Scientific Workflows for Metabolic Flux Analysis

Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften
(Dr. rer. nat.)

dem Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg vorgelegt von

Tolga Dalman
geboren in Siegen

May 22, 2017

Vom Fachbereich Mathematik und Informatik der Philipps-Universität Marburg (Hochschulkennziffer 1180) als Dissertation am 23.11.2016 angenommen.

1. Gutachter: Prof. Dr. Bernd Freisleben, Philipps-Universität Marburg
2. Gutachter: Prof. Dr. Wolfgang Wiechert, Forschungszentrum Jülich

Tag der Einreichung: 18.11.2016.
Tag der mündlichen Prüfung: 12.05.2017.

Abstract

Metabolic engineering is a highly interdisciplinary research domain that interfaces biology, mathematics, computer science, and engineering. Metabolic flux analysis with carbon tracer experiments (^{13}C-MFA) is a particularly challenging metabolic engineering application that consists of several tightly interwoven building blocks such as modeling, simulation, and experimental design. While several general-purpose workflow solutions have emerged in recent years to support the realization of complex scientific applications, the transferability of these approaches are only partially applicable to ^{13}C-MFA workflows. While problems in other research fields (e.g., bioinformatics) are primarily centered around scientific data processing, ^{13}C-MFA workflows have more in common with *business workflows*. For instance, many bioinformatics workflows are designed to identify, compare, and annotate genomic sequences by "pipelining" them through standard tools like BLAST. Typically, the next workflow task in the pipeline can be automatically determined by the outcome of the previous step. Five computational challenges have been identified in the endeavor of conducting ^{13}C-MFA studies: organization of heterogeneous data, standardization of processes and the unification of tools and data, interactive workflow steering, distributed computing, and service orientation.

The outcome of this thesis is a scientific workflow framework (SWF) that is custom-tailored for the specific requirements of ^{13}C-MFA applications. The proposed approach – namely, designing the SWF as a collection of loosely-coupled modules that are glued together with web services – alleviates the realization of ^{13}C-MFA workflows by offering several features. By design, existing tools are integrated into the SWF using web service interfaces and foreign programming language bindings (e.g., Java or Python). Although the attributes "easy-to-use" and "general-purpose" are rarely associated with distributed computing software, the presented use cases show that the proposed Hadoop MapReduce framework eases the deployment of computationally demanding simulations on cloud and cluster computing resources. An important building block for allowing interactive researcher-driven workflows is the ability to track all data that is needed to understand and reproduce a workflow. The standardization of ^{13}C-MFA studies using a folder structure template and the corresponding services and web interfaces improves the exchange of information for a group of researchers. Finally, several auxiliary tools are developed in the course of this work to complement the SWF modules, i.e., ranging from simple helper scripts to visualization or data conversion programs.

This solution distinguishes itself from other scientific workflow approaches by offering a system of loosely-coupled components that are flexibly arranged to match the typical requirements in the metabolic engineering domain. Being a modern and service-oriented software framework, new applications are easily composed by reusing existing components.

Zusammenfassung

Metabolic Engineering ist eine hochgradig interdisziplinäre Wissenschaftsdomäne, welche Biologie, Mathematik, Informatik und Ingenieurswissenschaften miteinander verknüpft. Metabolische Stoffflussanalyse mit ^{13}C markierten Isotopen (^{13}C-SFA) ist eine besonders herausfordernde Metabolic Engineering Anwendung, die aus vielen miteinander eng verwobenen Bausteinen besteht, wie etwa Modellierung, Simulation und Versuchsplanung. Obwohl eine Vielzahl universeller Workflow Lösungen zur Realisierung komplexer wissenschaftlicher Anwendungen in den vergangenen Jahren entwickelt wurden, ist die Übertragung dieser Ansätze auf 13C-SFA Workflows nur teilweise möglich. Während Probleme in anderen Wissenschaftszweigen (wie etwa der Bioinformatik) vornehmlich mit Datenprozessierung zu tun haben, sind ^{13}C-SFA Workflows eher mit *Business Workflows* vergleichbar. Beispielsweise sind viele Bioinformatik Workflows derart gestaltet, dass Genomsequenzen mittels "pipelining" durch Standardwerkzeuge wie BLAST identifiziert, verglichen und annotiert werden. Typischerweise kann der nächste Workflow Schritt in der "pipeline" automatisch durch das Ergebnis des vorangegangenen Schrittes ermittelt werden. Fünf rechenbetonte Herausforderungen wurden im Bemühen um ^{13}C-SFA Studien durchzuführen identifiziert: Organisation heterogener Daten, Standardisierung von Prozessen sowie die Vereinheitlichung von Werkzeugen und Daten, interaktive Workflow Steuerung, verteiltes Rechnen und Service Orientierung.

Das Ergebnis dieser Dissertation ist ein Scientific Workflow Framework (SWF), das auf die spezifischen Anforderungen von ^{13}C-SFA Anwendungen zugeschnitten ist. Der hier präsentierte Ansatz – nämlich das SWF als eine Sammlung von miteinander lose gekoppelten Modulen zu gestalten, die mittels Web Services miteinander interagieren – erleichtert mit einigen Besonderheiten die Umsetzung von ^{13}C-SFA Workflows. Bestehende Werkzeuge sind in das SWF durch Web Service Schnittstellen sowie Programmiersprachenanbindungen angebunden (z.B. an Java oder Python). Obwohl die Attribute "einfache Handhabung" und "Universalität" nur selten in Zusammenhang mit verteiltem Rechnen gebracht wird, zeigen die vorgestellten Anwendungsfälle, dass der Einsatz des vorgeschlagenen Hadoop MapReduce Frameworks die Umsetzung von rechenintensiven Simulationen auf Cloud und Cluster Computing Ressourcen vereinfacht. Ein wichtiger Baustein um interaktive, Wissenschaftler-affine Workflows zu ermöglichen ist die Fähigkeit, alle Daten zu beobachten, die notwendig sind um einen Workflow zu verstehen und zu reproduzieren. Die Standardisierung von ^{13}C-SFA Studien mittels einer Vorlage für eine Ordnerstruktur und den dazugehörigen Web Services und Schnittstellen verbessert den Austausch von Informationen mit anderen Wissenschaftlern. Schließlich wurden im Rahmen dieser Arbeit eine Vielzahl von Zusatzprogrammen entwickelt, welche

die eigentlichen SWF Module komplementieren. Diese reichen von einfachen Hilfsskripten bis hin zu Visualisierungs- und Datenkonvertierungsprogrammen.

Die in dieser Arbeit vorgestellte Lösung unterscheidet sich von anderen Scientific Workflow Ansätzen durch ein System von lose gekoppelten Komponenten, die flexibel angeordnet sind, um den typischen Anforderungen in der Metabolic Engineering Domäne gerecht zu werden. Die moderne Softwarearchitektur und Service-orientierung des SWF erleichtern die Entwicklung neuer Anwendungen durch das Zusammenstellen und die Wiederverwendung bereits existierender Komponenten.

Erklärung

Ich versichere, daß ich meine Dissertation

Scientific Workflows for Metabolic Flux Analysis

selbständig, ohne unerlaubte Hilfe angefertigt und mich dabei keiner anderen als der von mir ausdrücklich bezeichneten Quellen und Hilfen bedient habe. Die Dissertation wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

Hochdorf, den 18.11.2016 _____

Acknowledgments

I would like to thank to thank Prof. Dr. Wolfgang Wiechert and Prof. Dr. Bernd Freisleben for giving me the opportunity to write this PhD thesis. With their input, both critical and yet encouraging, I could improve the published papers (and this thesis in particular) significantly.

Furthermore, I wish to thank my supervisor Dr. Katharina Nöh, who readily shared her experience in discussions not only constrained to topics related to her original expertise (mathematics). She could always provide intelligent and helpful answers for my questions from biology, engineering, or computer science. Beyond that, I learned a lot about doing "real" scientific work by questioning even (seemingly) trivial facts. With that, I am grateful for her ceaseless support to improve my writing qualities. Looking back, I joyfully remember our overnight sessions via e-mail working on a paper, ever polishing sections, sentences, and sometimes even single words.

During my employment at the IBG-1 in Jülich and at the ModSim group in Siegen, I am grateful for my colleagues for always being open, kind, and curious. They gave me a lot of inspiration with their ideas feedback regarding my work. Especially, I would like to express my gratitude to Dr. Michael Weitzel, who introduced me into his own work, the 13CFLUX2 toolbox, and who allowed me to contribute to his source code. As a result, the SWF now employs 13CFLUX2 as one of its major constituents. Along similar lines, I would like to thank Dr. Peter Droste for his support. He has willingly adapted many of my proposals into his Omix software for better integration with the SWF.

At the IBG-1, Dr. Stephan Zelle was my "biological peer", who taught me some of the mysteries of metabolic engineering by patiently answering basic questions about procedures in the biolab. Conversely, he quickly adopted some of the techniques to automate 13CFLUX2 with his own Python scripts. As such, he was one of the first testers of the SWF. For this, I can only express my gratitude. Also, many thanks to Johannes Runkel. As part of his diploma thesis, he has contributed to the SWF by implementing major parts of the JBoss internet web portal for 13CFLUX.net (which is still in production use).

During the work on this thesis, the support from my friends and my family were invaluable to me. I would like to express my special thanks to my former fellow students at Bielefeld University Dr. Heiko Neuweger and Dr. Jan Reinkensmeier for their feedback and giving inspiration for my own text. I am grateful for the support and encouragement my parents gave me during the work on my thesis. Finally, I would like to thank my wife Katrin for her understanding, love, and support.

