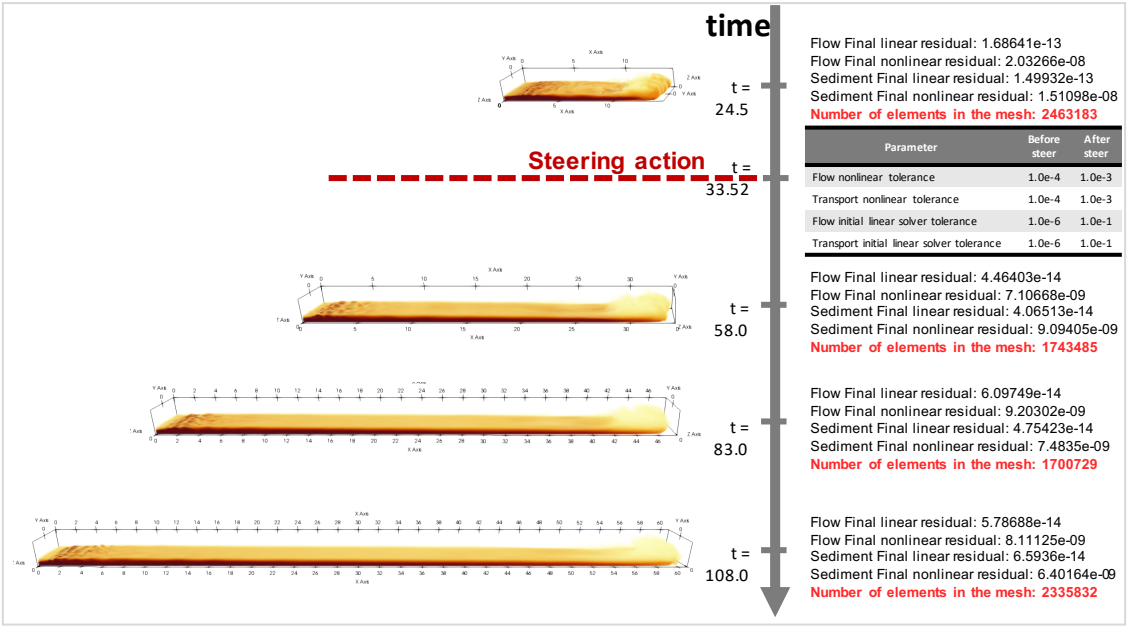


Figure 7.7: Snapshots of 3D visualization of the tanks and the sediments over time. Steering action occurs at $t = 33.53$ and user steering data are recorded.

values and how the sediments flow in the channel over time. Then, the user can run analytical queries to analyze the consequences of the fine-tunings, like the queries in Figures 7.3 and 7.4.

In Table 7.1, we show a small excerpt of these results, where we can see that the simulation time is cut down to 17 days, thanks to the fine-tunings made by the user. If we consider the average solver time by iteration before the fine-tunings, the simulation time would be approximately 27 days, *i.e.*, a reduction of 37%. The ability our approach gives to the users for them to have a detailed understanding of their steering actions and the consequences of their actions (*e.g.*, a reduction of about 10 days in the total execution time) improves the users' awareness, putting them in the loop of their simulations. These findings contribute to validating that WfSteer allows for tracking steering actions in workflow scripts.

Table 7.1: Summary of results of parameter tunings.

	Before	**After**	**Reduction**
Avg. Solver Time by iteration	3.82 min	2.21 min	42.14%
Avg. Number of elements	2.4e6	1.7e6	29.24%
Total execution time	(expected) $\sim$ 27 days	(real) $\sim$ 17 days	37%

7.1.3 Overhead Evaluation

We use the concepts and equations presented in Section 5.5 to evaluate the overhead added to execute DfAdapter coupled with libMesh-sedimentation workflow. In this experiment, the added overhead is caused by workflow data capture, raw data extraction, adaptation capabilities, and steering action data capture. Results are in Table 7.2. To obtain them, we first calculate each overhead component per task applying the Equations 5.1 to 5.5 using DfAdapter's logging data joining with tasks' performance data in the workflow database. Finally, we sum each contribution to the overall computational time as in Equation 5.6.

Table 7.2: The added overhead in the analysis and adaptation points account for less than 1%; data extraction account for 1.49%.

		Total CPU time (s)	**Total time (%)**
	Application computation $comp(Df)$	1,407,967.18	98.18%
Analysis	**Analysis points** $anl_{point}(Df)$	4,259.18	0.3%
	Data extraction $ext(Df)$	21,367.60	1.49%
Adaptation	**Adaptation point** $adp_{point}(Df)$	473.24	0.03%
	Action $action(Df)$	2.44	1.75e-5%
	Total $c(Df)$	1,434,069.64	100%

For analysis, workflow data capture overhead (analysis points) account for 0.3% caused by preparing the tuples to be sent to the Data Management services. Since data management services and the database system run in a separate computing resource and sending provenance data to be stored occurs asynchronously, the data capture overhead account only for preparing tuples to be sent. This represents a very low overhead, in the order of few milliseconds per task. libMesh-sedimentation workflow has an adaptation point at the beginning of the time loop iteration. Raw data extractors, provided by DfAnalyzer, extract convergence values from raw data files written as XDMF/HDF5 so the user can monitor and detect possible misbehavior of nonlinear and linear solvers. Extracting data values from raw data files to store in a database for analysis is done synchronously. Depending on the amount of data and how the raw data extractor is implemented, overhead may not be negligible. Here, these raw data extractions account for 1.49% of the total computation time. For adaptation points, since libMesh-sedimentation uses a file-based checks implementation, it verifies if a file has been modified at each new time iteration. This file verification is synchronous because the workflow script must verify if a change has happened before it can continue. In total, this check at each new iteration adds

0.03% overhead. When a steering action happens, the internal data structure that contains the solver parameters is reloaded and steering data are captured and sent to the Data Management services. Since the user steered 6 times during the execution of this workflow, the overhead for steering action tracking is close to 0%.

Besides, as a CSE application, in libMesh-sedimentation, tasks last for seconds-long on average (Figure 7.4) and the distributed CPUs spend significantly more time computing the application tasks than computing our data capture operations. Therefore, considering approximately 17 days of total execution time (about 1.4e6 seconds as shown in Figure 7.4), workflow data capture and steering action data capture together account for less than 1% overhead, whereas summing with raw data extractions, the total overhead is less than 2%.

Such reduced overhead is due to our system design principles related to asynchronicity and to the fact that the most costly data tracking operations, which give the structure and data relationships, occur in the data management services running on a separate node in the HPC machine. Anyhow, any overhead caused by any WfSteer implementation is greatly compensated by the benefits it makes to the user. For example, allowing for tracking the adaptations benefited reproducibility, validation, and interpretation. Also, observing at runtime that the adaptation reduced the execution time in ten days is relevant for further online tunings and result analyses. These findings contribute to validating that WfSteer allows for managing steering action with low execution overhead in workflow scripts.

7.2 Managing Steering Action Data in a WMS

In this section, we present the experiments to aid the validation of one of the implementations of WfSteer, data reduction using d-Chiron WMS, introduced in Chapter 6. For these experiments, we use the Ultra-deepwaters's Risers Fatigue Analysis workflow as the real-world application 2.3.2. We show how users can run data analysis and understand, at runtime, the impact of their steering actions in a WMS. We present the case for data reduction in the Risers workflow (Sec. 7.2.1), then we show steering action data analysis in this workflow (Sec. 7.2.2), and conclude with an overhead evaluation (Sec. 7.2.3)

7.2.1 Use case: Ultra-deepwaters' Risers Fatigue Analysis

As described before (Sec. 2.3.2), the Risers Fatigue Analysis workflow has seven data transformations. Except for the last one, they generate result data (both raw data files and some other domain-specific data values), which are consumed by the subsequent data transformations. These intermediate data need to be analyzed

during workflow execution. More importantly, depending on a specific range of data values for an output result data (*e.g.*, fatigue life value), there may be a specific combination of input data (*e.g.*, environmental conditions) that are more or less important during an interval of time within the workflow execution. The specific range is frequently hard to determine and requires a domain expert to analyze partial data during execution. For example, an input data element for DT_2 is a file that contains a large matrix of data values, composed of thousands of rows and dozens of columns. Each column contains data for an environmental condition and each row has data collected for a given time instant. Each row can be processed in parallel and the domain application needs to consume and produce other data files (on average, about 14 MB consumed and 6 MB produced per processed input data element). After many analyses online, the user finds that, for waves greater than 38 m with a frequency less than 1 Hz, riser fatigue will never happen. Thus, within the entire matrix, any input data element that contains this specific uninteresting range does not need to be processed. Therefore, by reducing the input dataset, the overall data processed and generated are reduced and thus the overall execution time.

HPC Environment and Deployment The experiments in this section were conducted on Grid5000[2], using a cluster with 39 computing nodes, containing 24 cores each (summing 936 cores). Every node has two AMD Opteron 1.7 GHz 12-core processors, 48 GB RAM, and 250GB of local disk. All nodes are connected via Gigabit Ethernet and access a shared storage of 10 TB. d-Chiron was executed with MySQL Cluster 7.4.9 as its in-memory distributed DBMS. The code to execute d-Chiron and setup files are available on GitHub [66].

7.2.2 User Steering Action Data Analysis

Running case

Let us consider the following synthetic scenario based on a real case in the O&G industry. A user, say Peter, is an offshore computational engineer, expert in riser analysis, and learned how to set up monitoring, analyze d-Chiron's workflow database, and use d-Chiron's `WfSteerCtl` program to steer the workflow. In Peter's project, the $Design\ Fatigue\ Factor$ is set to 3 and $service\ life$ is set to 20 years, meaning that fatigue life must be at least 60 years. Peter is only interested in analyzing risers with low fatigue life values as they are critical and might need repair or replacement. During workflow execution, it would be interesting if Peter could inform the WMS which input values would lead to low risk of fatigue so they could be removed. However, it is hard to determine the specific range of values *i.e.*, the slice, to be cut off. For this, Peter first needs to understand the pattern of input values associated with

[2]https://www.grid5000.fr

low risk of fatigue life values. In the workflow, the final value of fatigue life is calculated in the data transformation DT_6, but input values are obtained as the output of the data transformation DT_1, gathered from raw input files. Keeping provenance is essential to associate data from DT_1 with data from DT_6.

To understand which input values are leading to high fatigue life values, Peter monitors the generated data online. For simplicity, we consider *wind speed*, which is only one out of the many environmental condition parameter values captured in DT_1 to serve as input for the data transformation DT_2. Peter knows that wind speed has a strong correlation with fatigue life in risers. He expects that with low speed winds, there is a lower risk of an accident.

When the workflow execution starts, the Monitor Manager service is initialized. Then, Peter adds two monitoring queries: q_1 shows the average of the 10 greatest values of fatigue life calculated in the last 30 s of workflow execution, setting $\Delta t_1 = 30$ s; and q_2 shows the average wind speed associated with the 10 greatest values of fatigue life calculated in the last 30 s, also setting the query interval $\Delta t_2 = 30$ s. We recall from Table 2.1 that q_1 is similar to Q1, but only considering data processed in the last 30 seconds. q_1 and q_2 queries are added to the `Monitoring_Query` table in the database.

Peter monitors the results using the `Monitoring_Query` table. These results can be a data source for a monitoring tool that plots dashboards dynamically, refreshed according to the query intervals. After gaining insights from the results and understanding patterns, he can start cutting the undesired values for wind speed. The monitoring query results qr_{1t} and qr_{2t} for the queries q_1 and q_2, as well as when the user reduced the data, are plotted along the workflow elapsed time, as shown in Figure 7.8. It shows qr_{1t} (Fatigue life) in gray line with square markers and qr_{2t} (Wind speed) in black line with triangle markers. These markers determine when the monitoring query occurred.