Contents

1. Motivation: Computer-Assisted Metabolic Engineering **1**
- 1.1. The Experimental-Modeling Cycle . . . 2
- 1.2. From Tools, Processes, and Knowledge to a Scientific Workflow Framework 3
- 1.3. Computational Challenges . . . 4
- 1.4. Contributions . . . 6
- 1.5. Publications . . . 7
- 1.6. Outline . . . 8

2. Related Work **9**
- 2.1. Scientific Workflow Framework . . . 9
  - 2.1.1. Data Management . . . 10
  - 2.1.2. Web Services and Service-Oriented Architectures . . . 10
  - 2.1.3. Control-Flow Workflow Management . . . 10
  - 2.1.4. Tracking the Provenance of Data . . . 11
- 2.2. Comparison with Existing Scientific Workflow Solutions . . . 12
  - 2.2.1. General-Purpose Scientific Workflow Frameworks . . . 12
  - 2.2.2. Lightweight Scientific Workflow Approaches . . . 14
  - 2.2.3. Specialized ^{13}C-MFA approaches . . . 14
  - 2.2.4. Bio-jETI: An Alternative Approach . . . 15

3. Basic Concepts **19**
- 3.1. Isotope-based Metabolic Flux Analysis . . . 19
- 3.2. ^{13}C-MFA with the 13CFLUX2 Toolbox . . . 21
  - 3.2.1. Tools and Libraries . . . 21
  - 3.2.2. Model and Simulation Data Formats . . . 21
  - 3.2.3. Using the 13CFLUX2 Software in ^{13}C-MFA Workflows . . . 22
  - 3.2.4. Runtime Behavior of 13CFLUX2 Simulations . . . 23
  - 3.2.5. Monte Carlo Bootstrap Realization in 13CFLUX2 . . . 23
- 3.3. Metabolic Network Modeling and Visualization with Omix . . . 25
- 3.4. Cloud Computing with Hadoop MapReduce . . . 27
  - 3.4.1. The MapReduce Programming Model . . . 27
  - 3.4.2. Apache Hadoop MapReduce . . . 28
  - 3.4.3. Amazon's Elastic Map Reduce Cloud Service . . . 28
  - 3.4.4. Distributed MCB Implementation with Hadoop MapReduce . . . 28

4. Design and Architecture of the Scientific Workflow Framework **31**
4.1. Data Tier . . . 34
4.1.1. Storage of Scientific Raw Data in Relational Databases . . . 34
4.1.2. Version Control Systems for Storing Document Data . . . 36
4.1.3. Cloud Storage . . . 37
4.2. Application Tier . . . 38
4.2.1. Workflow Engine . . . 38
4.2.2. 13CFLUX2 and Other Simulation Tools . . . 40
4.2.3. Third-Party Software . . . 41
4.2.4. Cluster and Cloud Computing with Hadoop MapReduce . . . 42
4.2.5. Provenance Collection Framework . . . 44
4.3. Presentation Tier . . . 48
4.3.1. Software Components . . . 48
4.3.2. Web Page Design of 13CFLUX.net . . . 49
4.3.3. Content and Personas . . . 49
4.3.4. Ticket-based Authentication System . . . 51
4.3.5. SVNKit Web Integration . . . 51
4.4. Deployment Considerations . . . 55
4.5. Chapter Summary . . . 58

5. Service-oriented ^{13}C-MFA Solutions **59**
5.1. Making 13CFLUX2 Service-oriented . . . 61
5.1.1. Extending Legacy Tools with Java Interfaces using FluxCore . . . 61
5.1.2. Web Service Interfaces for Long-running Simulation Tasks . . . 62
5.2. 13CFLUX2 in the Cloud with Apache Hadoop MapReduce . . . 65
5.2.1. Life-Cycle of a 13CFLUX2 simulation using Hadoop MapReduce . 65
5.2.2. Straightforward MCB with 13CFLUX2 and Hadoop MapReduce . 66
5.2.3. FluxHadoop: Improved Implementation of the MCB . . . 67
5.2.4. Hybrid-Parallel Parameter Estimation . . . 70
5.3. Provenance Collection Services . . . 72
5.3.1. Creating Provenance Data using the Fluxlog Library . . . 72
5.3.2. Recording of Provenance Data . . . 74
5.3.3. Querying the Provenance Store . . . 75
5.3.4. Managing Provenance Services . . . 75
5.3.5. Choosing the Network Transport Protocol . . . 76
5.4. Version-Controlled ^{13}C-MFA Workflows . . . 77
5.4.1. Standard Project Template . . . 77
5.4.2. Accessing SWF Projects via Web Service Interface . . . 78
5.5. Revisiting the ^{13}C-MFA Workflow: Auxiliary Tools . . . 79
5.5.1. drawMixingTriangle: Using Python instead of MATLAB . . . 79
5.5.2. perturb: Completion of the MCB Implementation . . . 80
5.5.3. Multitools: Exploiting Simulation Performance on Local Nodes . . 80
5.5.4. collectfitdata: Consolidate the Outcome of MCB Simulations . . . 82

5.5.5. hdf5tocsv: A High-Performance Data Conversion Tool 83

6. Use Cases **85**
6.1. Mass Spectrometer Data Analysis with Hadoop DTW in R 85
6.2. Metabolic Reaction Network Modeling Workflow 86
6.3. Simulating and Comparing BCG Vaccine Models 88
6.4. Data-intensive Exploration of Flux Solution Spaces 90
6.5. Cloud Monte Carlo Bootstrap ^{13}C-MFA Workflows 93
6.5.1. Common Methods 94
6.5.2. Hadoop MapReduce Streaming Approach to the MCB Workflow . 96
6.5.3. Cloud Monte Carlo Bootstrap Workflow – Revised 98
6.6. Hybrid Parallelization Approach for MFA Simulation Workflows 100
6.7. Online Residual Tracking 104
6.8. Interactive Hadoop-based ^{13}C-MFA Workflow 105

7. Conclusions and Discussion **111**
7.1. Discussion 111
7.2. Lessons Learned 113
7.3. Outlook 114

A. Mathematical Background **115**
A.1. Notation 115
A.2. Estimation of Free Flux Parameters 115
A.3. Monte Carlo Bootstrap 115

B. Source Code Listings **117**
B.1. 13CFLUX.net front site JSP source code 117
B.2. HDF5ToCSV 118
B.3. Iterating many files on Linux 119
B.4. FWDSIM Post-Processing with Hadoop Streaming API 120
B.5. Synchronous and Asynchronous Python SOAP web service clients 121
B.6. RepoManager Web Service 122
B.7. Boxplot Script for the BCG Comparison Use Case 124
B.8. Sampling and Multicore Simulation Script 125
B.9. Histogram Plotting 126
B.10. Scatter Plot Visualization 128
B.11. Scripts for the Data Exploration Use Case 129

C. 13CFLUX2 Tools **131**
C.1. Simulation Tools 131
C.2. Sampling and Analysis Programs 132
C.3. Conversion Utilities and Reporting Tools 132

D. Apache HTTPD Configuration **135**

E. SWF Services and Components **137**

List of Symbols and Abbreviations **139**

List of Figures **141**

List of Tables **143**

Bibliography **145**

Chapter 1.

Motivation: Computer-Assisted Metabolic Engineering

Biology is the study of life and living organisms, their structure, function, growth, evolution, and distribution[1]. With the enormous scientific and technological advancements in modern biology, a paradigm shift has happened in recent years from seeking to understand single genes or cells to an integrative life science discipline (Cortassa et al., 2012). Systems biology is an emerging field that aims at a systems-level understanding of complex biological entities. Various other disciplines are fertilized by systems biology including classical life sciences such as theoretical biology, molecular physiology, cell biology, and ecology, but also mathematics, engineering, synthetic biology, and bioinformatics (Kitano, 2002).

Living cells are commonly understood as membrane-bounded organisms that consist of organic molecules in an aqueous solution (i.e., the *protoplasm*). In contrast to *proteins*, which are large biological molecules consisting of amino acid chains, the smaller organic molecules are called *metabolites*. The complete set of all metabolites found in a biological sample is termed *metabolome*. The living cell constitutes of organic matter which is the result of the cell's *metabolism*, i.e., the continuous process of transforming metabolites into cellular building blocks including proteins (*anabolism*) and vice versa (*catabolism*). Chains of biochemical reactions are called *metabolic pathways*.

Metabolic engineering is the purposeful modification of cells by using modern recombinant DNA technologies (Stephanopoulos, 1999). Contemporary metabolic engineering has a wide range of applications, e.g., industrial biofuel production using yeast cultures, production of amino acids, and analysis of pathogens for the development of effective drugs and vaccines (Stephanopoulos, Aristidou, and Nielsen, 1998). In recent years, the research focus has shifted from understanding (metabolic engineering) towards the construction of microorganisms (synthetic biology).

Today's metabolic engineering[2] workflows already include modeling and simulation steps using computational tools which are tightly coupled with experiments in biological laboratories. Recent developments embrace methods from foreign disciplines like engineering or computer science (Matsuoka, Ghosh, and Kitano, 2009). For instance, the

[1] see `http://www.biology-online.org/dictionary/Biology`; last accessed: May 22, 2017

[2] The terms *metabolic engineering workflows* can also be replaced by *systems biology* or *synthetic biology* in the following text.

emergence of high-throughput experimental setups, the increased cross-fertilization of the different "omics" fields (fluxomics, metabolomics, proteomics, genomics, etc), and the everlasting development of improved measurement devices and HPC facilities clearly hint at a continuation of this trend. Therefore, the efficient organization of transparent workflows is of utter importance for the successful realization of systems biology and metabolic engineering endeavors (Schwender, 2011). This thesis presents a *scientific workflow framework* (SWF) to support the organization and realization of complex research applications in the context of metabolic engineering.

1.1. The Experimental-Modeling Cycle

The key for the development of optimized bioproducts is the in-depth systemic analysis and understanding of the employed microorganisms. Because microbial cells are complete living organisms, and because they are comparatively easy to work with, microorganisms have been in the focus of many *in vivo* and *in vitro* studies in the metabolic engineering domain so far. At the same time, *in silico* analysis methods, i.e., computer simulations and model-based approaches, have gained lots of attention in experimental sciences (Palsson, 2011).

Driven by a biological question, model-based experimental studies typically start with a specific experiment (cf. fig. 1.1). Mimicking the functionality of the target organism or parts thereof, a mathematical model is then created that aims at answering the biological question. It is also common to take an existing model of a related organism or strain as template which is then adjusted to the scientist's requirements and biological conditions. Diverse experimental data is integrated into this model and simulations are performed. This process of model calibration requires a valid model and implies several further steps accompanying statistical assessment, sensitivity analysis, etc (cf. fig. 1.2). Afterwards, the outcome is (visually) analyzed which leads to new insights improving the knowledge about biological phenomena. Hence, by closing the loop, new experiments can be planned and conducted to iteratively improve the knowledge about the target organism.

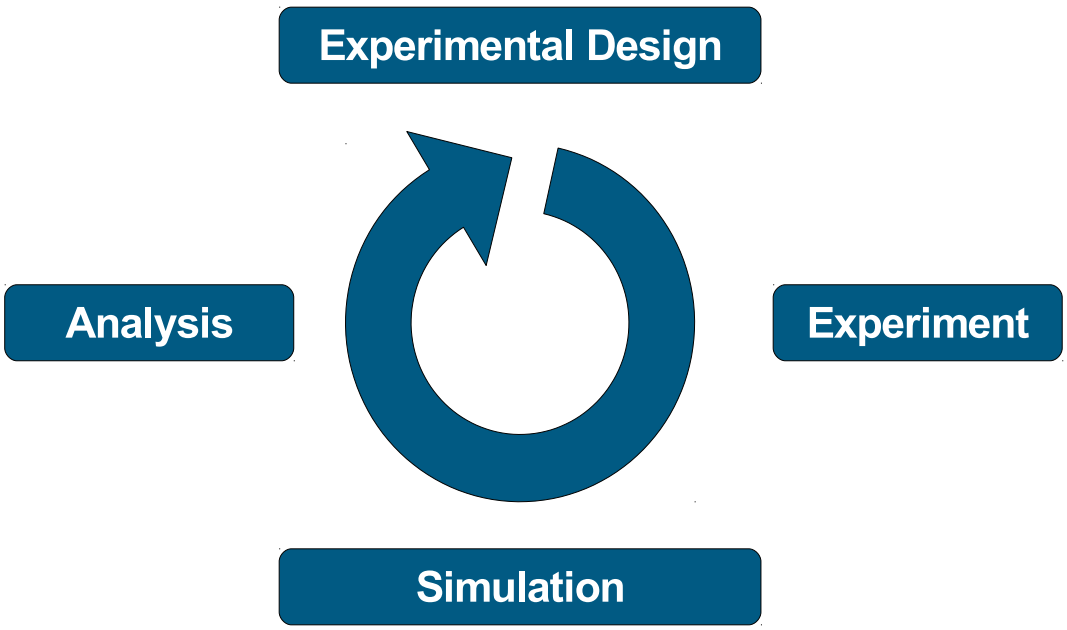


Figure 1.1.: Classical metabolic engineering experimental cycle.

1.2. From Tools, Processes, and Knowledge to a Scientific Workflow Framework

As a typical metabolic engineering application, ^{13}C-MFA is a model-based approach to estimate non-measurable intracellular reaction rates using highly-sophisticated *in silico* methods (Wiechert, 2001). Starting with a specific biological question (e.g., what are the carbon flows within a particular *C. glutamicum* strain under certain circumstances), the general procedure of ^{13}C-MFA perambulates a series of steps in a workflow (cf. fig. 1.2) (Niedenführ, Wiechert, and Nöh, 2015).

Scientists have to organize scientific data, utilize computational resources, realize workflows in a reproducible manner, and often coordinate their work with project partners from other institutes. In many scientific domains, researchers encounter similar challenges in managing their tasks (Gannon et al., 2006). Because scientific workflows consist of a series of experimental, simulation, or analysis tasks, SWFs aim at alleviating the research process by providing data management facilities, distributed computing support, data provenance tracking support, and convenient user interfaces (Tan, Missier, et al., 2010). While several prominent general-purpose SWFs are established in scientific and industrial applications today, various approaches have emerged that are adapted for the needs of

bioinformatics applications, e.g., Taverna (Hull et al., 2006), Kepler (Ludäscher et al., 2006), KNIME (Berthold et al., 2009), or Galaxy (Goecks et al., 2010).

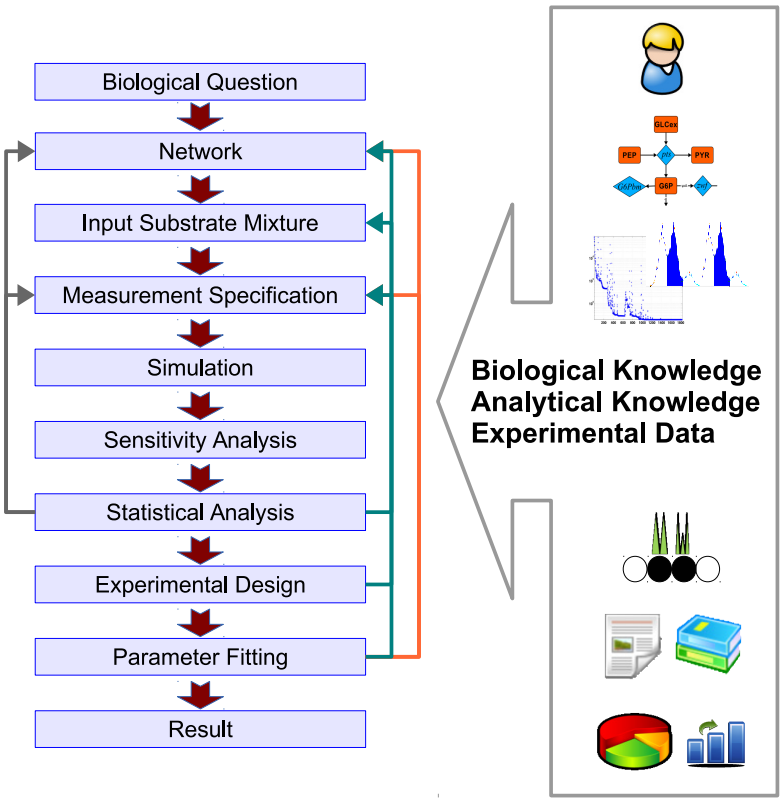


Figure 1.2.: Overview of the complete ^{13}C-MFA workflow. While the principle sequence of workflow steps is linear from top to bottom (left; red arrows), the evaluation of intermediate steps may yield a recursion to a previous step (gray, green and orange arrows). Furthermore, the individual steps are influenced by various biological, analytical and experimental knowledge and data sources (right). The figure is adopted from *Advanced Course on ^{13}C-based Metabolic Flux Analysis* materials (unpublished); see: `http://13cflux.net/13cflux2/courses.jsp`.

1.3. Computational Challenges

Many bioinformatics workflows are designed to identify, compare, and annotate genomic sequences by "pipelining" them through standard tools like BLAST or Clustal W (Altschul et al., 1990; Larkin et al., 2007). Typically, the next workflow task in the pipeline can be automatically determined by the outcome of the previous step. By contrast, ^{13}C-MFA

workflows consist of many tasks which are executed in a non-deterministic order where subsequent steps are decided by a researcher. Due to these differences, the corresponding workflows are difficult to realize in a SWF that is designed with the particular requirements of bioinformatics applications in mind.

This work presents a SWF that is custom-tailored for the specific needs of ^{13}C-MFA applications. Five computational challenges are identified for the SWF.

C1 *Heterogeneous and flexible data and tool organization*: The way from carbon labeling experiment planning to the final flux map takes a considerable amount of time (in the order of weeks and months). Scientists need support throughout all phases of this process. A plenitude of knowledge sources, experimental data, (web) databases, and software tools are involved and numerous models emerge from a typical study. All these data need to be organized in a retrievable manner. At any time, new tools (e.g., web databases, software solutions, algorithms) and knowledge may become available that need to be incorporated into the workflow.

C2 *Standardization of processes and unifying software tools and data*: A ^{13}C-MFA study inherently depends on various factors, e.g., the biological system, the experimental setup, or the analytical platform of choice. While a standardization of the overall ^{13}C-MFA procedure is currently beyond reach (Niedenführ, Wiechert, and Nöh, 2015), the unification of employed software tools and the computational parts of the ^{13}C-MFA workflow is possible. To this end, the heterogeneous information (models, experimental data) involved in computational steps has to be adequately parameterized.

C3 *Interactive research-driven workflow steering*: ^{13}C-MFA workflows are not static, but need to be adapted during their runtime. For instance, depending on the outcome of an intermediate processing step, the scientist reconfigures program parameters of subsequent simulation tasks or inserts an additional data preprocessing step. Thus, a scientist interactively decides on the basis of expert knowledge whether to re-parameterize and repeat, to proceed with a different step, or, in the worst case, to terminate and discard the entire simulation branch. These decision points can occur at any step along the workflow.

C4 *Distributed computing*: The computing intensity of ^{13}C-MFA greatly varies from task to task. For instance, during assembly, a model is frequently tested (i.e., simulated). A single simulation run takes less than a second even for a comprehensive network model on the researcher's local computer (Weitzel et al., 2013). On the other hand, large-scale computational studies (e.g., non-linear statistical analyses with Monte Carlo algorithms) require thousands of optimization runs and millions of simulations. Such calculations must be parallelized and efficiently performed on cluster or cloud computing resources.

C5 *Service orientation*: ^{13}C-MFA experts need a solution that supports the flexible assembly of workflows, while, at the same time, largely covering the complexity of

technical tasks (e.g., data security or parallelization). At many steps, on-demand control of the execution of a workflow is needed, e.g., to make ad hoc decisions based on expert knowledge. Likewise, by publishing a composite (sub-) workflow as a service, it can be readily repeated (i.e., validated) and reused for new ^{13}C-MFA applications. A service-oriented architecture (SOA) leverages the realization of such complex processes (Josuttis, 2007).

1.4. Contributions

This work presents a SWF for ^{13}C-MFA to tackle the challenges C.1–C.5. Many SWFs have been developed that abstract computational and data resources and strive to take the computational workload away from the scientists to manage operational complexities (Curcin and Ghanem, 2008). However, to the author's best knowledge, so far the aforementioned challenges have been only partially addressed by existing approaches. In this work, a novel solution is proposed that is specially tailored to the needs of ^{13}C-MFA users. The framework is located between script-oriented computational ad hoc pipelines and a dedicated software system, while combining the advantages of both solutions.

In particular, this thesis makes the following contributions to advance the state of the art:

- A software framework is developed that provides service interfaces to combine existing ^{13}C-MFA tools to scientific workflows. It is shown that purely computational workflow steps are effectively automated, thus, decreasing the complexity of typical metabolic engineering applications.
- Computationally demanding ^{13}C-MFA simulations are deployed on high-performance computing (HPC) resources. In particular, Amazon's Cloud offering is employed to effectively reduce the total computation time.
- A software solution to capture, manage, and query all data to be able to reproduce ^{13}C-MFA workflows. In particular, this so-called *provenance collection framework* is capable of gathering workflow meta information after a workflow has completed, but also in an online fashion (i.e., while a workflow is running).
- By providing a template for organizing all files of a scientific study in a Version Control System (VCS), metabolic engineering studies are effectively standardized.
- Several auxiliary tools are realized in the course of the development of ^{13}C-MFA workflows, e.g., data visualization, data conversion, or performing specialized simulation tasks.

Thereby, this work is part of an ecosystem of ^{13}C-MFA research topics at the *Institute of Bio- and Geosciences* (IBG-1) at Forschungszentrum Jülich (FZJ). Specifically, the database storing analytical raw data is conceived as part of the Jülich Measurement Data

Selector (JuMeDaS) and integrated in the context of a middle- and high-throughput methodology for the ^{13}C-MFA procedure (Miebach, 2012). Being developed several years in parallel with the SWF, the 13CFLUX2 toolbox (Weitzel, 2009) and the visualization and modeling software Omix[3] (Droste, 2011) are directly integrated in the SWF.