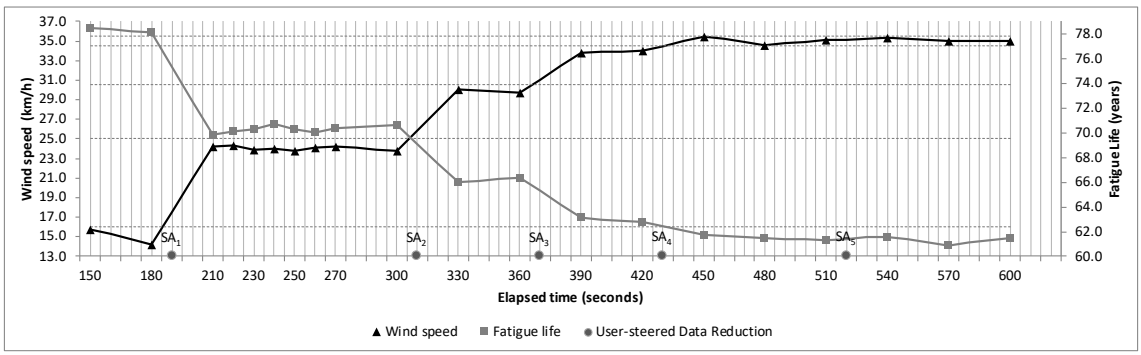


Figure 7.8: Analyzing the impact of user-steered data reduction comparing Wind Speed (input) with Fatigue life.

The workflow execution starts at $t = 0$ s, but only after approximately 150 s,

the first output results from DT_6 start to be generated. From the first results, at $t = 150$ s and $t = 180$ s, Peter checks that when wind speed is less than 16 km/h (see horizontal dashed line in *wind speed* = 16 in Figure 7.8), the results lead to the largest fatigue life values. Since risers with large fatigue life values are not interesting in this analysis, he decides, at $t = 190$ s, to remove all input data elements that contain wind speed less than 16 km/h. For this, the first user steering action SA_1 is issued with a command line to the `WfSteerCtl` program. User steering actions are represented with gray circles in the horizontal axis (*Elapsed time*). The time a user issued an interactive query is stored in `User_Query` table.

The next marker after SA_1 happens at $t = 210$ s. Comparing with the previous monitoring mark, at $t = 180$ s, we can observe that this Peter's action SA_1 increases the minimum wind speed values to be considered from 14.2 km/h to 24.1 km/h. Also, we observe a significant decrease in the slope of the largest values for fatigue life (10.6% lower). This means that the removal of these input data containing wind speed less than 16 km/h made the WMS not process data containing low wind speed values, which would lead to larger fatigue life results.

Then, monitoring continues, but that slope decrease in the fatigue life after $t = 180$ calls Peter's attention. To obtain a finer detail of what is happening, he decides to adjust monitoring settings, the monitoring interval times Δt_1 and Δt_2 in this case, at runtime. He reduces them to 10 s to get monitoring feedback more frequently. We can observe that for both lines qr_{1t} and qr_{2t}, the markers become more frequent during $t = [220, 270]$ s. This is because monitoring is registered at every 10 s. Although we show monitoring correlations between wind speed and fatigue life, other monitoring correlations could also be analyzed and users can add, remove or adjust monitoring queries at any time during execution. After verifying that the results are reasonable, he decides to adjust the monitoring setting to increase back the monitoring query intervals for both queries to 30 s after $t = 270$ s. Then he observes that since SA_1, wind speed less than 25 km/h are leading to large fatigue life values. Then, at $t = 310$ s, he calls `Steer` again to issue SA_2 hat removes input data for wind speed < 25 km/h. The next markers after SA_2 shows that this steering made the wind speed value associated with large fatigue life be at least 30.5 km/h and a decrease of 6.5% in large fatigue life values between $t = 300$ s and $t = 330$ s.

Similarly, Peter continues to monitor and steer the execution. He issues SA_3 at $t = 370$ s to remove input data with wind speed < 30.5 km/h, making a decrease of 4.9% in large fatigue life (comparing fatigue life in $t = 360$ s and $t = 390$ s). Then, he issues SA_4 at $t = 430$ s to remove input data with wind speed < 34.5, attaining a decrease of 1.7% in large fatigue life (comparing fatigue life in $t = 420$ s and $t = 450$ s). Despite this small decrease, he decides at $t = 520$ to further remove data, but with wind speed < 35.5 km/h. However, no decrease greater than 1% in the large

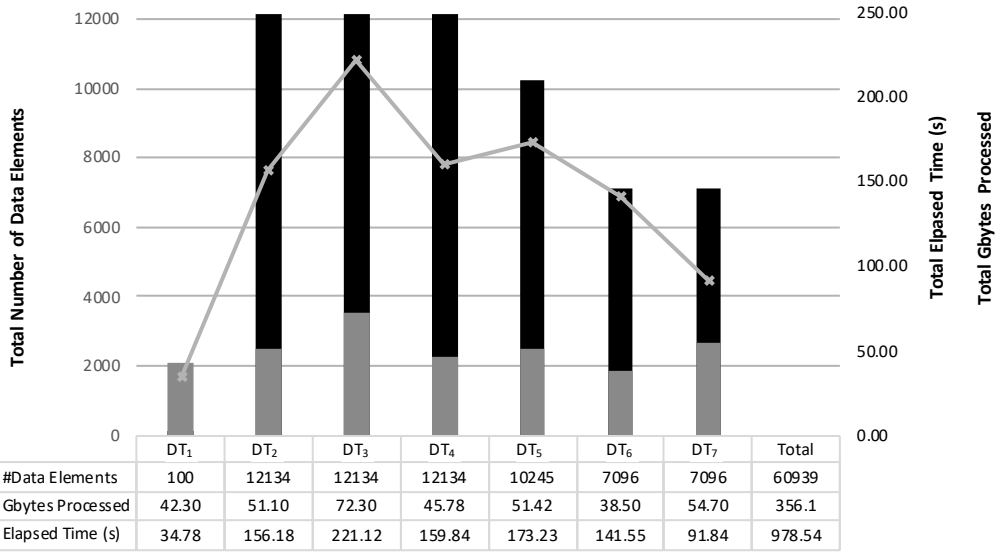
fatigue life values were registered after this last Peter's steering. Thus, he keeps analyzing the monitoring results, but does not remove input data anymore until the end of execution.

We store each interaction query, issued by the user, in the `User_Query` table and map (in table `Modified_Elements`) its rows with rows in `Dataset` and `Task` tables, to consistently manage the steering action data of which data elements were modified (in this case, removed) by each specific user steering action. Thus, managing steering action data helps to analyze how specific action impacted the results. Figure 7.8 shows that some specific action imply significant changes in lines' slopes (key output values for the user).

User-steered Data Reduction Analysis

Now we analyze how those previous steering actions impact the number of resources saved during the workflow execution. More specifically, we analyze three aspects: (i) the number of data elements reduced, (ii) the time that was saved due to the input data not processed, and (iii) the number of bytes of the raw data files that were not processed. For validation purposes, we count the resources saved as consequences of a data reduction. For this, we compare the executions with and without user steering. We run the same workflow and input datasets for both scenarios. The workflow execution with no steering processes all input data, including those containing wind speed values that lead to risers with low risk of fatigue, which are not valuable for Peter's analyses.

In Figure 7.9, we depict the three analyzed aspects per data transformation in the workflow. In other words, we count the total input data elements each data transformation consumes; the total number of gigabytes of data files processed in each data transformation; and the total time each data transformation took to complete. In total, considering all data transformations, the workflow with no steering processed 60,939 input data elements in parallel, 356 GB of domain data files and the overall execution time was 16.3 min running on the 936-cores cluster.



	DT_1	DT_2	DT_3	DT_4	DT_5	DT_6	DT_7	Total
Total #Data Elements	100	12134	12134	12134	10245	7096	7096	60939
Total Gbytes Processed	42.30	51.10	72.30	45.78	51.42	38.50	54.70	356.1
Total Elapsed Time (s)	34.78	156.18	221.12	159.84	173.23	141.55	91.84	978.54

Figure 7.9: Total data elements, gigabytes, and time consumed by data transformation with no user steering.

Then, we can compare these numbers with analogous numbers in the scenario with user-steered data reductions. Table 7.3 summarizes the user steering actions (*i.e.*, user-steered reductions) performed.

Table 7.3: Summary of the user-steered reductions (SA_1 – SA_5) with their user-defined slice criteria (*wind speed* is in km/h).

Steering Action	Issued time (s)	Slice criteria
SA_1	190	$wind\ speed < 16$
SA_2	310	$wind\ speed < 25$
SA_3	370	$wind\ speed < 30$
SA_4	430	$wind\ speed < 34.5$
SA_5	520	$wind\ speed < 35.5$

Figure 7.10 illustrates how each steering action SA_i affected the three analyzed aspects in each workflow data transformation: Figure 7.10 (A) shows the number of input data elements reduced, Figure 7.10 (B) shows the time saved, and Figure Figure 7.10 (C) shows the amount of gigabytes not processed due to data reduction. In the three charts, although the reductions happen in the dataset `I_Preprocessing` consumed by DT_2, we can see that they impact all subsequent data transformations (DT_1, which is a preceding data transformation, is not affected by the reductions). In particular, we can see that the first steering action, SA_1, alone causes a time reduction of 15%, *i.e.*, SA_1 makes the data transformation DT_3 complete 33 s faster, whereas without reductions DT_3 would take 221 s.

Figure 7.11 shows the summary of the impacts in the entire workflow by each

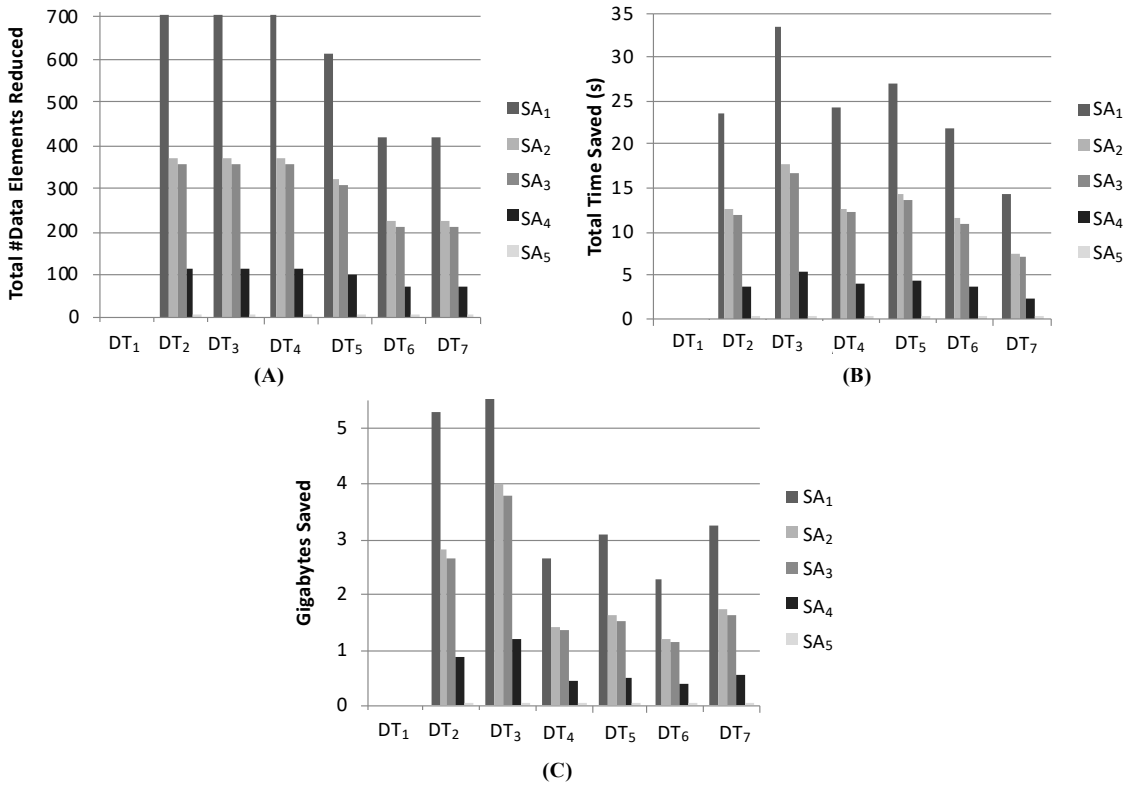


Figure 7.10: Reduced resources by data transformation caused by each user-steered reduction SA_i.

action (SA_i). Overall, the user-steered reductions in this experimental validation yield a reduction of 7,854 out of 60,939 data elements (12.89%), including elements in `I_Preprocessing` and elements in subsequent datasets as consequences of the reduction in `I_Preprocessing`. Also, the steering actions make the WMS not process about 51 GB out of 356 GB (14.9% of data files processing reduction) and the data transformations run faster, reducing in total 5.3 min out of 16.3 min (32.4% of total workflow execution time reduction) in the 936-cores cluster. In particular, we see that the first user-steered reduction, SA_1, represents 45% of the total amount of time saved, meaning that at the beginning, the user can identify a large slice of the input data that would not lead to interesting results, and we see that the last action SA_5 did not considerably affect execution. These results were obtained by querying the workflow database at the end of execution. By monitoring and interactively analyzing the workflow database online, users can have a better understanding of how their steering actions influenced the results of their computational experiments, especially they can explicitly inspect how their interactions reduced the used computational resources, thus contributing to validate that WfSteer allows for tracking steering actions in a WMS.