1.5. Publications

In the context of the work on this thesis, the following papers have been published:

T. Dalman, E. Juhnke, T. Dörnemann, M. Weitzel, K. Nöh, W. Wiechert, and B. Freisleben (2010). "Service workflows and distributed computing methods for ^{13}C metabolic flux analysis". In *Proceedings of 7th EUROSIM Congress on Modelling and Simulation*, pp. 1–7

T. Dalman, P. Droste, M. Weitzel, W. Wiechert, and K. Nöh (2010). "Workflows for metabolic flux analysis: data integration and human interaction". In: *Proceedings of the 4th International Conference on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA)*. vol. 6415. Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin, Heidelberg, pp. 261–275

T. Dalman, T. Dörnemann, E. Juhnke, M. Weitzel, K. Nöh, W. Wiechert, and B. Freisleben (2010). "Metabolic flux analysis in the cloud". In: *Proceedings of IEEE 6th International Conference on e-Science 2010*, pp. 57–64

T. Dalman, M. Weitzel, W. Wiechert, B. Freisleben, and K. Nöh (2011). "An online provenance service for workflows for distributed metabolic flux analysis". In: *Proceedings of IEEE 9th European Conference on Web Services (ECOWS)*. IEEE Press, pp. 91–98

T. Dalman, M. Weitzel, B. Freisleben, W. Wiechert, and K. Nöh (2011). "A hybrid parallelization approach for cloud-enabled metabolic flux analysis simulation workflows". In: *Proceedings of 4th GRID4TS Workshop*, pp. 30–31

M. Weitzel, K. Nöh, T. Dalman, S. Niedenführ, B. Stute, and W. Wiechert (2013). "13CFLUX2 – high-performance software suite for ^{13}C-metabolic flux analysis". In: *Bioinformatics* 29 (1), pp. 143–145

T. Dalman, T. Dörnemann, E. Juhnke, M. Weitzel, K. Nöh, W. Wiechert, and B. Freisleben (2013). "Cloud MapReduce for Monte Carlo bootstrap applied to metabolic flux analysis". In: *Future Generation Computer Science* 29 (2), pp. 582–590.

[3] Meanwhile, Omix^{TM} has become a commercially maintained project. See `http://www.omix-visualization.com/` for more information.

T. Dalman, W. Wiechert, and K. Nöh (2016). "A scientific workflow framework for ^{13}C metabolic flux analysis". In: *Journal of Biotechnology* 232. Bioinformatics for Biotechnology and Biomedicine, pp. 12–24

1.6. Outline

This thesis is organized as follows.

Chapter 2 introduces the scientific workflows concepts and the basic ingredients of a SWF. A selection of existing scientific workflow approaches are conceptually and technically compared a with the outcome of this thesis.

Chapter 3 introduces the necessary background knowledge and basic concepts. The ^{13}C-MFA method is introduced along with the domain-specific simulation toolbox 13CFLUX2 and the modeling and visualization software Omix. A brief introduction to cloud computing concepts and the Hadoop MapReduce software framework are discussed next.

Chapter 4 presents the design and architecture of the SWF which is split into three layers (data, application, and presentation tier). Thereby, the derived design decisions are discussed in due depth. Finally, considerations regarding the deployment of the software framework are illuminated.

Chapter 5 highlights the service-oriented approach of the SWF, i.e., the extension of 13CFLUX2 with web service interfaces, the combination of 13CFLUX2 with Hadoop MapReduce, services for collecting workflow runtime data, the services to organization and standardize data and studies, and a set of complementary tools and services to complete the ^{13}C-MFA procedure is discussed.

Chapter 6 presents use cases that are realized with the SWF. Thereby, the selected workflow examples are taken from different areas of the complete ^{13}C-MFA procedure as depicted in fig. 1.2.

Chapter 7 discusses the insights gained from the presented use cases and their solution using the SWF. The outcome of the thesis is summarized and hints for possible future work is given on this topic.

Chapter 2.

Related Work

By presenting related research in the area of scientific workflows, this chapter sets the conceptual and technical context of this thesis. § 2.1 gives a short introduction to scientific workflows and selected technical aspects that are relevant for this work. In § 2.2, the applicability of existing SWF approaches in the context of MFA is discussed. A special focus is set on FiatFlux-P, an alternative solution based on the Bio-jETI workflow engine.

2.1. Scientific Workflow Framework

Several definitions for *workflows* are proposed in the literature (see for example Gannon et al., 2006; W. M. P. v. d. Aalst and Hee, 2002; W. v. d. Aalst and Stahl, 2011). In this thesis, a workflow is defined as a sequence of connected steps to perform a predefined task[1].

Scientific workflows are specialized workflow applications with requirements that are typically not covered by conventional (i.e., business) workflow frameworks, e.g., support for HPC and long-running tasks (I. J. Taylor et al., 2006). Moreover, scientific workflows are typically *data-driven* because tokens of information and knowledge is generated, consumed, transformed, and deleted in a decentralized fashion. Conversely, some scientific applications require fine-grained control over the workflow execution (Tan and Zhou, 2013). *Control-flow* applications allow the fine-grained execution of the workflow using programming language structures. In the ^{13}C-MFA context, the necessity for a SWF with *control-flow* support is derived specifically from requirement C4 (cf. § 1.3).

SWFs are understood as structured environments that contain the building blocks of which scientific workflows are composed (I. J. Taylor et al., 2006). In the context of ^{13}C-MFA workflows, we identified four basic building blocks that constitute a service-oriented software solution: (i) data management facilities, (ii) distributed computing support, (iii) data provenance tracking support, and (iv) convenient user interfaces. Hence, to realize a service-oriented SWF, several functional units and software components have to be made available and interconnected through unified interfaces. These technical aspects are briefly covered next.

[1] A similar wording is found at `https://en.wikipedia.org/wiki/Workflow`; last accessed: May 22, 2017

2.1.1. Data Management

^{13}C-MFA workflows involve various inputs and outputs, i.e., measurement data from biological experiments, genome data, proteins, reaction pathways, models, and simulation data (Miebach, 2012). These inputs are typically available in data formats that need to be supported by the SWF, e.g., SBML or FluxML in the case of models.

In addition to research *bulk data*, the presence of additional model or application information is important for the reproducibility of scientific workflows, e.g., original author, document version, creation time, etc. This information is called *metadata*. Consequently, various data storage technologies need to be employed, i.e., relational and non-relational databases, public web and cloud database services, model documents, raw data files, application- and device-specific formats, and metadata storage concepts to trace the provenance of information. Further background information about state of the art data storage and organization concepts is available in the literature (e.g., see Shoshani and Rotem, 2009; Edlich et al., 2010).

2.1.2. Web Services and Service-Oriented Architectures

To fulfill the requirements of ^{13}C-MFA applications, the architecture of the SWF needs to provide several (in parts competing) features, a flexible design by modularization, support for distributed computing, and seamless access to third-party applications. Service-Oriented Architecture (SOA) is a paradigm for the implementation and maintenance of business applications and processes that employ large distributed computing resources (Josuttis, 2007). The services offered in a modern SOA are almost always realized as web services (Erl, 2014). A web service is a standardized machine-to-machine message passing communication interface that today comes in one of the two different flavors: SOAP and RESTful web services.

In many scientific disciplines the utilization of existing (legacy) software tools is crucial. For instance, several man years have been invested in the development of ^{13}C-MFA software tools, most prominently 13CFLUX2 and Omix. Hence, the re-implementation of these tools is out of reach and, thus, their integration is an important technical requirement for a ^{13}C-MFA SWF. The adoption of a service-oriented architecture is an elegant approach to integrate legacy software into complex software frameworks (Sneed, 2006). Several solutions are found in the literature that aim at the utilization of legacy tools in workflow frameworks, e.g., Soaplab2, Opal, or LCDL (Senger et al., 2008; Krishnan et al., 2009; Juhnke et al., 2009). Inspired by these approaches, § 4.2.2 presents a custom-tailored application wrapper for legacy tools that matches the requirements of 13CFLUX2 and similar programs.

2.1.3. Control-Flow Workflow Management

The de-facto standard language for *control-flow* workflows is WS-BPEL (BPEL for short) (Andrews et al., 2003). Although originating from *Business Intelligence* applications,

BPEL is in principle suited to be applied in the scientific context as well (Akram, Meredith, and Allan, 2006). BPEL is designed to perfectly fit into the SOAP web service stack by leveraging the interoperability with XML, WSDL, WS-Addressing, and others. Several commercial and non-commercial BPEL interpreters are available today[2]. A popular solution is the open source ActiveBPEL[3] engine. By providing extensions to the ActiveBPEL engine (e.g., support for long-running workflows and cloud deployment), the applicability of BPEL in the context of scientific applications is recently discussed in detail elsewhere in the literature (Dörnemann, 2013).

Beside BPEL, other workflow modeling and execution languages are available including UML and YAWL (Fowler, 2003; W. M. P. v. d. Aalst and Hofstede, 2005). Various lightweight workflow management engines have emerged that employ a traditional programming language rather than a specialized workflow language, e.g., PaPY, PyUtilib, or Hadoop (Cieslik and Mura, 2011; Hart, 2011; White, 2009). Because these approaches directly access traditional programming language features to define control-flow applications, lightweight workflow management solutions are seamlessly integrated in larger workflow applications.

2.1.4. Tracking the Provenance of Data

The collection and management of auxiliary data like intermediate results, process messages, logs, and workflow job information (e.g., date and time, or executive scientist) is of utter importance for the reproducibility and the understanding of a scientific study (Davidson and Freire, 2008). The necessary information to reproduce results from a computational step is called provenance data (Moreau, Groth, et al., 2008). Hence, provenance data consists not only of intermediate results, but also of process data, metadata, such as information about hardware and software environments, and log messages.

In particular in e-Science environments, the need to support data provenance has been identified as a vital information surplus (Oinn et al., 2006). Recently, for data-driven SWFs several provenance solutions emerged (Altintas, Barney, and Jaeger-Frank, 2006; Missier, Paton, and Belhajjame, 2010; Cao et al., 2009; Anand, Bowers, and Ludäscher, 2010). For these solutions, the so-called *provenance dependency graph* can be readily obtained by unfolding the workflow execution graph (Tan, Missier, et al., 2010). In contrast, service-oriented (i.e., BPEL-based) workflow frameworks form a *control-flow* graph. Thus, the generation and capture of provenance data has to be explicitly defined in the workflow service interfaces (Curbera et al., 2008).

The Open Provenance Model (OPM) has emerged as a comprehensive specification for a generic provenance model (Moreau, Clifford, et al., 2011). The major aim of OPM is the ability to exchange provenance information across various implementations of the standard. Several provenance solutions already provide support for the OPM standard.

[2] A list of BPEL engines is found here: `https://en.wikipedia.org/wiki/List_of_BPEL_engines`; last accessed: May 22, 2017

[3] ActiveBPEL is GPL-licensed up to version 2. The engine is further developed as the commercially distributed *ActiveVOS* solution.