	SA_1	SA_2	SA_3	SA_4	SA_5	Total Amount Reduced	% Reduced
Total #Data Elements Reduced	3572	1889	1797	583	13	7854	12.89%
Total Gbytes Saved	24.11	12.75	12.13	3.94	0.09	53.02	14.89%
Total Time Saved (s)	144.18	76.24	72.53	23.53	0.52	317.00	32.40%

Figure 7.11: Summary of the user-steered reductions (SA_1 – SA_5) in the workflow.

7.2.3 Overhead Evaluation

Now we evaluate the overhead added to execute the experiment conducted using the Risers Fatigue Analysis workflow running on d-Chiron. d-Chiron implements the dataflow-oriented approach and manages domain, provenance, and execution data at runtime, enabling users to steer the workflow, but introduces overhead. Measuring the overhead caused by execution control management has been presented in past works [41, 56, 102]. Here, we discuss the overhead caused by user-steered data reduction and adaptive monitoring. First, when a data reduction happens, there are data movements in the workflow database, *i.e.*, some tasks and input data elements are updated or transferred from a table to another (Sec. 6.2.2). Time spent doing these updates in the database is significantly lower than the overall workflow execution time. Each steering action SA_i (Table 7.3) takes less than 1 second to finish, whereas the overall execution time of the workflow, after the reductions, is 661 seconds. Thus, those data movements' overhead are negligible. Second, our adaptive monitoring approach adds overhead and needs to be measured. Recall that every monitoring query $q_i \in QS$ is run by a thread at each Δt_i seconds (Sec. 6.2.2). Depending on the number $|QS|$ of threads and on the interval Δt_i there may be too many concurrent accesses to the workflow database, which may add overhead.

To measure this, we set up the `WfSteerCtl` in d-Chiron to run monitoring queries. The queries are variations of the queries Q1–Q7 (Tables 2.1 and 2.1). For example, in query Q2, we vary the curvature value. We also modify them to calculate only the results over the last Δt seconds, at each Δt seconds. To evaluate the overhead, we measure execution time without monitoring and then with monitoring, but varying the number $|QS|$ of queries and the interval Δt, which is considered the same for all queries in QS in this experiment. The experiments were repeated until the standard deviation of workflow elapsed times were less than 1%. The results are the median

of these times within the 1% margin. Figure 7.12 shows the results, where the gray portion represents the workflow execution time when no monitoring is used and the black portion represents the difference between the workflow execution time with and without monitoring (*i.e.*, the monitoring overhead).

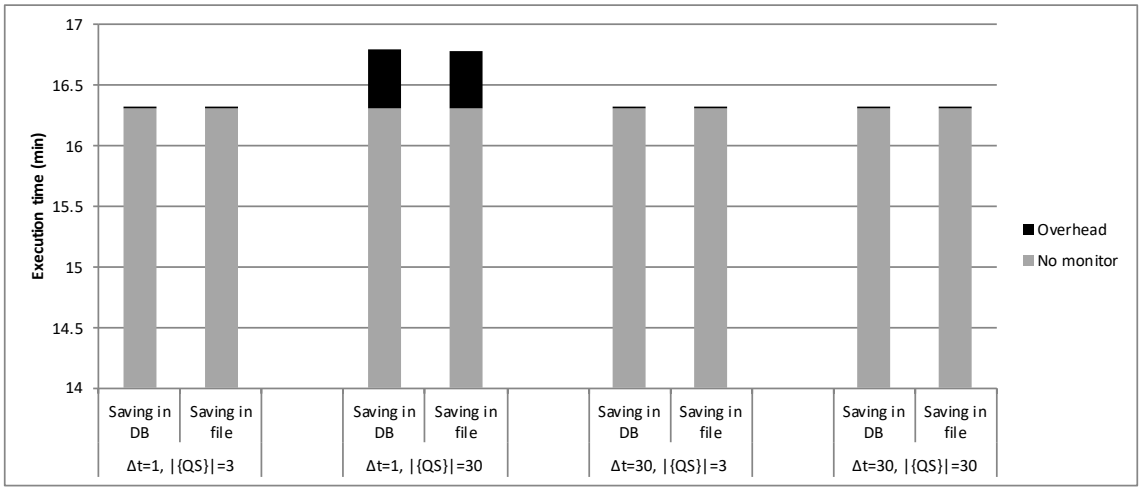


Figure 7.12: Results of adaptive monitoring overhead.

From these results, we observe that when the interval Δt is equal to 30 s, the overhead is negligible. For 1 s interval, the overhead is higher when the number of monitoring threads is also higher. This happens because three queries are executed in each time interval (see Listing 6.1), for each thread. In the scenarios with 30 threads, there are 120 queries in a single time interval Δt. In that case, if Δt is small, *e.g.*, $\Delta t = 1$, there are 120 queries being executed per second, just for the monitoring. In d-Chiron, the database that is queried by the monitors is also concurrently queried by the WMS engine, thus adding higher overhead. However, even in this specific scenario that shows higher overhead $|QS| = 30$ and $\Delta t = 1$, it is only 33 s or 3.19% higher than when no monitoring is used. Most of the real monitoring cases do not need such frequent (every second) updates. If 30 s is frequent enough for the user, the overhead is negligible, like in this test case. We also evaluated the same scenarios without storing monitoring results in the workflow database, but rather appending in CSV files, which is simpler. As Figure 7.12 shows, the results are nearly the same as in associated withase (saving in the workflow database or saving in CSV files). This suggests storing all monitoring results in the database at runtime, which enables users to submit powerful queries as the monitoring results are generated, with the workflow data in the database. This would not be possible with a solution that appends data to CSV files. Therefore, these findings contribute to validate that WfSteer allows for managing steering action with low execution overhead in a WMS.

Chapter 8

Conclusions

In this thesis, we aimed at supporting the workflow steering lifecycle to enable computational scientists and engineers (users) to understand their experiments when they are steering (monitoring, analyzing, adapting) large-scale workflows. To address the problem of managing steering action data in large-scale workflows, we proposed WfSteer, an approach that combines new provenance data management concepts, methods, and techniques that allow for tracking steering actions, which is critical to support the workflow steering lifecycle while adding negligible overhead. We designed and built two instantiations for WfSteer, *i.e.*, DfAdapter, for workflow scripts, and another in d-Chiron, for applications that can benefit from WMSs, which are the two typical ways users conduct their experiments. In this chapter, we share the lessons we learned (Sec. 8.1) and future work (Sec. 8.2).

8.1 Lessons Learned

To validate the proposed approach and this thesis's hypothesis itself, we split the hypothesis into two main aspects: one is to allow for tracking steering actions by means of managing steering action data in large-scale workflows and the other is to do this incurring negligible execution overhead. In this validation process, we learned several lessons that we share in this section.

For the first aspect, we began with a thorough analysis of the state-of-the-art on user steering in general, extending with an in-depth search in the literature extending to approaches with and without WMSs. We concluded that the literature lacks techniques and methods to support the management of steering action data, either for WMSs or for parallel computational simulations written as scripts. Second, after analyzing how users interact with their CSE experiments, we learned the data that characterizes the *user steering action data*, and how these data are generated when users perform *steering actions*, which are the two core concepts behind WfSteer. Finally, we proposed the notion of provenance of steering actions and a

provenance data diagram, called PROV-DfA, an extension of the W3C PROV standard, to model provenance of steering action data. We observed that the adherence to PROV and its dataflow extensions, namely PROV-Df, facilitated the modeling of important data that need to be related to the steering actions. Modeling data elements reduced in a *Cut* action and relating the iterations with parameters tuned in a *Tune* action in an iterative simulation were challenging modeling tasks. The formal characterization of user steering action in general and its specializations (*Cut* and *Tune*) contributed to the modeling. These findings are applicable either for workflow scripts and for WMSs. However, during our investigations, we found specific issues for the management of steering actions for each case that are worth mentioning separately.

For workflow scripts, we can include that by implementing DfAdapter, we faced system engineering difficulties that were addressed with the general conceptual architecture of DfAdapter. For instance, exposing the data the user wants to steer, so our data capture system can capture when the data change in case of an adaptation was solved by specifying how users can wrap adapters of their workflow so that the adapter is called, steering action data are captured, related, and stored. In the experiments carried out on Lobo Carneiro cluster running a real case in Computational Fluid Dynamics in Geoscience for the O&G industry, we observed that the ability our approach gives to the users for them to have a detailed understanding of their steering actions and the consequences of their actions improves the users' awareness, putting them in the loop of their simulations. For example, the user can explicitly identify that specific fine-tunings he made in the simulation solver made the workflow successfully finish without crash and with an approximate reduction of 37% (10 days) of the total time.

For the instance of WfSteer in a WMS, we observed that a data-centric WMS, like d-Chiron, enables dynamic adaptations of the data at runtime, but addressing consistency issues when users are adapting is a significant part of the implementation efforts when allowing for tracking steering action data in a WMS. We learned that our design principles that exploit a highly efficient distributed in-memory relational DBMS that allows for ACID, strong-consistent parallel transactions facilitated the implementation of the generic method we proposed for addressing consistency issues. Then, the implementation of WfSteer techniques in d-Chiron, such as the capture and relationships of steering action data using PROV-DfA concepts, enabled users to analyze the consequences of their actions at runtime. In the experiments carried out on a Grid5000 HPC machine with 936 CPU cores running a real use case, ultra-deepwaters' risers fatigue analysis workflow in the O&G industry, we observed that by monitoring and interactively analyzing the workflow database online, users could evaluate that because of their specific interactions they managed to save 14.9% of

physical disk space and 32.4% of total workflow execution time, hence significantly saving computing resources.

Now, for the second aspect of the hypothesis, that is, to keep the execution overhead low, we also separate by workflow scripts and WMS.

For workflow scripts, we learned that avoiding conflicts with the users' workflows is not trivial. To address this, we proposed system design principles, such as asynchronous API calls and leave heavy operations for provenance data management, such as the creation of provenance data relationships, to the Data Manager services, which are deployed on a separate computing node, hence avoiding competition with the user application. Additionally, allowing users to specify what should be monitored or adapted is far from trivial and to address this we proposed a methodology of use. The methodology foments the participation of the user in the specification of what should be captured for steering. We learned that this not only helps the data analysis, as the user knows what are being managed for steering and hence specifies only the relevant data, but it also reduces overhead because only the interesting data for steering are captured. In the experiments with DfAdapter, we observed that the added execution overhead caused by steering action data capture and adaptation is less than 1%, whereas the overhead for workflow data capture and raw data extraction sum about 1.5%, which is also negligible.

For the WMS instance, we learned that the exploitation of a highly efficient in-memory distributed DBMS both as the main source of analysis (managing the workflow database) and as the main data structure for parallel task scheduling improves the overall performance of the workflow execution. This includes the adaptive monitoring approach, which added less than 1% of overhead when the monitoring query intervals were 30 seconds and about 3% when it was 1 second, as shown in the experiments with d-Chiron.

Therefore, we conclude with these findings that the user steering action data management concepts and techniques introduced with WfSteer allowed users to track their actions online, enabling them to understand how their steering actions were influencing a running workflow.

8.2 Future Work

There are still several other open challenges to support the workflow steering lifecycle that could be addressed as future work. This thesis is mainly focused on data management aspects to support online data analysis. However, since the context involves a human in the loop, aspects traditionally studied by the Human-Computer Interaction scientific community could be addressed to enhance the steering support. For example, to interact with our implemented system, users need to run command

lines to call the adapter or run SQL queries to analyze the workflow data. Usable interfaces could potentially improve the users' engagement with their own workflow data. Similarly, *in-situ* data visualization techniques [50] could be proposed and extended, which would highly benefit the users in understanding their workflow data, combined with the steering action data managed by our solution.

In addition, our techniques and methods are heavily dependent on the users' knowledge to identify correlations between input and output data to determine which data are relevant and which subsets of input datasets are interesting or not. While this helps to maintain the workflow database with data that are more likely to be useful for analysis, such dependence on users may be considered a limitation. It can be time-consuming to specify the workflows and to a code. Thus, solutions to address this problem can be explored in the future. A possible direction is to use ML techniques to identify potential points of instrumentation in a workflow and then recommend a specification. The issue in this case is that the autonomously identified workflow may either lack details that are important for the users or may add recommend an excessively detailed workflow, which can compromise the understanding of the provenance data. Thus, further studies will require analysis of such trade-off.

Furthermore, in this thesis we have shown that enriching the workflow database with steering action data, jointly with provenance, execution, and domain data, enables a new class of interaction data analysis. In addition to improving the reliability on the generated workflow data and reproducibility, having such data potentially enables users to learn from their own adaptations: they may find that when they tune certain parameters to a given range of values, the convergence of the solver improves by a certain amount. We did not do any further user-based experiment to verify the potential of allowing users to learn from their own interactions.

Moreover, we believe that managing steering action data jointly with multiworkflow data [43] can highly enhance the analytical capabilities in even more complex settings that require several, distributed, heterogeneous workflow executions in CSE. Qualitative experiments of such a solution would investigate how the data from multiple, heterogeneous data stores with structured, unstructured, relational or NoSQL databases, can integrate with steering action data.

Also, the test cases we explored in this thesis were conducted on typical HPC machines. We expect that issues such as failure handling and budget constraints to be explored for tracking steering actions while users are steering workflows in a cloud cluster, as new steering actions hence their data can emerge. For this, the performance of cloud environments would need to be monitored and related to the actions, which can challenging and new user steering action data management techniques (*e.g.*, data modeling, efficient capture mechanisms in clouds) may be required.

Finally, all these steering action data allow for building AI-based systems that help users while they are steering simulations [125] or even do autonomous adaptations as they can extend their training database with provenance of adaptations.

Bibliography

[1] VERITAS, D. N. "Recommended practice: riser fatigue", *DNV-RP-F204*, 2010.

[2] GROTH, P., MOREAU, L. "W3C PROV - An Overview of the PROV Family of Documents". 2013. Available at: <https://www.w3.org/TR/prov-overview/>.

[3] MEIGNAN, D., KNUST, S., FRAYRET, J.-M., PESANT, G., GAUD, N. "A Review and Taxonomy of Interactive Optimization Methods in Operations Research", *ACM Trans. Interact. Intell. Syst.*, v. 5, n. 3, pp. 17:1–17:43, 9 2015. ISSN: 2160-6455. doi: 10.1145/2808234.

[4] PICKLES, S. M., HAINES, R., PINNING, R. L., PORTER, A. R. "A practical toolkit for computational steering", *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences*, v. 363, n. 1833, pp. 1843–1853, 8 2005. ISSN: 1364-503X. doi: 10.1098/rsta.2005.1611. PMID: 16099752.

[5] MULDER, J. D., VAN WIJK, J. J., VAN LIERE, R. "A Survey of Computational Steering Environments", *Future Generation Computer Systems*, v. 15, n. 1, pp. 119–129, 1999. ISSN: 0167-739X. doi: 10.1016/S0167-739X(98)00047-8.

[6] DANANI, B. K., D'AMORA, B. D. "Computational Steering for High Performance Computing: Applications on Blue Gene/Q System". In: *Symposium on High Performance Computing*, HPC '15, pp. 202–209, San Diego, CA, USA, 2015. Society for Computer Simulation International. ISBN: 978-1-5108-0101-1. Available at: <http://dl.acm.org/citation.cfm?id=2872599.2872624>.

[7] HAN, J., BROOKE, J. "Hybrid Computational Steering for Dynamic Data-driven Application Systems", *Procedia Computer Science*, v. 80, pp. 407–417, 1 2016. ISSN: 1877-0509. doi: 10.1016/j.procs.2016.05.341.

[8] LIERE, V. R., MULDER, J. D., WIJK, V. J. J. "Computational steering". In: *High-Performance Computing and Networking*, Lecture Notes

in Computer Science, pp. 696–702. International Conference on High-Performance Computing and Networking, Springer, Berlin, Heidelberg, 4 1996. ISBN: 978-3-540-61142-4. doi: 10.1007/3-540-61142-8_616. Available at: <https://link.springer.com/chapter/10.1007/3-540-61142-8_616>.

[9] VAN LIERE, R., D. MULDER, J., VAN WIJK, J. J. "Computational steering", *Future Generation Computer Systems*, v. 12, n. 5, pp. 441–450, 1997. ISSN: 0167-739X. doi: 10.1016/S0167-739X(96)00029-5.

[10] WIJK, J. J., LIERE, R. *An Environment for Computational Steering*. Technical report, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands, 1994.

[11] GU, W., EISENHAUER, G., KRAEMER, E., SCHWAN, K., STASKO, J., VETTER, J., MALLAVARUPU, N. "Falcon: on-line monitoring and steering of large-scale parallel programs". In: *Fifth Symposium on the Frontiers of Massively Parallel Computation, 1995. Proceedings. Frontiers '95*, pp. 422–429, 1995. doi: 10.1109/FMPC.1995.380483.

[12] GOEL, A., PHANOURIOU, C., KAMKE, F. A., RIBBENS, C. J., SHAFFER, C. A., WATSON, L. T. "WBCSim: A Prototype Problem Solving Environment for Wood-Based Composites Simulations", *Engineering with Computers*, v. 15, n. 2, pp. 198–210, 4 1999. ISSN: 0177-0667, 1435-5663. doi: 10.1007/s003660050014.

[13] SHU, J., WATSON, L. T., RAMAKRISHNAN, N., KAMKE, F. A., DESHPANDE, S. "Computational steering in the problem solving environment WBCSim", *Engineering Computations*, v. 28, n. 7, pp. 888–911, 10 2011. ISSN: 0264-4401. doi: 10.1108/02644401111165121.

[14] SHU, J., WATSON, L. T., ZOMBORI, B. G., KAMKE, F. A. "WBCSim: An Environment for Modeling Wood-based Composites Manufacture", *Eng. with Comput.*, v. 21, n. 4, pp. 259–271, 6 2006. ISSN: 0177-0667. doi: 10.1007/s00366-006-0010-5.

[15] GITHUB. "PROV-DfA GitHub Repository". 2019. Available at: <https://github.com/hpcdb/PROV-DfA>.

[16] RÜDE, U., WILLCOX, K., MCINNES, L. C., STERCK, H. D. "Research and Education in Computational Science and Engineering", *SIAM Review*, v. 60, n. 3, pp. 707–754, 2018. ISSN: 0036-1445. doi: 10.1137/16M1096840. Available at: <https://epubs.siam.org/doi/10.1137/16M1096840>.

[17] F. DA SILVA, R. "Pegasus and LIGO". In: *Pegasus Blog Post*, 2016. Available at: <https://pegasus.isi.edu/2016/02/23/pegasus-and-ligo/>.

[18] IBM RESEARCH. "AI and the Future of Oil: An AI Tool to Advise Geoscientists". In: *IBM Research Blog Post*, 2018. Available at: <https://www.ibm.com/blogs/research/2018/07/ai-future-oil/>.

[19] F. DA SILVA, R., FILGUEIRA, R., PIETRI, I., JIANG, M., SAKELLARIOU, R., DEELMAN, E. "A characterization of workflow management systems for extreme-scale applications", *Future Generation Computer Systems*, v. 75, pp. 228–238, 2017. ISSN: 0167739X. doi: 10.1016/j.future.2017.02.026.

[20] OGASAWARA, E., DIAS, J., OLIVEIRA, D., PORTO, F., VALDURIEZ, P., MATTOSO, M. "An algebraic approach for data-centric scientific workflows", *Proceedings of the VLDB Endowment*, v. 4, n. 12, pp. 1328–1339, 2011. ISSN: 2150-8097.

[21] DEELMAN, E., VAHI, K., JUVE, G., RYNGE, M., CALLAGHAN, S., MAECHLING, P. J., MAYANI, R., CHEN, W., FERREIRA DA SILVA, R., LIVNY, M., WENGER, K. "Pegasus, a workflow management system for science automation", *Future Generation Computer Systems*, v. 46, pp. 17–35, 5 2015. ISSN: 0167739X. doi: 10.1016/j.future.2014.10.008.

[22] MATTOSO, M., OCAÑA, K., HORTA, F., DIAS, J., OGASAWARA, E., SILVA, V., DE OLIVEIRA, D., COSTA, F., ARAÚJO, I. "User-steering of HPC workflows: state-of-the-art and future directions". In: *International Workshop on Scalable Workflow Execution Engines and Technologies (SWEET) co-located with the ACM Special Interest Group on Management of Data (SIGMOD)*, pp. 1–6. ACM Press, 2013. ISBN: 978-1-4503-2349-9. doi: 10.1145/2499896.2499900. Available at: <http://dl.acm.org/citation.cfm?doid=2499896.2499900>.

[23] MATTOSO, M., DIAS, J., OCAÑA, K. A., OGASAWARA, E., COSTA, F., HORTA, F., SILVA, V., DE OLIVEIRA, D. "Dynamic steering of HPC scientific workflows: a survey", *Future Generation Computer Systems*, v. 46, pp. 100–113, 2015. ISSN: 0167-739X. doi: 10.1016/j.future.2014.11.017. Available at: <http://dx.doi.org/10.1016/j.future.2014.11.017>.

[24] SILVA, V., NEVES, L., SOUZA, R., COUTINHO, A. L. G. A., DE OLIVEIRA, D., MATTOSO, M. "Adding domain data to code profiling tools to debug workflow parallel execution", *Future Generation Computer Systems*,

pp. 624–643, 2018. ISSN: 0167-739X. doi: 10.1016/j.future.2018.05.078. Available at: <https://doi.org/10.1016/j.future.2018.05.078>.

[25] SOUZA, R., SILVA, V., COUTINHO, A. L. G. A., VALDURIEZ, P., MATTOSO, M. "Data reduction in scientific workflows using provenance monitoring and user steering", *Future Generation Computer Systems*, v. online, pp. 1–34, 2017. ISSN: 0167-739X. doi: 10.1016/j.future.2017.11.028. Available at: <https://doi.org/10.1016/j.future.2017.11.028>.

[26] DIAS, J., GUERRA, G., ROCHINHA, F., COUTINHO, A. L. G. A., VALDURIEZ, P., MATTOSO, M. "Data-centric iteration in dynamic workflows", *Future Generation Computer Systems*, v. 46, n. C, pp. 114–126, 5 2015. ISSN: 0167-739X. doi: 10.1016/j.future.2014.10.021.

[27] CAMATA, J. J., SILVA, V., VALDURIEZ, P., MATTOSO, M., COUTINHO, A. L. G. A. "In situ visualization and data analysis for turbidity currents simulation", *Computers & Geosciences*, v. 110, pp. 23–31, 1 2018. ISSN: 0098-3004. doi: 10.1016/j.cageo.2017.09.013.

[28] HERSCHEL, M., DIESTELKÄMPER, R., BEN LAHMAR, H. "A survey on provenance: What for? What form? What from?" *The VLDB Journal*, v. 26, n. 6, pp. 881–906, 2017. ISSN: 1066-8888. doi: 10.1007/s00778-017-0486-1. Available at: <https://doi.org/10.1007/s00778-017-0486-1>.

[29] COSTA, F., SILVA, V., DE OLIVEIRA, D., OCAÑA, K., OGASAWARA, E., DIAS, J., MATTOSO, M. "Capturing and querying workflow runtime provenance with PROV: a practical approach". In: *Joint EDBT/ICDT 2013 Workshops*, EDBT '13, pp. 282–289. ACM, 2013. ISBN: 978-1-4503-1599-9. doi: 10.1145/2457317.2457365. Available at: <http://doi.acm.org/10.1145/2457317.2457365>.