2.2. Comparison with Existing Scientific Workflow Solutions

In the last decade, life sciences have experienced an increasing demand for integrative data analysis owing to the advent of various omics technologies. In turn, a variety of scientific workflow solutions have been developed that assist scientists with the implementation of new, increasingly complex data integration methodologies. Traditionally, the bioinformatics community has been the main driver of progress being pushed by the increasing flood of, e.g. (meta) genomic, proteomic and imaging data. As a consequence, several approaches have emerged that are optimized to process and analyze these data types using large collections of tools and libraries. Reviews on this topic can be found in the literature (e.g., Barker and Hemert, 2008; Curcin and Ghanem, 2008; Deelman, Gannon, et al., 2009; Romano, 2008; Tan and Zhou, 2013; Tan, Missier, et al., 2010; I. J. Taylor et al., 2006). Typically, these solutions focus on providing convenient user interfaces for modeling the data flow or deploying the computational tasks on local or distributed resources.

Existing scientific workflow solutions are categorized into three groups: (1) general-purpose engines, (2) lightweight workflow solutions, and (3) specialized systems biology solutions. Table 2.1 summarizes the findings of this section.

2.2.1. General-Purpose Scientific Workflow Frameworks

General-purpose workflow engines are widely established in sciences and industry. These scientific workflow frameworks provide a wealth of features for improving common research tasks, e.g., organization of workflow steps, management of large-scale data, distributed computing and HPC support, and they are often designed to be easily usable by non-IT experts. The Java-based *Kepler* engine aims at solving common technical problems found in modern bioinformatics workflows, i.e., the use of web service and grid computing technology, the integration of domain-specific tools, and the need to manage scientific data (Ludäscher et al., 2006). Especially with the current Kepler 2.4 (released in 2013), former extensions to improve scientific data handling and provenance data management have become an integral part of this software suite (Altintas, Barney, and Jaeger-Frank, 2006; Barseghian et al., 2010). Kepler provides a convenient graphical user interface for workflow modeling, managing workflow instances, and for inspecting the scientific results and runtime information. Kepler introduces the so-called *actor* concept, an elegant approach to integrate third-party applications or web services into the framework (Bowers and Ludäscher, 2005).

Originally aimed at life sciences (and genomics applications in particular), *Galaxy* is a domain-agnostic web platform to facilitate scientific research in general today (Goecks et al., 2010). The authors focus on *accessibility* to HPC and cloud computing resources for non-IT experts, *reproducibility* of workflows with a provenance framework, and *transparency* by providing a simplistic yet convenient web user interface. Because Galaxy is

	Kepler	Galaxy	ActiveBPEL	Hadoop	PaPy	jORCA	ReMatch	Tav4SB	LabKey	Bio-jETI
Data Management										
relational DB	+	+	-	-	-	-	-	+	+	-
version control	-	+	-	-	-	-	-	-	-	-
SBML support	-	-	-	-	-	-	+	-	-	+
FluxML support	-	-	-	-	-	-	-	-	-	-
Simulation										
workstation	+	+	+	+	+	+	+	-	+	+
Cluster	+	+	+	+	+	+	-	+	+	+
Grid	+	+	+	-	-	-	-	+	+	+
Cloud	+	+	+	+	-	-	-	+	+	+
Web Services										
SOAP client	+	+	+	-	-	+	-	+	+	+
SOAP services	-	+	+	-	-	+	-	-	-	+
REST client	+	+	-	-	-	+	-	+	+	+
REST services	-	+	-	+	-	+	-	-	-	+
asynchronous calls	+	-	+	+	-	-	-	-	+	+
Workflow Engine										
control-flow	-	-	+	+	-	+	+	-	+	+
data-driven	+	+	-	+	+	-	-	+	+	+
legacy tools	+	+	-	+	-	+	-	+	+	-
stand-alone client	-	-	-	+	+	+	+	-	-	+
multi-user support	-	+	-	+	-	-	-	+	+	+
Provenance Collection										
post-mortem analysis	+	+	-	+	+	-	-	-	+	+
online collection	-	-	-	+	+	-	-	-	-	-

Table 2.1.: Comparison between various existing scientific workflow approaches. The following technical aspects, derived from the requirements for a scientific workflow framework, are examined: (a) data management including access to relational databases, version-controlled document repositories, and support for specific SBML and FluxML formats; (b) simulations on various computer types; (c) web service support; (d) workflow engine capabilities; and (e) support for provenance collection. The selected solutions are deliberately chosen as representative scientific workflow approaches grouped in three categories: general-purpose, lightweight, and special-purpose bioinformatics.

an open source, Python-based software with a clean object-oriented architecture, several extensions and custom sites have been released already[4].

In this thesis, ActiveBPEL is classified as a general-purpose scientific workflow solution. Recently, the feasibility of ActiveBPEL is evaluated in the scientific context (Dörnemann, 2013). Other prominent scientific workflow approaches include Taverna, KNIME, Pegasus, and Triana (Hull et al., 2006; Berthold et al., 2009; Deelman, Singh, et al., 2005; Churches et al., 2006).

2.2.2. Lightweight Scientific Workflow Approaches

Lightweight approaches have emerged that provide solutions to ease the realization or the automation of research applications. These approaches distinguish themselves from general-purpose solutions by only addressing a subset of scientific workflow framework features. Originally, Apache Hadoop has emerged as a MapReduce implementation to process large amounts of data (White, 2009). Since version 2.0 onward, the open source Hadoop project is a general *Big Data Analysis* framework that includes solutions for data storage (Cassandra, HBase), data processing (MapReduce, Mahout), and workflow management (Pig, Zookeeper). As demonstrated in the previous chapters with Hadoop MapReduce, the integration of such a lightweight solutions into the scientific workflow framework is surprisingly straightforward, especially when the interfaces are kept simple.

The Python programming language has recently gained popularity in the scientific community (J. Stewart, 2014). Along with this development, several scientific workflow approaches in Python have emerged[5]. One such approach is *PaPY* (Cieslik and Mura, 2011). This approach combines data-driven processing via *parallel pipelines* with the procedural control-flow programming using the Python programming language itself. Because the parallel pipelines approach resembles the MapReduce design pattern, PaPY is comparable to our approach using Hadoop MapReduce. Like our approach, PaPY offers an extensive logging framework, which is a prerequisite for online provenance collection. PaPY provides a minimalistic GUI for visual composition and management (e.g., start, stop) of workflows.

jORCA provides a graphical bioinformatics workbench for visually composing web services (Martín-Requena et al., 2010). As such, the authors focus on a user-oriented graphical interfaces to compose and parameterize workflows, (semi-)automatic data conversion, access to various web service repositories (e.g., BioMoby and EBI), and a sophisticated service discovery approach.

2.2.3. Specialized ^{13}C-MFA approaches

In the context of ^{13}C-MFA, only two scientific workflow solutions have been made available so far. The first is ReMatch (Pitkänen et al., 2008) in combination with Biomine (Eronen

[4] `https://wiki.galaxyproject.org/PublicGalaxyServers`; last accessed: May 22, 2017

[5] For example, a partial list of scientific workflow engines in Python is found here: `https://wiki.openstack.org/wiki/NovaOrchestration/WorkflowEngines`; last accessed: May 22, 2017

and Toivonen, 2012), which is a web-based solution that leverages the assembly and export of network files for the ^{13}C-MFA tools OpenFLUX and 13CFLUX. Like 13CFLUX.net, Java Server Pages (JSP) and Java Servlets are utilized to realize the user workflows of ReMatch. The ReMatch client web pages are rendered in a JavaScript-capable web browser. Because the main functionality of ReMatch is centered around the modeling of MFA networks, it is comparable to Omix. However, SBML is the primary model format and the stoichiometry export is currently restricted to 13CFLUX FTBL models. ReMatch has KEGG and other public databases statically included, while Omix can directly access the reaction information or the reaction database of the scientific workflow framework via web services.

The second SWF approach in the context of ^{13}C-MFA is FiatFlux-P, which fosters automated calculation of metabolic flux ratios with the FiatFlux software in the context of routine computational analyses (Ebert et al., 2012; Zamboni, Fischer, and Sauer, 2005). Because FiatFlux-P shares many similarities with the present thesis, the next section is dedicated for an in-depth analysis and comparison with the SWF.

2.2.4. Bio-jETI: An Alternative Approach

During the development of this thesis an alternative scientific workflow approach is published that is based on the Bio-jETI framework (Lamprecht, 2013). To meet the requirements of typical users of the Bio-jETI framework (i.e., Biologists and Biotechnologists), the author realizes six technical aspects of that work:

1. Because the primary focus of their work is the simplification of user processes during the workflow design, Bio-jETI offers convenient user interfaces to relieve the scientists from the need to create workflow applications with traditional programming languages. Figure 2.1 exemplarily depicts the workflow design in the Bio-jETI Java client application.

2. Bio-jETI provides mechanisms to leverage the workflow design by introducing mechanisms to handle services (i.e., control flow), data (i.e., data flow), and workflows (i.e., hierarchical abstraction).

3. The author recognizes the need to integrate existing domain-specific tools in a scientific workflow approach. Thus, the need to integrate these programs in a service-oriented fashion technically and semantically is emphasized in the Bio-jETI aproach.

4. To handle the combinatorial explosion of possible workflow sequences in typical bioinformatics applications, a special focus on semantic workflow composition is handled in their work. Thereby, two aspects are highlighted: (a) the discovery of available services; and (b) (semi-) automatic workflow composition.

5. Beside supporting the composition of workflow semantically, Bio-jETI provides static and runtime workflow validation and verification mechanisms.

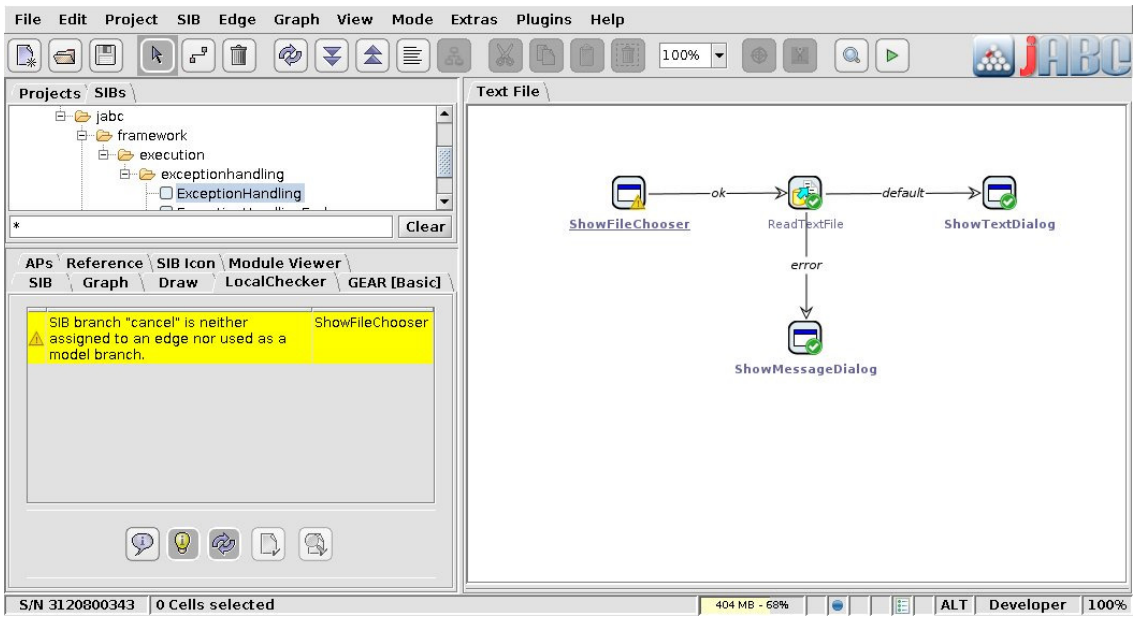


Figure 2.1.: Screenshot of the Bio-jETI client using an example from the project page. By providing a visual editor, workflows are composed in a drag-and-drop fashion. The most distinguishing feature of this approach is the semantic model checker (exemplarily displayed on the lower left pane).

6. Once defined, the execution of a workflow needs to be handled by the scientific workflow framework. The author identifies the need to handle the execution of workflow tasks. While it is possible to validate runtime tasks within the environment of the workflow framework, many tasks are executed on a different infrastructure (e.g., clusters). Thus, Bio-jETI provides a code generator to perform tasks on external systems as stand-alone applications.

Several bioinformatics applications from different areas are realized with Bio-jETI. One of these applications, called *FiatFlux-P*, is dedicated to realize ^{13}C-MFA workflows and therefore directly comparable with the SWF.

2.2.4.1. Bio-jETI/FiatFlux-P

FiatFlux-P is an application scenario using Bio-jETI that aims at standardizing and automating high-throughput ^{13}C-MFA applications (Ebert et al., 2012). FiatFlux-P uses a 'headless' MATLAB^{TM} version of the FiatFlux software to perform the ^{13}C-MFA simulations (Zamboni, Fischer, and Sauer, 2005). By employing the FiatFlux software, FiatFlux-P realizes a flux estimation based on metabolic flux ratio analysis. This static workflow consists of three sub-workflows that are executed sequentially:

- Extract and pre-process MS raw data.

- Calculate flux ratios and data post-processing.
- Estimate the intracellular flux distribution.

FiatFlux-P combines raw data analytics with the network model analysis using heuristics-based threshold values, which makes the integration of new measurement methods impossible. By exposing the computational MATLAB functions of FiatFlux, FiatFlux-P emulates the graphical user input and, hence, successfully integrates FiatFlux into the Bio-jETI workflow framework. Like the SWF, FiatFlux-P supports a variety of standardized data exchange formats, such as CSV, netCDF, and XML. Furthermore, FiatFlux-P is capable of utilizing third-party tools, particularly Omix. Comparable to the SWF, the Bio-jETI framework provides the utilization of HPC resources and the integration of existing tools via command-line programs or modern web service technology.

In summary, Bio-jETI/FiatFlux-P module is a feature-rich workflow solution for ^{13}C-MFA applications which addresses the challenges stated in the introduction. With the visual workflow editor and the semantic checker, Bio-jETI/FiatFlux-P provides well-designed graphical user interfaces and workflow steering capabilities.

Chapter 3.

Basic Concepts

This chapter introduces the basic ingredients that are relevant for the outcome of this thesis, the SWF. § 3.1 reviews the ^{13}C-MFA procedure. The simulation toolbox 13CFLUX2 and the modeling and visualization software Omix are used to realize many ^{13}C-MFA tasks (§§ 3.2 and 3.3). The SWF employs a so-called *MapReduce* framework to deploy computationally demanding 13CFLUX2 computations on high-performance machines, e.g., computer clusters or cloud computing resources (cf. § 3.4) (Jin et al., 2011).

3.1. Isotope-based Metabolic Flux Analysis

Metabolic networks are constituted by enzyme-catalyzed biochemical reactions converting substrate pools to intermediate and product pools. Key properties for the quantitative understanding of the intracellular metabolism are absolute metabolite *concentrations* and the reaction rates (the so-called *fluxes*) as well as their interaction which forms the metabolic network structure.

Because intracellular fluxes are not directly measurable, model-based approaches aim at estimating the reaction rates from available experimental data. Isotope-based MFA is a powerful method for the accurate determination of these fluxes within living cells using stable isotope tracer experiments (Wiechert, 2001). Basically, this process consists of two steps (cf. fig. 3.1):

I. **Isotope Labeling Experiment**. The workflows discussed in this thesis are based on the classical steady-state ^{13}C-MFA approach. The general approach comprises several steps: (a) *Pre-cultivation*: microbial cells are cultivated in a bioreactor until a critical population density is available (using non-labeled sources like ^{12}C-glucose). (b) *Main cultivation with feed of tracer isotopes*: the cell culture is fed with a mixture of specifically labeled isotope substrates (e.g., by exchanging specific ^{12}C substrate labeling positions in parts or completely by ^{13}C or ^{14}C tracers); (c) *Sampling*: during the transient labeling phase or as soon as the isotope enrichment is saturated (i.e., the cell culture is in the *isotopic stationary* state), samples are drawn, cell metabolism is immediately quenched, cell supernatants are removed, and samples from the purified cell solution are subjected to chemical analysis (e.g., ^{1}H-NMR or GC-MS). (d) *Chemical analysis*: labeling patterns of the samples are measured.

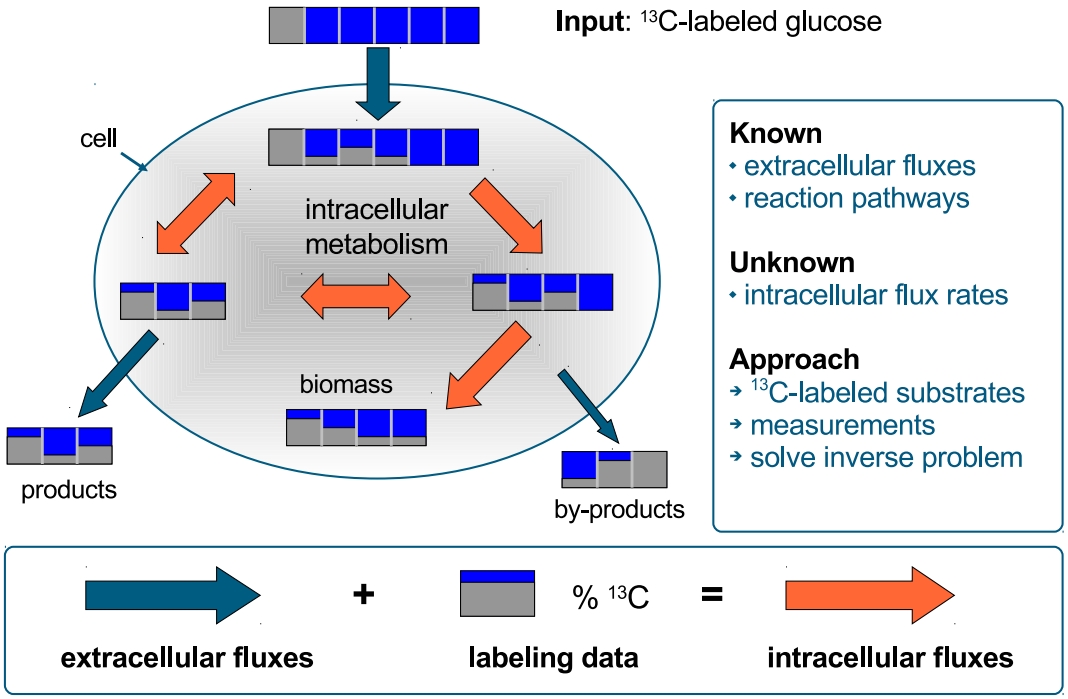


Figure 3.1.: Schematic overview of the ^{13}C-MFA method. Non-measurable intracellular fluxes are estimated by solving a high-dimensional and nonlinear parameter fitting problem (Wiechert, 1996). The underlying equation system is determined by the mass balancing of the organism reaction pathway model, whose parameters are the fluxes that have to be determined from the known input and the measured labeling fractions, an isotope-labeled input substrate mixture, and intracellular metabolite labeling patterns obtained from non-invasive measurements (e.g., ^{1}H-NMR or GC-MS).

II. **Computer-based Evaluation**. The measured fractional labeling enrichments are incorporated into an organism-specific network model that describes the labeling patterns as functions of the (unknown) fluxes. A nonlinear mathematical model is deduced that relates intracellular fluxes and measurements. The *in-vivo* fluxes are indirectly determined by solving a nonlinear least-squares problem. Finally, the quality of these estimations is assessed using statistical methods. A comprehensive overview and mathematical deduction of the general MFA procedure is found in the literature (Wiechert, 1996).

3.2. ^{13}C-MFA with the 13CFLUX2 Toolbox

For computer-based evaluation, high-performance simulation tools are readily available that are well-suited for the evaluation of experimental data sets. Today, several MFA software solutions exist, e.g., *FiatFlux*, *Metran*, or *OpenFlux* (Zamboni, Fischer, and Sauer, 2005; Antoniewicz, Kelleher, and Stephanopoulos, 2007; Quek et al., 2009). However, these approaches are based on monolithic (and in parts closed source) software architectures that makes the integration in a workflow framework difficult to realize (these issues are discussed in more detail in § 2.2.4). Simulation workflows presented in this thesis are realized with 13CFLUX2, a high-performance software toolbox for performing isotope-labeled ^{13}C-MFA in silico experiments that is developed at FZJ/IBG-1 (Weitzel, 2009).

This section introduces the contents of the 13CFLUX2 toolbox, the most important data types, and the typical usage patterns in ^{13}C-MFA applications. The runtime behavior of ^{13}C-MFA simulations with 13CFLUX2 is discussed. Because the Monte Carlo Bootstrap approach is extensively employed in this thesis, the realization of this method with 13CFLUX2 is presented at the end of this paragraph (Efron and Tibshirani, 1993).

3.2.1. Tools and Libraries

The 13CFLUX2 toolbox (Weitzel et al., 2013) consists of 20 stand-alone applications with a rich set of configuration parameters, a comprehensive C++ programming library, and a variety of convenience scripts. 13CFLUX2 programs are grouped in three categories: core simulation tools, sampling and analysis programs, and peripheral conversion and reporting utilities. A summary of the 13CFLUX2 tools is found in appendix C. The 13CFLUX2 C++ programming library allows the implementation of custom MFA applications by offering interfaces to a variety of classes, e.g., simulation, optimizer, model and measurement data access, and highly optimized internal utility functions.