[30] JAGADISH, H. V., GEHRKE, J., LABRINIDIS, A., PAPAKONSTANTINOU, Y., PATEL, J. M., RAMAKRISHNAN, R., SHAHABI, C. "Big data and its technical challenges", *Communications of the ACM*, v. 57, n. 7, pp. 86–94, 2014. ISSN: 00010782. doi: 10.1145/2611567. Available at: <http://dl.acm.org/citation.cfm?doid=2622628.2611567>.

[31] RUSSELL, S. J., NORVIG, P. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[32] DEELMAN, E., PETERKA, T., ALTINTAS, I., CAROTHERS, C. D., KLEESE VAN DAM, K., MORELAND, K., PARASHAR, M., RA-

MAKRISHNAN, L., TAUFER, M., VETTER, J. "The future of scientific workflows", *International Journal of HPC Applications*, v. 32, n. 1, pp. 159–175, 2017. ISSN: 1094-3420, 1741-2846. doi: 10.1177/1094342017704893. Available at: <http://journals.sagepub.com/doi/10.1177/1094342017704893>.

[33] ATKINSON, M., GESING, S., MONTAGNAT, J., TAYLOR, I. "Scientific workflows: Past, present and future", *Future Generation Computer Systems*, v. 75, pp. 216–227, 10 2017. ISSN: 0167-739X. doi: 10.1016/j.future.2017.05.041.

[34] NETTO, M. A. S., CALHEIROS, R. N., RODRIGUES, E. R., CUNHA, R. L. F., BUYYA, R. "HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges", *ACM Computing Surveys (CSUR)*, v. 51, n. 1, pp. 8:1–8:29, 1 2018. ISSN: 0360-0300. doi: 10.1145/3150224.

[35] SILVA, V., LEITE, J., CAMATA, J. J., DE OLIVEIRA, D., COUTINHO, A. L., VALDURIEZ, P., MATTOSO, M. "Raw data queries during data-intensive parallel workflow execution", *Future Generation Computer Systems*, v. 75, pp. 402–422, 10 2017. ISSN: 0167739X. doi: 10.1016/j.future.2017.01.016.

[36] SILVA, V., SOUZA, R., CAMATA, J., DE OLIVEIRA, D., VALDURIEZ, P., COUTINHO, A. L. G. A., MATTOSO, M. "Capturing Provenance for Runtime Data Analysis in Computational Science and Engineering Applications". In: *International Provenance and Annotation Workshop (IPAW)*, Lecture Notes in Computer Science (LNCS), pp. 183–187. Springer International Publishing, 2018. ISBN: 978-3-319-98379-0. Available at: <https://link.springer.com/chapter/10.1007/978-3-319-98379-0_15>.

[37] SOUZA, R., SILVA, V., CAMATA, J. J., COUTINHO, A. L. G. A., VALDURIEZ, P., MATTOSO, M. "Keeping track of user steering actions in dynamic workflows", *Future Generation Computer Systems*, v. 99, pp. 624–643, 2019. ISSN: 0167-739X. doi: 10.1016/j.future.2019.05.011. Available at: <https://hal-lirmm.ccsd.cnrs.fr/lirmm-02127456>.

[38] SILVA, V., DE OLIVEIRA, D., VALDURIEZ, P., MATTOSO, M. "DfAnalyzer: runtime dataflow analysis of scientific applications using provenance", *Proceedings of the VLDB Endowment*, v. 11, n. 12, pp. 2082–

2085, 2018. ISSN: 2150-8097. doi: 10.14778/3229863.3236265. Available at: <https://doi.org/10.14778/3229863.3236265>.

[39] GONCALVES, J., OLIVEIRA, D. D., OCANA, K., OGASAWARA, E., DIAS, J., MATTOSO, M. "Performance Analysis of Data Filtering in Scientific Workflows", *Journal of Information and Data Management*, v. 4, n. 1, pp. 17–26, 2013.

[40] SANTOS, I., DIAS, J., OLIVEIRA, D., OGASAWARA, E., OCANA, K., MATTOSO, M. "Runtime Dynamic Structural Changes of Scientific Workflows in Clouds". In: *IEEE/ACM International Workshop on Clouds and eScience Applications Management (CloudAM)*, pp. 417–422, Dresden, Germany, 2013.

[41] SOUZA, R., SILVA, V., OLIVEIRA, D., VALDURIEZ, P., LIMA, A. A. B., MATTOSO, M. "Parallel Execution of Workflows Driven by a Distributed Database Management System". In: *ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, pp. 1–3, 2015. Available at: <http://sc15.supercomputing.org/sites/all/themes/SC15images/tech_poster/tech_poster_pages/post284.html>.

[42] SOUZA, R., MATTOSO, M. "Provenance of Dynamic Adaptations in User-Steered Dataflows". In: *International Provenance and Annotation Workshop (IPAW)*, Lecture Notes in Computer Science (LNCS), pp. 16–29. Springer International Publishing, 2018. ISBN: 978-3-319-98379-0. Available at: <https://link.springer.com/chapter/10.1007/978-3-319-98379-0_2>.

[43] SOUZA, R., AZEVEDO, L., THIAGO, R., SOARES, E., NERY, M., NETTO, M., BRAZIL, E. V., CERQUEIRA, R., VALDURIEZ, P., MATTOSO, M. "Efficient Runtime Capture of Multiworkflow Data Using Provenance". In: *IEEE International Conference on e-Science (eScience)*, pp. 1–10, 2019. Available at: <https://hal-lirmm.ccsd.cnrs.fr/lirmm-02265932>.

[44] SOUZA, R., SILVA, V., COUTINHO, A., VALDURIEZ, P., MATTOSO, M. "Online Input Data Reduction in Scientific Workflows". In: *Workflows in Support of Large-Scale Science (WORKS) workshop co-located with the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, pp. 1–10, 2016. Available at: <https://hal.archives-ouvertes.fr/lirmm-01400538>.

[45] SOUZA, R., SILVA, V., CAMATA, J., COUTINHO, A., VALDURIEZ, P., MATTOSO, M. "Tracking of online parameter fine-tuning in scientific workflows". In: *Workflows in Support of Large-Scale Science (WORKS) workshop co-located with the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2017. Available at: <https://hal-lirmm.ccsd.cnrs.fr/lirmm-01620974>.

[46] SOUZA, R., NEVES, L., AZEREDO, L., LUIZ, R., TADY, E., CAVALIN, P., MATTOSO, M. "Towards a human-in-the-loop library for tracking hyperparameter tuning in deep learning development". In: *Latin American Data Science (LaDaS) workshop co-located with the Very Large Database (VLDB) conference*, pp. 84–87, 2018. Available at: <http://ceur-ws.org/Vol-2170/paper12.pdf>.

[47] SOUZA, R., SILVA, V., MIRANDA, P., LIMA, A. A. B., VALDURIEZ, P., MATTOSO, M. "Spark Scalability Analysis in a Scientific Workflow". In: *Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 288–293, 2017. Available at: <http://sbbd.org.br/2017/wp-content/uploads/sites/3/2018/02/p288-293.pdf>.

[48] SILVA, V., NEVES, L., SOUZA, R., COUTINHO, A., OLIVEIRA, D. D., MATTOSO, M. "Integrating Domain-data Steering with Code-profiling Tools to Debug Data-intensive Workflows". In: *Workflows in Support of Large-Scale Science (WORKS) workshop co-located with the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2016.

[49] BERNHOLDT, D., DUBEY, A., HEROUX, M., KLINVEX, A., MCINNES, L. C. "Improving Reproducibility Through Better Software Practices". 2017.

[50] BAUER, A. C., H., A., J., A., H., C., B., G., S., K., K., M., P., O., V., V., B., W., W., B. E. "In situ methods, infrastructures, and applications on high performance computing platforms", *Comp. G. Forum*, v. 35, n. 3, pp. 577–597, 7 2016. ISSN: 0167-7055. doi: 10.1111/cgf.12930.

[51] IKEDA, R., DAS SARMA, A., WIDOM, J. "Logical provenance in data-oriented workflows". In: *ICDE*, pp. 877–888, Finland, 2013. ISBN: 978-1-4673-4910-9. doi: 10.1109/ICDE.2013.6544882. Available at: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6544882>.

[52] NGUYEN, H. A., ABRAMSON, D., KIPOUROS, T., JANKE, A., GALLOWAY, G. "WorkWays: interacting with scientific workflows", *Concurrency and Computation: Practice and Experience*, v. 27, n. 16, pp. 4377–4397, 11 2015. ISSN: 15320626. doi: 10.1002/cpe.3525.

[53] FREIRE, J., KOOP, D., SANTOS, E., SILVA, C. T. "Provenance for Computational Tasks: A Survey", *Computing in Science & Engineering*, v. 10, n. 3, pp. 11–21, 2008. ISSN: 1521-9615. doi: 10.1109/MCSE.2008.79.

[54] DAVIDSON, S. B., FREIRE, J. "Provenance and Scientific Workflows: Challenges and Opportunities". In: *ACM International Conference on Management of Data (SIGMOD)*, SIGMOD '08, pp. 1345–1350, New York, NY, USA, 2008. ISBN: 978-1-60558-102-6. doi: 10.1145/1376616.1376772. Available at: <http://doi.acm.org/10.1145/1376616.1376772>.

[55] DE OLIVEIRA, D., SILVA, V., MATTOSO, M. "How much domain data should be in provenance databases?" In: *Workshop on Theory and Practice of Provenance (TaPP)*, Edinburgh, Scotland, 2015. USENIX Association.

[56] SOUZA, R., SILVA, V., NEVES, L., DE OLIVEIRA, D., MATTOSO, M. "Monitoramento de Desempenho usando Dados de Proveniência e de Domínio durante a Execução de Aplicações Científicas". In: *Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*. Sociedade Brasileira de Computação, 2015.

[57] BARBOSA, T., SOUZA, R., CRUZ, S., CAMPOS, M., COTTRELL, L. *Applying data warehousing and big data techniques to analyze internet performance*. Technical report, SLAC National Accelerator Lab., Menlo Park, CA (United States), 2016.

[58] STAMATOGIANNAKIS, M., KAZMI, H., SHARIF, H., VERMEULEN, R., GEHANI, A., BOS, H., GROTH, P. "Trade-Offs in Automatic Provenance Capture". In: *International Provenance and Annotation Workshop (IPAW)*, IPAW 2016, pp. 29–41, Berlin, Heidelberg, 2016. Springer-Verlag. ISBN: 978-3-319-40592-6. doi: 10.1007/978-3-319-40593-3_3. Available at: <https://doi.org/10.1007/978-3-319-40593-3_3>.

[59] MOREAU, L., BATLAJERY, B. V., HUYNH, T. D., MICHAELIDES, D., PACKER, H. "A Templating System to Generate Provenance", *IEEE Transactions on Software Engineering*, v. 44, n. 2, pp. 103–121, 2 2018. ISSN: 0098-5589, 1939-3520. doi: 10.1109/TSE.2017.2659745.

[60] PIMENTEL, J. F., MURTA, L., BRAGANHOLO, V., FREIRE, J. "noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts", *Proceedings of the VLDB Endowment*, v. 10, n. 12, pp. 1841–1844, 8 2017. ISSN: 21508097. doi: 10.14778/3137765.3137789.

[61] KIRK, B. S., PETERSON, J. W., STOGNER, R. H., CAREY, G. F. "libMesh : a C++ library for parallel adaptive mesh refinement/coarsening simulations", *Engineering with Computers*, v. 22, n. 3-4, pp. 237–254, 12 2006. ISSN: 0177-0667, 1435-5663. doi: 10.1007/s00366-006-0049-3.

[62] AYACHIT, U., BAUER, A., GEVECI, B., O'LEARY, P., MORELAND, K., FABIAN, N., MAULDIN, J. "ParaView Catalyst: enabling in situ data analysis and visualization". In: *In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization in Supercomputing workshops*, pp. 25–29. ACM Press, 2015. ISBN: 978-1-4503-4003-8. doi: 10.1145/2828612.2828624. Available at: <http://dl.acm.org/citation.cfm?doid=2828612.2828624>.

[63] OLIVEIRA, D., OGASAWARA, E., BAIAO, F., MATTOSO, M. "SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows". In: *International Conference on Cloud Computing*, CLOUD '10, pp. 378–385, Washington, DC, USA, 2010. International Conference on Cloud Computing. ISBN: 978-0-7695-4130-3. doi: 10.1109/CLOUD.2010.64. Available at: <http://dx.doi.org/10.1109/CLOUD.2010.64>.