3.2.2. Model and Simulation Data Formats

Being developed for several years in parallel, 13CFLUX2 and the SWF could be successfully aligned to use the same data exchange file formats.

- *FluxML*: metabolic network models are represented in XML documents. The XML format contains information about stoichiometry, metabolic network structure including metabolic reaction and atom transition networks, measurement specifications (which are arranged in *measurement groups* for each type of labeling), and initial flux values.

- *FWDSIM*: results from simulation and parameter estimation are stored in another XML document. The XML format includes estimated flux values, application runtime information, and solver and optimizer configuration parameters.

- *HDF5*: bulk data, such as matrices of floating point values, experimental and simulated measurement data, system states, or sensitivity information are stored in the HDF5 file format (Folk, Cheng, and Yates, 1999).

The unification of data types is a crucial element of the SWF because it eases the realization of composite ^{13}C-MFA programs to larger (semi-automated) workflows. For instance, a researcher can customize the visualization of data obtained from 13CFLUX2 simulations for his particular requirements (cf. § 6.4). These workflow programs are then reused in an iterative fashion for modified simulation parameters (e.g., input model variations).

Because the employed data formats are standardized across various programming languages and scientific computing tools, ^{13}C-MFA applications can be created with greater flexibility (e.g., this allowed the realization of the workflows described in §§ 6.4 and 6.8, which use different programming languages).

3.2.3. Using the 13CFLUX2 Software in ^{13}C-MFA Workflows

The 13CFLUX2 toolbox is a versatile collection of programs that can be used to realize various steps of the ^{13}C-MFA procedure. Here, a brief categorization of workflows using 13CFLUX2 is presented. A comprehensive mapping of 13CFLUX2 and other tools to the overall procedure in the context of the SWF is discussed in chapter 5 (cf. fig. 5.1 for an overview). The following types of ^{13}C-MFA workflows using the 13CFLUX2 toolbox are identified:

- **Simulation workflows**: to automate repetitive tasks, researchers often employ simple shell scripts that invoke sequences of 13CFLUX2 programs and other tools. These (sub-) workflows are often used to perform simulations on the scientist's workstation before deploying the workflow on a HPC resource.

- **Non-automatic workflows**: 13CFLUX2 tools are used in workflows involving other tools and human interaction. For instance, post-processing of simulation outcomes are typically performed using data analysis tools such as MATLAB$^{\text{TM}}$ or Microsoft Excel$^{\text{TM}}$.

- **Command-line use**: assuming valid input files (e.g., FluxML models) are present, 13CFLUX2 programs are directly executed on a Linux console terminal. This approach requires in-depth knowledge of the 13CFLUX2 toolbox and basic Linux system know-how. Experienced scientists frequently use 13CFLUX2 with the command-line during the modeling phase.

- **Custom MFA programs**: new MFA tools are developed using the 13CFLUX2 libraries to implement specialized functions. For instance, a Monte Carlo bootstrap

(MCB) parameter estimation application running on the FZJ super computer with MPI demonstrates the feasibility of this approach[1].

Addressing the above uses of 13CFLUX2, chapter 6 discusses several examples that are implemented with the SWF.

3.2.4. Runtime Behavior of 13CFLUX2 Simulations

^{13}C-MFA simulation programs show a great variance in runtime and memory usage, depending on the given input. 13CFLUX2 programs that use optimizers, like *fitfluxes* or *edopt*, are inherently non-deterministic in their runtime behavior. In addition, many algorithms are difficult to parallelize due to their dependence on model complexity. To demonstrate the variance in memory usage and runtime behavior of 13CFLUX2 simulations, four network models of different sizes are chosen. Table 3.1 summarizes the characteristics of the network models.

	file size (kB)	pools	reactions (fluxes)	C-atoms	measurements	system dimensions (unred./red.)
Model 1	5	6	7	15	5	5/2
Model 2	24	34	65	141	115	30/18
Model 3	139	76	117	404	906	51/72
Model 4	81	90	106	418	502	23/71

Table 3.1.: Overview of network model characteristics. The four models differ in file size, model size (i.e., number of defined metabolite pools, reactions, C-atom transitions, and reduced and unreduced system dimensions), and in the number of specified measurement groups.

Each model is run 1,000 times with *fwdsim* and, respectively, *fitfluxes*. The resulting mean and standard deviation values of the runtime and memory measurements are shown in Table 3.2. The models run with *fwdsim* in the order of several seconds, while the runtime of *fitfluxes* inherently depends on the input parameters. Thus, the simulations times are in the order of seconds, minutes, or hours. With Model 4, *fwdsim* consumes at most 200 MB RAM. The peak memory usage of *fitfluxes* is below 300 MB.

3.2.5. Monte Carlo Bootstrap Realization in 13CFLUX2

In order to describe the integration of long-running calculation tasks into the ^{13}C-MFA workflow, an easy to parallelize, sampling-based Monte Carlo approach is chosen. With this conceptually simple approach, the reliability of the resulting flux estimations can be assessed. Here, the MCB generates an ensemble of simulated data sets mimicking repeated experimental data from random numbers according to a probability distribution

[1] This application is realized by Sebastian Niedenführ and Birgit Stute as part of the *Advanced Course on ^{13}C-based Metabolic Flux Analysis* (unpublished; see `http://13cflux.net/13cflux2/courses.jsp`).

	fwdsim				*fitfluxes*			
	time (s)		mem (MB)		time (s)		mem (MB)	
Model 1	0.05	± 0.00	40.93	± 0.01	0.10	± 0.02	56.14	± 4.00
Model 2	0.32	± 0.02	54.75	± 0.01	84.44	± 33.61	73.05	± 3.82
Model 3	8.72	± 0.14	91.02	± 0.02	2106.42	± 633.72	144.48	± 2.91
Model 4	1.86	± 0.02	188.34	± 0.01	11726.95	± 5726.51	280.39	± 3.39

Table 3.2.: Benchmarks with *fwdsim* and *fitfluxes* reveal a great variance in memory usage and simulation runtime depending on the input network model. The simulations are performed on a server with Intel© Xeon© X7350 with 2,93 GHz and 128 GB RAM. The servers run openSUSE Linux version 12.3 with Linux kernel version 3.7.10.

(Efron and Tibshirani, 1993). For each sample (i.e., artificial measurement data set) the model fitting step is repeated. Evaluating the resultant flux estimates finally yields a measure of confidence in the estimated flux parameters. Each random data set is processed independently, hence, the MCB is inherently parallel. A brief review of the mathematical foundations of MCB is presented in appendix A.

3.2.5.1. Implementation of the MCB Algorithm with 13CFLUX2

Because the MCB method is used in many simulation workflows discussed in this thesis, the implementation of the algorithm with 13CFLUX2 is presented in detail next. The MCB method is parameterized by the number of mimicked (artificial) measurements N and the number of initial vectors M. Due to the law of large numbers, the reliability of the estimated confidence intervals grows with the increasing sampling number N (Papoulis and Pillai, 2002). For a set of experimental measurements with given uncertainties, the MCB method has been realized with 13CFLUX2. The algorithm takes a FluxML model file, the number of sample measurements, and the number of free fluxes as its input. The procedure consists of three phases (cf. algorithm 1):

1. **Preparation**: the *ssampler* call in line 2 seeds M randomly distributed initial flux values. These flux values are stored in the sample file *SAMPLES*. Following the specification of the measurements' probability distribution, N artificial data sets are generated in the outer loop using *perturb* in a new *FluxML* model file FML_P (line 6). In the inner loop (line 7), an initial flux value from *SAMPLES* is assigned to each perturbed model FML_P with the command *setfluxes* in line 10.

2. **Parameter fitting**: for each *FluxML* model (V) a *FWDSIM* file (F) is written by the parameter estimation program *fitfluxes* (line 12). Each of these XML formatted result files contains optimization-specific data, like estimated flux values, best residual, simulation runtime, and optimizer parameters.

3. **Data filtering and post-processing**: the set of $N \times M$ *FWDSIM* files generated by the parameter estimation step are collected. The *collectfitdata* program (line 13) condenses *FWDSIM* files into a binary HDF5 file for subsequent workflow tasks, e.g., for post-processing and data analysis with MATLAB^{TM}.

Algorithm 1 13CFLUX2 realization of the MCB algorithm

```
MFA-MCBootstrap(FML, N, M)
 1  ▷ Generate M random initial flux vectors
 2  SAMPLES ← ssampler(FML, M)
 3  for i ← 1 to N
 4       do
 5          ▷ Model with perturbed measurements
 6          FML_P ← perturb(FML)
 7          for j ← 1 to M
 8               do
 9                  ▷ Insert random initial fluxes
10                  V[i, j] ← setfluxes(FML_P, SAMPLES[j])
11                  ▷ Parameter estimation
12                  F[i, j] ← fitfluxes( V[i, j])
13  R ← collectfitdata(F)
```

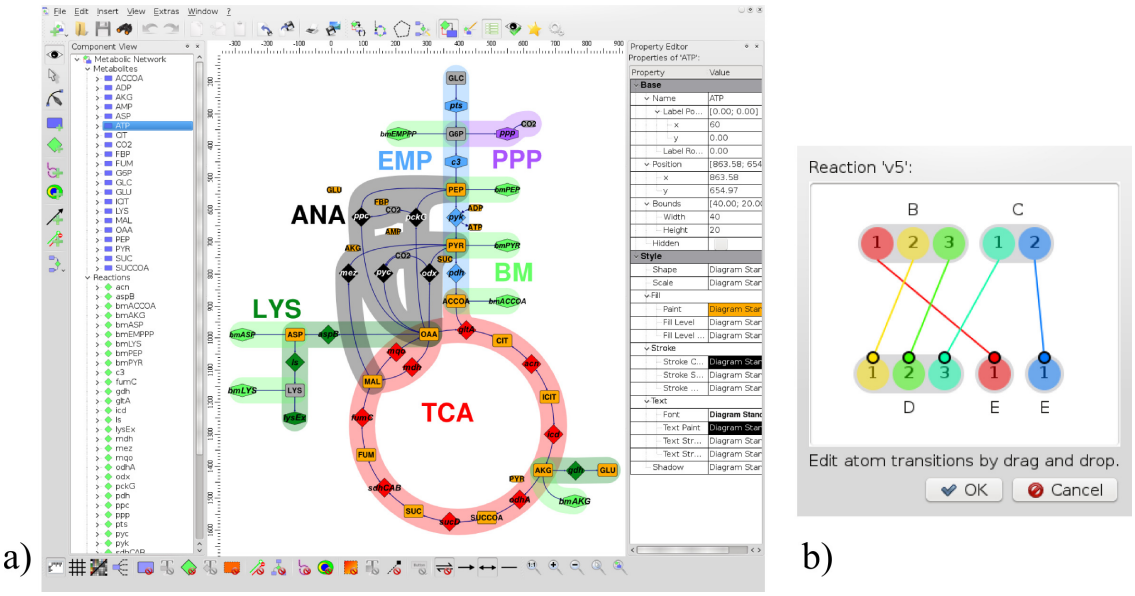
While the sampling of random measurements and initial flux values is comparably fast, the parameter fitting step consumes the vast bulk of computing time[2]. Because the input data is pair-wise independent, parameter estimation can be performed in a parallel manner, i.e., *fitfluxes* can be executed for each pair (i, j) simultaneously. This is exploited for parallel implementations of the MCB algorithm as described in § 5.2.