[64] "The ProvONE data model for scientific workflow provenance". 2019. Available at: <http://vcvcomputing.com/provone/provone.html>.

[65] CASTRO, R., SOUZA, R., SILVA, V., OCAÑA, K., OLIVEIRA, D., MATTOSO, M. "Uma Abordagem para Publicação de Dados de Proveniência de Workflows Científicos na Web Semântica". In: *Simpósio Brasileiro de Banco de Dados*, 2015.

[66] GITHUB. "d-Chiron GitHub Repository". 2019. Available at: <http://github.com/hpcdb/d-Chiron>.

[67] XIAN, F. *Computational Steering Systems in Grid Computing Environments*. Technical report, University of Nebraska Lincoln, 2008. Available at: <http://citeseerx.ist.psu.edu/viewdoc/citations;jsessionid=D76C4340E38BA71A314970699B115114?doi=10.1.1.104.4599>.

[68] AYACHIT, U., BAUER, A., DUQUE, E. P. N., et al. "Performance Analysis, Design Considerations, and Applications of Extreme-scale in Situ Infrastructures". In: *ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, SC '16, pp. 79:1–79:12, Piscataway, NJ, USA, 2016. IEEE Press. ISBN: 978-1-4673-8815-3. Available at: <http://dl.acm.org/citation.cfm?id=3014904.3015010>.

[69] BOURHIS, P., DEUTCH, D., MOSKOVITCH, Y. "Analyzing data-centric applications: Why, what-if, and how-to". In: *International Conference on Data Engineering*, pp. 779–790, 2016. doi: 10.1109/ICDE.2016.7498289.

[70] JABLONOWSKI, D. J., BRUNER, J. D., BLISS, B., HABER, R. B. "VASE: The visualization and application steering environment". In: *ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, pp. 560–569. Supercomputing '93. Proceedings, 11 1993. doi: 10.1109/SUPERC.1993.1263505.

[71] PARKER, S. G., JOHNSON, C. R. "SCIRun: a scientific programming environment for computational steering". In: *ACM/IEEE Conference on Supercomputing*, pp. 52–71, San Diego, California, United States, 1995. ACM. ISBN: 0-89791-816-9. doi: 10.1145/224170.224354. Available at: <http://portal.acm.org/citation.cfm?id=224170.224354>.

[72] VAN LIERE, R., D. MULDER, J., VAN WIJK, J. J. "Computational steering", *Future Generation Computer Systems*, v. 12, n. 5, pp. 441–450, 1997. ISSN: 0167-739X. doi: 10.1016/S0167-739X(96)00029-5.

[73] VETTER, J., SCHWAN, K. "Techniques for high-performance computational steering", *IEEE Concurrency*, v. 7, n. 4, pp. 63–74, 10 1999. ISSN: 1092-3063. doi: 10.1109/4434.806980.

[74] KOHL, J. A., WILDE, T., BERNHOLDT, D. E. "Cumulvs: Interacting with High-Performance Scientific Simulations, for Visualization, Steering and Fault Tolerance", *The International Journal of High Performance Computing Applications*, v. 20, n. 2, pp. 255–285, 5 2006. ISSN: 1094-3420. doi: 10.1177/1094342006064502.

[75] RATHMAYER, S., LENKE, M. "A tool for on-line visualization and interactive steering of parallel HPC applications". In: *International Parallel Processing Symposium*, pp. 181–186. International Parallel Processing Symposium, 4 1997. doi: 10.1109/IPPS.1997.580882.

[76] EISENHAUER, G., SCHWAN, K. "An Object-based Infrastructure for Program Monitoring and Steering". In: *Symposium on Parallel and Distributed Tools (SIGMETRICS)*, SPDT '98, pp. 10–20, New York, NY, USA, 1998. ACM. ISBN: 978-1-58113-001-0. doi: 10.1145/281035.281037. Available at: <http://doi.acm.org/10.1145/281035.281037>.

[77] WOOD, J., BRODLIE, K., WALTON, J. "gViz-Visualization and Steering for the Grid". In: *e-Science All Hands Meeting*. Citeseer, 2003.

[78] MANN, V., MATOSSIAN, V., MURALIDHAR, R., PARASHAR, M. "DISCOVER: An environment for Web-based interaction and steering of high-performance scientific applications", *Concurrency and Computation: Practice and Experience*, v. 13, n. 8-9, pp. 737–754, 2001.

[79] GLASNER, C., HUGL, R., REITINGER, B., KRANZLMULLER, D., VOLKERT, J. "The Monitoring and Steering Environment". In: *Computational Science - ICCS 2001*, Lecture Notes in Computer Science, pp. 781–790. International Conference on Computational Science, Springer, Berlin, Heidelberg, 5 2001. ISBN: 978-3-540-42233-4. doi: 10.1007/3-540-45718-6_83. Available at: <https://link.springer.com/chapter/10.1007/3-540-45718-6_83>.

[80] BRODLIE, K., POON, A., WRIGHT, H., BRANKIN, L., BANECKI, G., GAY, A. "GRASPARC: A Problem Solving Environment Integrating Computation and Visualization". In: *Proceedings of the 4th Conference on Visualization '93*, VIS '93, pp. 102–109, Washington, DC, USA, 1993. IEEE Computer Society. ISBN: 978-0-8186-3940-1. Available at: <http://dl.acm.org/citation.cfm?id=949845.949868>.

[81] REED, D. A., ELFORD, C. L., MADHYASTHA, T. M., SMIRNI, E., LAMM, S. E. "The Next Frontier: Interactive and Closed Loop Performance Steering." In: *International Conference on Parallel Processing (ICPP) Workshops*, pp. 20–31, 1996.

[82] SWIFT, B., SORENSEN, A., GARDNER, H., DAVIS, P., DECYK, V. "Live Programming in scientific simulation", *Supercomputing Frontiers and Innovations: an International Journal*, v. 2, n. 4, pp. 4–15, 3 2015. ISSN: 2409-6008. doi: 10.14529/jsfi150401.

[83] GOODALE, T., ALLEN, G., LANFERMANN, G., MASSO, J., RADKE, T., SEIDEL, E., SHALF, J. "The Cactus Framework and Toolkit: Design and Applications". In: *Proceedings of the 5th International Confer-*

ence on High Performance Computing for Computational Science, VECPAR'02, pp. 197–227, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN: 978-3-540-00852-1. Available at: <http://dl.acm.org/citation.cfm?id=1766851.1766868>.

[84] ESNARD, A., RICHART, N., COULAUD, O. "A Steering Environment for Online Parallel Visualization of Legacy Parallel Simulations". In: *IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pp. 7–14. 2006 Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications, 10 2006. doi: 10.1109/DS-RT.2006.7.

[85] HAIMES, R. "Concurrent distributed visualization and solution steering". In: Ecer, A., Periaux, J., Satdfuka, N., Taylor, S. (Eds.), *Parallel Computational Fluid Dynamics 1995*, North-Holland, pp. 41–50, Amsterdam, 1996. ISBN: 978-0-444-82322-9. Available at: <https://www.sciencedirect.com/science/article/pii/B9780444823229500591>. DOI: 10.1016/B978-044482322-9/50059-1.

[86] KRESS, J., PUGMIRE, D., KLASKY, S., CHILDS, H. "Visualization and Analysis Requirements for in Situ Processing for a Large-scale Fusion Simulation Code". In: *Proceedings of the 2Nd Workshop on In Situ Infrastructures for Enabling Extreme-scale Analysis and Visualization*, ISAV '16, pp. 45–50, Piscataway, NJ, USA, 2016. IEEE Press. ISBN: 978-1-5090-3872-5. doi: 10.1109/ISAV.2016.14. Available at: <https://doi.org/10.1109/ISAV.2016.14>.

[87] FIGUEIRA, S., BUI, S. "CS_LITE: A lightweight computational steering system." In: *International Conference on Parallel and Distributed Computing and Networks*, pp. 1–6, 2004. Available at: <https://www.researchgate.net/publication/221352388_CS_LITE_A_lightweight_computational_steering_system>.

[88] PARASHAR, M., LEE, C. A. "Grid computing: introduction and overview", *Proceedings of the IEEE, special issue on grid computing*, v. 93, n. 3, pp. 479–484, 2005.

[89] RIBLER, R. L., VETTER, J. S., SIMITCI, H., REED, D. A. "Autopilot: Adaptive control of distributed applications". In: *International Symposium on High Performance Distributed Computing*, pp. 172–179. IEEE, 1998.

[90] YI, H., RASQUIN, M., FANG, J., BOLOTNOV, I. A. "In-situ visualization and computational steering for large-scale simulation of turbulent flows

in complex geometries". In: *IEEE International Conference on Big Data (Big Data)*, pp. 567–572. IEEE International Conference on Big Data (Big Data), 10 2014. doi: 10.1109/BigData.2014.7004275.

[91] MA, K.-L., WANG, C., YU, H., TIKHONOVA, A. "In-situ processing and visualization for ultrascale simulations", *Journal of Physics: Conference Series*, v. 78, n. 1, pp. 012043, 2007. ISSN: 1742-6596. doi: 10.1088/1742-6596/78/1/012043.

[92] MATKOVIC, K., GRACANIN, D., JELOVIC, M., CAO, Y. *Adaptive Interactive Multi-Resolution Computational Steering for Complex Engineering Systems*. The Eurographics Association, 2011. ISBN: 978-3-905673-82-1. Available at: <https://diglib.eg.org:443/handle/10.2312/PE.EuroVAST.EuroVA11.045-048>. DOI: http://dx.doi.org/10.2312/PE/EuroVAST/EuroVA11/045-048.

[93] BUTNARU, D. *Computational steering with reduced complexity*. Ph.D. Thesis, Technische Universitat Munchen, 2013.

[94] KNEZEVIC, J., FRISCH, J., MUNDANI, R.-P., RANK, E. "Interactive computing framework for engineering applications", *Journal of Computer Science*, v. 7, n. 5, pp. 591, 2011.

[95] WANG, I., TAYLOR, I., GOODALE, T., HARRISON, A., SHIELDS, M. "gridMonSteer: Generic Architecture for Monitoring and Steering Legacy Applications in Grid Environments". In: *e-Science All Hands Meeting*. e-Science All Hands Meeting, 2006. Available at: <https://www.researchgate.net/publication/237325712_gridMonSteer_Generic_Architecture_for_Monitoring_and_Steering_Legacy_Applications_in_Grid_Environments>.

[96] JAGADISH, H. V., GEHRKE, J., LABRINIDIS, A., PAPAKONSTANTINOU, Y., PATEL, J. M., RAMAKRISHNAN, R., SHAHABI, C. "Big data and its technical challenges", *Communications of the ACM*, v. 57, n. 7, pp. 86–94, 2014. ISSN: 00010782. doi: 10.1145/2611567.

[97] WOZNIAK, J. M., ARMSTRONG, T. G., WILDE, M., KATZ, D. S., LUSK, E., FOSTER, I. T. "Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing". In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 95–102. IEEE, 2013. ISBN: 978-0-7695-4996-5. doi: 10.1109/CCGrid.2013.99. Available at: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6546066>. 00008.

[98] DURO, F. R., BLAS, J. G., ISAILA, F., WOZNIAK, J. M., CARRETERO, J., ROSS, R. "Flexible Data-Aware Scheduling for Workflows over an In-memory Object Store". In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 321–324. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 5 2016. doi: 10.1109/CCGrid.2016.40.

[99] ABBOTT, B. P., ABBOTT, R., ABBOTT, T. D., et al. "GW170104: Observation of a 50-Solar-Mass Binary Black Hole Coalescence at Redshift 0.2", *Physical Review Letters*, v. 118, pp. 221101, 2017. doi: 10.1103/PhysRevLett.118.221101. Available at: <`https://link.aps.org/doi/10.1103/PhysRevLett.118.221101`>.

[100] GUNTER, D., DEELMAN, E., SAMAK, T., BROOKS, C., GOODE, M., JUVE, G., MEHTA, G., MORAES, P., SILVA, F., SWANY, M., VAHI, K. "Online workflow management and performance analysis with Stampede". In: *Proceedings of the 7th International Conference on Network and Service Management (CNSM)*, pp. 1–10, 10 2011.

[101] JAIN, A., ONG, S. P., CHEN, W., MEDASANI, B., QU, X., KOCHER, M., BRAFMAN, M., PETRETTO, G., RIGNANESE, G.-M., HAUTIER, G., GUNTER, D., PERSSON, K. A. "FireWorks: a dynamic workflow system designed for high-throughput applications", *Concurrency and Computation: Practice & Experience*, v. 27, n. 17, pp. 5037–5059, 2015. ISSN: 1532-0634. doi: 10.1002/cpe.3505.

[102] SOUZA, R. *Controlling the Parallel Execution of Workflows Relying on a Distributed Database*. MSc. Thesis, COPPE/Federal University of Rio de Janeiro, 2015.

[103] REYES, S., MUNOZ-CARO, C., NINO, A., SIRVENT, R., BADIA, R. M. "Monitoring and Steering Grid Applications with GRID Superscalar", *Future Generation Computer Systems*, v. 26, n. 4, pp. 645–653, 4 2010. ISSN: 0167-739X. doi: 10.1016/j.future.2009.12.002.

[104] MATKOVIC, K., GRACANIN, D., SPLECHTNA, R., JELOVIC, M., STEHNO, B., HAUSER, H., PURGATHOFER, W. "Visual analytics for complex engineering systems: Hybrid visual steering of simulation ensembles", *IEEE transactions on visualization and computer graphics*, v. 20, n. 12, pp. 1803–1812, 2014.

[105] CORDASCO, G., DE CHIARA, R., RAIA, F., SCARANO, V., SPAGNUOLO, C., VICIDOMINI, L. "Designing Computational Steering Fa-

cilities for Distributed Agent Based Simulations". In: *ACM Conference on Principles of Advanced Discrete Simulation (SIGSIM)*, SIGSIM PADS '13, pp. 385–390, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-1920-1. doi: 10.1145/2486092.2486147. Available at: <http://doi.acm.org/10.1145/2486092.2486147>.

[106] FOSTER, I., AINSWORTH, M., ALLEN, B., et al. "Computing just what you need: online data analysis and reduction at extreme scales". In: *European Conference on Parallel and Distributed Computing (Euro-Par)*, Lecture Notes in Computer Science, pp. 3–19. Euro-Par, Springer, Cham, 8 2017. ISBN: 978-3-319-64202-4. doi: 10.1007/978-3-319-64203-1_1. Available at: <https://link.springer.com/chapter/10.1007/978-3-319-64203-1_1>.

[107] TERRAZ, T., RIBES, A., FOURNIER, Y., IOOSS, B., RAFFIN, B. "Melissa: large scale in transit sensitivity analysis avoiding intermediate files". In: *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, p. 61. ACM, 2017.

[108] INGRAM, S., MUNZNER, T., IRVINE, V., TORY, M., BERGNER, S., MOLLER, T. "Dimstiller: Workflows for dimensional analysis and reduction". In: *IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pp. 3–10. IEEE, 2010.

[109] JIN, T., ZHANG, F., SUN, Q., BUI, H., PARASHAR, M., YU, H., KLASKY, S., PODHORSZKI, N., ABBASI, H. "Using cross-layer adaptations for dynamic data management in large scale coupled scientific workflows". In: *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12. ACM, 2013.

[110] GARCIA, M., DUQUE, J., BOULANGER, P., FIGUEROA, P. "Computational steering of CFD simulations using a grid computing environment", *International Journal on Interactive Design and Manufacturing (IJIDeM)*, v. 9, n. 3, pp. 235–245, 8 2015. ISSN: 1955-2513, 1955-2505. doi: 10.1007/s12008-014-0236-1.

[111] SPINUSO, A. *Active provenance for data intensive research*. Ph.D. Thesis, University of Edinburgh, 2018.

[112] ZHANG, Z., SPARKS, E. R., FRANKLIN, M. J. "Diagnosing Machine Learning Pipelines with Fine-grained Lineage". In: *International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '17, pp.

143–153, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4699-3. doi: 10.1145/3078597.3078603. Available at: <http://doi.acm.org/10.1145/3078597.3078603>.

[113] RE, C., AGRAWAL, D., BALAZINSKA, M., CAFARELLA, M., JORDAN, M., KRASKA, T., RAMAKRISHNAN, R. "Machine Learning and Databases: The Sound of Things to Come or a Cacophony of Hype?" In: *ACM International Conference on Management of Data (SIGMOD)*, SIGMOD '15, pp. 283–284, New York, NY, USA, 2015. ACM. ISBN: 978-1-4503-2758-9. doi: 10.1145/2723372.2742911. Available at: <http://doi.acm.org/10.1145/2723372.2742911>.

[114] XIN, D., MA, L., LIU, J., MACKE, S., SONG, S., PARAMESWARAN, A. "Accelerating Human-in-the-loop Machine Learning: Challenges and Opportunities", *Data Management for End-to-end Machine Learning (DEEM) Workshop co-located with the ACM Special Interest Group on Management of Data (SIGMOD)*, 2018. Available at: <http://arxiv.org/abs/1804.05892>.

[115] SOUZA, R., AZEVEDO, L., LOURENÇO, V., et al. "Provenance Data in the Machine Learning Lifecycle in Computational Science and Engineering". In: *Workflows in Support of Large-Scale Science (WORKS) workshop co-located with the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2019. Available at: <https://arxiv.org/abs/1910.04223>.

[116] LOFSTEAD, J. F., KLASKY, S., SCHWAN, K., PODHORSZKI, N., JIN, C. "Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS)". In: *Proceedings of the 6th International Workshop on Challenges of Large Applications in Distributed Environments (CLADE '08)*, pp. 15–24, New York, NY, USA, 2008. ACM Press. ISBN: 978-1-60558-156-9. doi: 10.1145/1383529.1383533. Available at: <http://portal.acm.org/citation.cfm?doid=1383529.1383533>.

[117] DREHER, M., PETERKA, T. *Decaf: Decoupled dataflows for in situ high-performance workflows*. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2017.

[118] ABRAMSON, D., ENTICOTT, C., ALTINAS, I. "Nimrod/K: towards massively parallel dynamic grid workflows". In: *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*,

pp. 1–11, Austin, Texas, USA, 2008. ISBN: 978-1-4244-2835-9. Available at: <http://portal.acm.org/citation.cfm?id=1413370.1413395>.

[119] VISTRAILS. "VisTrails". 2014. Available at: <http://www.vistrails.org/>.

[120] NGUYEN, H. A., ABRAMSON, D., KIPOROUS, T., JANKE, A., GALLOWAY, G. "WorkWays: interacting with scientific workflows". In: *Gateway Computing Environments Workshop*, GCE '14, pp. 21–24, Piscataway, NJ, USA, 2014. IEEE Press. ISBN: 978-1-4799-7030-8. doi: 10.1109/GCE.2014.6. Available at: <http://dx.doi.org/10.1109/GCE.2014.6>.

[121] HART, D., KRAEMER, E. "Consistency Considerations in the Interactive Steering of Computations", *International Journal of Parallel and Distributed Systems and Networks*, v. 2, pp. 171–179, 1999.

[122] "GitHub epository for RealityGrid Computational Steering Tools". 2019. Available at: <https://github.com/RealityGrid>.

[123] HAN, J., HAINES, R., SALHLI, A., BROOKE, J. M., D'AMORA, B., DANANI, B. "Virtual science on the move: Interactive access to simulations on supercomputers". In: *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, pp. 178–179. 2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, 6 2014. doi: 10.1109/ASAP.2014.6868654.

[124] DIAS, J., OGASAWARA, E., DE OLIVEIRA, D., PORTO, F., COUTINHO, A. L., MATTOSO, M. "Supporting Dynamic Parameter Sweep in Adaptive and User-steered Workflow". In: *Workflows in Support of Large-Scale Science (WORKS) workshop co-located with the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, WORKS '11, pp. 31–36, New York, NY, USA, 2011. ACM. ISBN: 978-1-4503-1100-7. doi: 10.1145/2110497.2110502. Available at: <http://doi.acm.org/10.1145/2110497.2110502>.

[125] SILVA, B., NETTO, M. A. S., CUNHA, R. L. F. "JobPruner: A machine learning assistant for exploring parameter spaces in HPC applications", *Future Generation Computer Systems*, v. 83, pp. 144–157, 6 2018. ISSN: 0167-739X. doi: 10.1016/j.future.2018.02.002.

[126] OCANA, K. A. C. S., OLIVEIRA, D. D., OGASAWARA, E., DAVILA, A. M. R., LIMA, A. A. B., MATTOSO, M. "SciPhy: A Cloud-Based Work-

flow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes". In: *Brazilian Symposium on Bioinformatics*, Lecture Notes in Computer Science, pp. 66–70. Springer, 2011. ISBN: 978-3-642-22824-7. Available at: <https://doi.org/10.1007/978-3-642-22825-4_9>.

[127] GITHUB. "DfAdapter GitHub Repository". 2019. Available at: <https://github.com/hpcdb/DfAdapter>.

[128] RAICU, I., FOSTER, I. T., ZHAO, Y. "Many-task computing for grids and supercomputers". In: *Workshop on Many-task Computing on Grids and Supercomputers (MTAGS)*, pp. 1–11. IEEE, 2008.

[129] OZSU, M. T., VALDURIEZ, P. *Principles of Distributed Database Systems*. 3 ed. New York, Springer, 2011.

[130] GITHUB. "libMesh-sedimentation Workflow GitHub Repository". 2019. Available at: <https://github.com/hpcdb/workflow-sedimentation>.

[131] DE ROOIJ, F., DALZIEL, S. B. "Time- and space-resolved measurements of deposition under turbidity currents". In: McCaffrey, W., Kneller, B., Peakall, J. (Eds.), *Particulate Gravity Currents*, Blackwell Publishing Ltd., pp. 207–215, Oxford, UK, 4 2001. ISBN: 978-1-4443-0427-5. Available at: <http://doi.wiley.com/10.1002/9781444304275.ch15>. DOI: 10.1002/9781444304275.ch15.

Appendix A

SQL Code for Analytical Queries

Domain dataflow provenance queries

Analytical Queries for Risers Fatigue Analysis. The relational data diagram is on GitHub [66].

Query 1. What is the average of the environmental conditions that are leading to the 10 greatest fatigue life values?

```
SELECT AVG(datagath.wind_speed), AVG(datagath.wave_freq), AVG(datagath.
    wave_height), AVG(datagath.wtr_temp), AVG(datagath.air_temp)
FROM
 dchiron.eactivation t,
 rfa.oDatagathering datagath,
 rfa.oPreprocessing preproc,
 rfa.oStressanalysis stress,
 rfa.oStressSriticalSaseSelection StSel,
 rfa.oCurvatureCriticalCaseSelection CvSel,
 rfa.oCalculateFatigueLife calcfatigue
WHERE
 datagath.previoustaskid = Idatagath.nexttaskid AND
 preproc.previoustaskid = datagath.nexttaskid AND
 stress.previoustaskid = preproc.nexttaskid AND
 StSel.previoustaskid = stress.nexttaskid AND
 CvSel.previoustaskid = StSel.nexttaskid AND
 calcfatigue.previoustaskid = CvSel.previoustaskid AND
 calcfatigue.fatigue_life IN (
  SELECT fatigue_life
  FROM rfa.calcfatigue
  ORDER BY fatigue_life DESC
  LIMIT 10
 )
```

Query 2. What are the float unit conditions that are leading to risers' curvature

lower than 800?

```
SELECT stress.fcurvature_val, datagath.funit_name, datagath.
    funit_status, datagath.funit_lat, datagath.funit_long, datagath.
    funit_type
 rfa.iDatagathering Idatagath,
 rfa.oDatagathering datagath,
 rfa.oPreprocessing preproc,
 rfa.oStressanalysis stress
WHERE
 datagath.previoustaskid = Idatagath.nexttaskid AND
 preproc.previoustaskid = datagath.nexttaskid AND
 stress.previoustaskid = preproc.nexttaskid AND
 stress.fcurvature_val < 800
ORDER BY stress.fcurvature_val
LIMIT 10
```

Query 3. What are the top 10 compressed data files (input for Activity 1) that contain original data that are leading to lowest fatigue life value (output from Activity 6)?

```
SELECT calcfatigue.fatigue_life, f.name, f.path, f.size, f.fieldname
 dchiron.eactivation t,
 rfa.iDatagathering Idatagath,
 rfa.oDatagathering datagath,
 rfa.oPreprocessing preproc,
 rfa.oStressanalysis stress,
 rfa.oStressSriticalSaseSelection StSel,
 rfa.oCurvatureCriticalCaseSelection CvSel,
 rfa.oCalculateFatigueLife calcfatigue
WHERE
 datagath.previoustaskid = Idatagath.nexttaskid AND
 preproc.previoustaskid = datagath.nexttaskid AND
 stress.previoustaskid = preproc.nexttaskid AND
 StSel.previoustaskid = stress.nexttaskid AND
 CvSel.previoustaskid = StSel.nexttaskid AND
 calcfatigue.previoustaskid = CvSel.previoustaskid AND
 calcfatigue.previoustaskid = t.taskid AND
 f.taskid =  t.taskid
ORDER BY calcfatigue.fatigue_life
LIMIT 10
```

Integrating domain dataflow and execution data queries

Query 4. What are the histograms and finite element meshes files related when computed fatigue life based on stress analysis is lower than 60?

```
SELECT stress.stress_fatigue_val, mesh.name, mesh.path, mesh.size, hist
    .name, hist.path, hist.size
FROM
  dchiron.eactivation t,
  dchiron.efile mesh,
  dchiron.efile hist,
  rfa.oDatagathering datagath,
  rfa.oPreprocessing preproc,
  rfa.oStressanalysis stress
WHERE
  preproc.previoustaskid = datagath.nexttaskid AND
  stress.previoustaskid = preproc.nexttaskid AND
  stress.previoustaskid = t.taskid AND
  stress.stress_fatigue_val < 60 AND
  mesh.taskid = t.taskid AND
  mesh.fieldname = 'ELEM_MESH' AND
  hist.taskid = t.taskid AND
  hist.fieldname = 'HISTOGRAM'
ORDER BY stress.stress_fatigue_val, mesh.fieldname, hist.fieldname
```

Query 5 Determine the average of each environmental conditions (output of Data Gathering – Activity 1) associated to the tasks that are taking execution time more than 1.5 the average of Curvature Critical Case Selection (Activity 5).

```
SELECT AVG(datagath.wind_speed), AVG(datagath.wave_freq), AVG(datagath.
    wave_height), AVG(datagath.wtr_temp), AVG(datagath.air_temp)
FROM
  dchiron.eactivation t,
  rfa.oDatagathering datagath,
  rfa.oPreprocessing preproc,
  rfa.oStressanalysis stress,
  rfa.oStressSriticalSaseSelection StSel,
  rfa.oCurvatureCriticalCaseSelection CvSel
WHERE
  preproc.previoustaskid = datagath.nexttaskid AND
  stress.previoustaskid = preproc.nexttaskid AND
  StSel.previoustaskid = stress.nexttaskid AND
  CvSel.previoustaskid = StSel.nexttaskid AND
  CvSel.previoustaskid = t.taskid AND
  (t.endtime-t.starttime) > (
   SELECT 1.5*AVG(endtime-starttime)
   FROM dchiron.eactivation
   WHERE tag = 'CurvatureCriticalCaseSelection'
  )
```

Query 6 Determine the finite element meshes files (output of Preprocessing – Activity 2) associated to the tasks that are finishing with error status.

```
SELECT f.name, f.path, f.size
FROM
  dchiron.emachine m,
  dchiron.eactivation t,
  dchiron.eactivity acti,
  dchiron.efile f
WHERE
  t.actid = acti.actid AND
  t.status = 'FINISHED_WITH_ERROR' AND
  f.taskid = t.taskid AND
  f.fieldname = 'ELEM_MESH' AND
  acti.tag = 'Preprocessing'
ORDER BY f.size DESC
```

Query 7. List the 5 computing nodes with the greatest number of Preprocessing activity tasks that are consuming tuples that contain wind speed values greater than 70 Km/h.

```
SELECT m.hostname, COUNT(*)
FROM
  dchiron.emachine m,
  dchiron.eactivation t,
  dchiron.eactivity acti,
  rfa.odatagathering datagath
WHERE
  m.machineid = t.machineid AND
  t.actid = acti.actid AND
  acti.tag = 'Preprocessing' AND
  datagath.nexttaskid = t.taskid AND
  datagath.windspeed > 70
GROUP BY m.hostname
ORDER BY 2 DESC
LIMIT 5
```

Appendix B

DfAdapter's steering action data tracker implementation

```
import sys
import os
import json
from time import gmtime, strftime
from subprocess import Popen, PIPE
import pymonetdb

from numpy import random

def get_steered_parameters_str(parameters_json):
    if os.path.isfile(parameters_json):
        with open(parameters_json) as data_file:
            steered_parameters = json.load(data_file)
    else:
        try:
            steered_parameters = json.loads(parameters_json)
        except:
            print "This is not a valid JSON: %s \n" % parameters_json
            sys.exit(-1)
    return json.dumps(steered_parameters)

if __name__ == "__main__":

    if len(sys.argv) < 4:
        print "Arguments: user dataset parameters_json [optional path to adapter script]"
        sys.exit(0)

    connection = pymonetdb.connect(username="monetdb", password="monetdb", hostname="localhost", database="dataflow_analyzer", autocommit=True)
```

```
    cursor = connection.cursor()


    user = sys.argv[1]
    dataset = sys.argv[2]
    parameters_json = sys.argv[3]
    if len(sys.argv) > 4:
        adapter_implementation_script = sys.argv[4]
    else:
        adapter_implementation_script = "src/adapters/redis_based_adapter.py"

    parameters_str = get_steered_parameters_str(parameters_json)
    cmd = ["python", adapter_implementation_script, parameters_str]
    process = Popen(cmd, stdout=PIPE)
    (adapted_parameters_json, err) = process.communicate()

    exit_code = process.wait()
    if (exit_code == 0):
        provenance = json.loads(adapted_parameters_json)
        provenance.update({"user":user, "dataset":dataset, "time_gmt":strftime("%Y-%m-%d %H:%M:%S", gmtime())})
        with open('data/adaptations.jsonl', 'a+') as outfile:
            outfile.write(json.dumps(provenance)+"\n")

        cursor.execute('set schema "public"')

        human_activity_id = int(random.random()*100)

        sql = ("INSERT INTO human_activity (id, ha_type, time, description) values \
        (" + str(human_activity_id) + ", 'TUNING','" + provenance['time_gmt'] +  "', '');")

        print(sql)

        cursor.execute(sql)



        cursor.execute("INSERT INTO attribute_tuned (human_activity_id, attribute_id, old_value, new_value)\
         VALUES ({}, {},{},{});".format(human_activity_id, 230, provenance['parameter-provenance']['attr1']['old'],provenance['parameter-provenance']['attr1']['new']))

        print provenance
```

```
69          print "Steered successfully"
```

File-based Adaptation Data Tracker

```
1  from __future__ import print_function
2  import ConfigParser
3  import sys
4  import os
5  import json
6  from StringIO import StringIO
7  from iniparse import INIConfig
8  import time
9  import sys
10
11 INI_SECTION_HEADER = "parameters"
12 TIMES_FILE_NAME = "times.json"
13 RESET_FILE_NAME = "reset.run"
14
15 def eprint(*args, **kwargs):
16     print(*args, file=sys.stderr, **kwargs)
17
18 def read_ini_file(in_filepath):
19     with open(in_filepath, "r") as ini_file:
20         ini_file_content = ini_file.read()
21     vfile = StringIO(u'[%s]\n%s' % (INI_SECTION_HEADER,
       ini_file_content))
22     in_parameters = INIConfig(vfile)
23     return in_parameters
24
25 def write_new_ini_file(in_parameters):
26     new_ini_content = in_parameters.__str__().replace("[%s]\n"%
       INI_SECTION_HEADER,"")
27     with open(in_filepath, "w") as new_file:
28         new_file.write(new_ini_content)
29
30 def steer_parameters(in_parameters, steered_parameters):
31     steering_provenance = {}
32     for p in steered_parameters:
33         if p not in in_parameters[INI_SECTION_HEADER]:
34             eprint("There is no such parameter: %s " % p)
35             sys.exit(-1)
36
37         old_v = in_parameters[INI_SECTION_HEADER][p]
38         new_v = steered_parameters[p]
39         in_parameters[INI_SECTION_HEADER][p] = new_v
40         steering_provenance[p] = {
```

```
41              "old": old_v,
42              "new": new_v
43          }
44      return steering_provenance
45
46  def write_reset_file(execution_dir):
47      reset_file_path = execution_dir+"/"+RESET_FILE_NAME
48      with open(execution_dir+"/"+RESET_FILE_NAME, "w") as run_sh:
49          run_sh.write("")
50
51
52  def wait_to_get_time_dict(execution_dir):
53      time_path = execution_dir+"/"+TIMES_FILE_NAME
54
55      while not os.path.isfile(time_path):
56          eprint("times.json not ready yet. Waiting...")
57          time.sleep(3)
58
59      try:
60          with open(time_path) as time_file:
61              time_dict = json.load(time_file)
62              os.remove(time_path)
63              return time_dict
64      except:
65          eprint("Invalid JSON in %s " % time_path)
66          sys.exit(-1)
67
68  if __name__ == "__main__":
69      if len(sys.argv) != 2:
70          eprint("You need to pass the parameters to be tuned as a valid
    JSON format")
71          sys.exit(0)
72
73      in_filepath = os.getenv("INFILE", None)
74      if not in_filepath:
75          eprint("You need define INFILE environment var with the ini
    file to be modified")
76          sys.exit(0)
77
78      steered_parameters = json.loads(sys.argv[1])
79      execution_dir = os.getcwd()
80
81      in_parameters = read_ini_file(in_filepath)
82      parameter_provenance = steer_parameters(in_parameters,
    steered_parameters)
83      write_new_ini_file(in_parameters)
84      write_reset_file(execution_dir)
```

```
    time_dict = wait_to_get_time_dict(execution_dir)

    return_dict = {
        "parameter-provenance": parameter_provenance,
        "extra-domain-data": time_dict
    }
    print(json.dumps(return_dict))
```

Redis-based Adaptation Data Tracker

```

import redis
import json
import sys
import time
import ast
from time import gmtime, strftime

class Redis_DataSet_Adapter(object):
    def __init__(self):
        self.redis_client = redis.from_url('redis://localhost:6379')

    def has_adaptation(self):
        return self.redis_client.get("adapted") == "t"

    def get_new_values(self):
        attr1 = float(self.redis_client.get("attr1"))
        attr2 = float(self.redis_client.get("attr2"))
        attr3 = float(self.redis_client.get("attr3"))
        return [attr1, attr2, attr3]

    def update_values(self,dataset_lst,iteration):
        new_values = self.get_new_values()
        self.redis_client.set("iteration", iteration)
        self.redis_client.set("old_values", dataset_lst[0])
        self.redis_client.set("adapted", "f")
        return [[new_values[0], new_values[1], new_values[2]]]


class DataSet_Adapter_ProcCall(object):
    def __init__(self):
        self.redis_client = redis.from_url('redis://localhost:6379')

    def has_adaptation(self):
        return self.redis_client.get("adapted") == "t"

```

```
37      def adapt(self, dict_new_values):
38          self.redis_client.set("adapted", "t")
39          for param in dict_new_values:
40              self.redis_client.set(param, dict_new_values[param])
41
42          #wait for the other process to conclude the adaptation
43          while self.has_adaptation():
44              time.sleep(1)
45
46          lst_old_values = ast.literal_eval(self.redis_client.get("
    old_values"))
47          iteration = self.redis_client.get("iteration")
48
49          self.save_prov(dict_new_values,lst_old_values,iteration)
50
51      def save_prov(self,dict_new_values,lst_old_values, iteration):
52          prov = dict()
53          prov["parameter-provenance"] = {
54              "attr1": {"new":dict_new_values["attr1"],"old":
    lst_old_values[0]},
55              "attr2": {"new":dict_new_values["attr2"],"old":
    lst_old_values[1]},
56              "attr3": {"new":dict_new_values["attr3"],"old":
    lst_old_values[2]}
57          }
58          prov["extra-domain-data"] = {"iteration":iteration}
59          print json.dumps(prov)
60
61  if __name__ == '__main__':
62      dict_new_values = json.loads(sys.argv[1])
63      ds2Adpt = DataSet_Adapter_ProcCall()
64      ds2Adpt.adapt(dict_new_values)
```