3.3. Metabolic Network Modeling and Visualization with Omix

The importance of user-centric and graphical guidance to perform the ^{13}C-MFA procedure has been recently discussed in the literature (Nöh, Droste, and Wiechert, 2015). Omix[3] is conceived as primary visualization and modeling tool in the FZJ/IBG-1 ^{13}C-MFA toolchain. The software graphically assists the network modeling process, i.e., the specification of mixing input substrates, measurement models, network constraints, flux directionalities, and atom transitions of reactions (Droste, 2011). By providing specialized

[2] For instance, the simulation runtime of 1,000 samples with **Model 3** is $2106.42 \pm 633.72\,s$ with *fitfluxes* and $32.44 \pm 0.24\,s$ with *ssampler*. Furthermore, the high standard deviation of the optimization compared to the raw forward simulation time emphasizes the problem.

[3] Omix web site: `http://www.omix-visualization.com/`; last accessed: May 22, 2017



a) b)

Figure 3.2.: Screenshot of the Omix user interface. a) The central window component is the drawing area for network diagrams. Toolbars containing icons around this component make various editing options available. Sidebars provide information about network properties. b) Atom transitions of the reactions can be edited graphically with drag and drop.

visualization methods, Omix is employed to explore and analyze the outcome of ^{13}C-MFA studies. Figure 3.2 depicts screenshots of the Omix user interface.

During the realization of ^{13}C-MFA workflows with both tools, Omix and the SWF, several technical challenges are identified, which are worthwhile to improve (Dalman, Droste, et al., 2010). Because Omix is typically run on the scientist's local workstation, the handling of model and result files during the modeling and visualization phase means that the interaction between the different modeling, simulation, visualization, and analysis tools needs the exchange of files with other tools (export, save, transfer, etc).

This procedure becomes at least cumbersome and error-prone – if not impossible – in the large-scale: firstly, large-scale simulations are infeasible to be performed on local nodes. Instead, these simulation jobs need to be deployed on a compute cluster or cloud computing resources. Secondly, in the FZJ/IBG-1, Omix (which is developed in Java) is executed on researcher workstations on Microsoft Windows operating systems, while the 13CFLUX2 simulation toolbox is optimized to run on Linux and Unix operating systems.

The optimization of such technical tasks is one step towards a unified and, thus, more standardized way of performing ^{13}C-MFA (challenge C2). Beside the variance in runtime and memory usage, ^{13}C-MFA tasks can be performed locally or in a distributed fashion. For instance, networks are tested on the scientist's workstation during the modeling phase, while a compute cluster is employed to perform large-scale Monte Carlo simulations.

Fortunately, Omix provides an extensible plug-in mechanism which eases the integration with third-party tools. Specifically, Omix and 13CFLUX2 programs are combined by employing these plug-ins and the SWF web services. Later sections present the design (§ 4.2.2) and implementation (§ 5.1) of the SWF web service interfaces. The challenging variety of ^{13}C-MFA simulations using the scientific workflow approach taken by this thesis is discussed in chapter 6 in due detail.

3.4. Cloud Computing with Hadoop MapReduce

Because the ^{13}C-MFA workflow employs several highly-specialized software packages (which are possibly maintained by a third-party), it is desirable to integrate these tools seamlessly into a distributed computing environment. Like many other contemporary scientific workflow projects, this work addresses computationally challenging tasks by employing sophisticated cloud computing solutions. Cloud computing in general is covered in depth elsewhere in the literature (e.g., Mell and Grance, 2011; Buyya, Broberg, and Gościński, 2011; Erl, 2014).

In recent years, the *MapReduce* architectural pattern has evolved as a generic, domain-independent processing method for large amounts of data. Several *MapReduce* implementations have emerged that utilize cluster and cloud computing resources for performing computationally demanding tasks in an *embarrasingly parallel*[4] fashion. With *MapReduce*, an elegant approach to implement Monte Carlo methods (the Bootstrap algorithm in particular) massively parallel on local clusters and cloud computing HPC resources is available (Scott, Blocker, and Bonassi, 2016).

The usefulness of *MapReduce* is demonstrated in various scientific disciplines (especially life sciences) has been presented multiple times in the literature (e.g., Pratx and Xing, 2011; Wall et al., 2010; Matsunaga, Tsugawa, and Fortes, 2008). *MapReduce* and the cloud-capable de-facto standard implementation Hadoop MapReduce is briefly recapitulated in the next paragraphs. On the example of the aforementioned MCB algorithm, the applicability of this approach in ^{13}C-MFA applications is demonstrated in this thesis.

3.4.1. The MapReduce Programming Model

Two functions, `map` and `reduce`, are required to be implemented by the user with the following prototypes (Dean and Ghemawat, 2004):

$$\begin{aligned} \texttt{map}\ (k1, v1) &\rightarrow list\ (k2, v2) \\ \texttt{reduce}\ (k2, list(v2)) &\rightarrow list\ (v2) \end{aligned}$$

These interfaces are similar to those present in Lisp and other functional programming languages. *list* denotes a list of objects, $k1$ and $k2$ represent key types, $v1$ and $v2$ are value types. The input key/value pairs ($k1$, $v1$) are pairwise independent, thus, `map` can be

[4] https://en.wikipedia.org/wiki/Embarrassingly_parallel

invoked in parallel for all pairs, yielding an intermediate list of mapped ($k2$, $v2$) pairs. For each key $k2$, the corresponding values $v2$ are grouped and passed to the `reduce` function, which merges – or reduces – final result values to a list of type $v2$.

3.4.2. Apache Hadoop MapReduce

The open-source Apache *Hadoop* project has emerged as the de-facto standard implementation for the *MapReduce* programming model (White, 2009). Providing custom `map` and `reduce` functions, *Hadoop* automatically manages parallel and fail-safe execution of these functions on traditional clusters as well as on-demand cloud infrastructures. As an outstanding feature, *MapReduce* jobs may be defined by using native libraries (e.g., C++ and Java), or by providing `map` and `reduce` as console applications for the streaming API.

3.4.3. Amazon's Elastic Map Reduce Cloud Service

To deploy Hadoop jobs in the cloud, Amazon's cloud Hadoop offering named *Elastic Map Reduce* (EMR) is employed. EMR is built on top of the web storage service Amazon S3 and the virtual infrastructure service Amazon EC2. On-demand resource access to computational nodes and storage is possible by web service interfaces without knowledge or control of the technology and the infrastructure provided. Virtual machine instances are used like dedicated physical hosts typically within a few minutes after the provisioning request. The configuration is customized with respect to the number of CPUs, amount of RAM, and instance storage. In addition to these settings, the price per time slot varies (Dörnemann, Juhnke, and Freisleben, 2009). Pre-configured virtual machines running Hadoop are offered that obviate the need for setting up an own Hadoop cluster.

3.4.4. Distributed MCB Implementation with Hadoop MapReduce

The presented MCB algorithm with 13CFLUX2 is easily applied to the *MapReduce* programming model. Given a list of prepared FluxML files (i.e., configured with artificial measurements and random initial flux values), Algorithm 2 depicts a straight-forward *MapReduce* solution of the MCB procedure. A remarkable feature of this solution is that the 13CFLUX2 tools are used *as-is* to parallelize the simulation on cloud computing resources. Thus, with Hadoop MapReduce and Amazon's EMR, a cloud solution for ^{13}C-MFA is available. § 5.2 presents the implementation details of the Hadoop MapReduce approach taken in this thesis.

Algorithm 2 Parallel MCB algorithm with MapReduce

Map(*Filename*, *FML*)

1 ▷ Parameter estimation
2 *FWDSIM* ← fitfluxes(*FML*)
3 ▷ *Success* indicates whether the execution of fitfluxes was successful
4 **return** (*Success*, *FWDSIM*)

Reduce(*Success*, *FWDSIMList*)

1 ▷ Only collect successful fitfluxes runs
2 **if** *Success* = *True*
3 **then return** collectfitdata(*FWDSIMList*))

Chapter 4.

Design and Architecture of the Scientific Workflow Framework

Various definitions for *software architecture* exist in the literature. In this work, the functional units and their arrangement as well as the sum of all software components, their interfaces, and interconnections are understood as the system's *software architecture*. In this sense, it's main purpose is to document and to show the "big picture" of the software (Hohmann, 2003). A popular way to represent software architectures is the so-called "4+1 model", which consists of four different views (logical, process, physical, and development) and a set of scenarios that put the different views together (Kruchten, 1995).

By roughly following this scheme, this chapter is mainly concerned with the presentation of the SWF from a *logical view* (cf. fig. 4.1). Thereby, the components are distributed in a classical three layer architecture (Fowler, 2002). The *data tier* (§ 4.1) provides persistent storage resources, while the *application tier* (§ 4.2) contains the processing and service layers of the workflow execution environment and, thus, represents the heart of the SWF. Web-based user interfaces to access scientific and process data produced throughout various stages of the ^{13}C-MFA workflow are included in the *presentation tier* (§ 4.3).

By taking a *physical view* on the software architecture, the deployment of the SWF components is discussed in the context of the FZJ/IBG-1 server infrastructure (§ 4.4). Although explicit discussions concerning the remaining views are omitted in this chapter, the 4+1 model is complemented by situational *process view* discussions (e.g., § 5.2.1) and the use cases (cf. chapter 6). These general design decisions are made for the SWF:

- **Design Decision 1:** The SWF follows the SOA paradigm, hence, functional components of the ^{13}C-MFA procedure (i.e., tasks of sub-workflows) are wrapped and exposed as web services whenever possible. While some programming languages come with convenient web service facilities (e.g., Java, C#, Python, or Ruby), extending programs written in C, C++, or Fortran with such an interface requires a dedicated wrapper layer (Erl, 2004). These applications are attributed as legacy, despite the fact that many modern high-performance applications are developed in a programming language that enables the exploitation of the underlying hardware (Kumar et al., 2003).

- **Design Decision 2:** While it is possible to develop every part of the SWF from scratch, existing software solutions (i.e., libraries and tools) are employed whenever feasible and new components are developed only when necessary.

In a similar manner, design decisions that are derived from the specific requirements of the ^{13}C-MFA procedure and the employed tools are highlighted as the three layers of the SWF design are presented in subsequent sections.

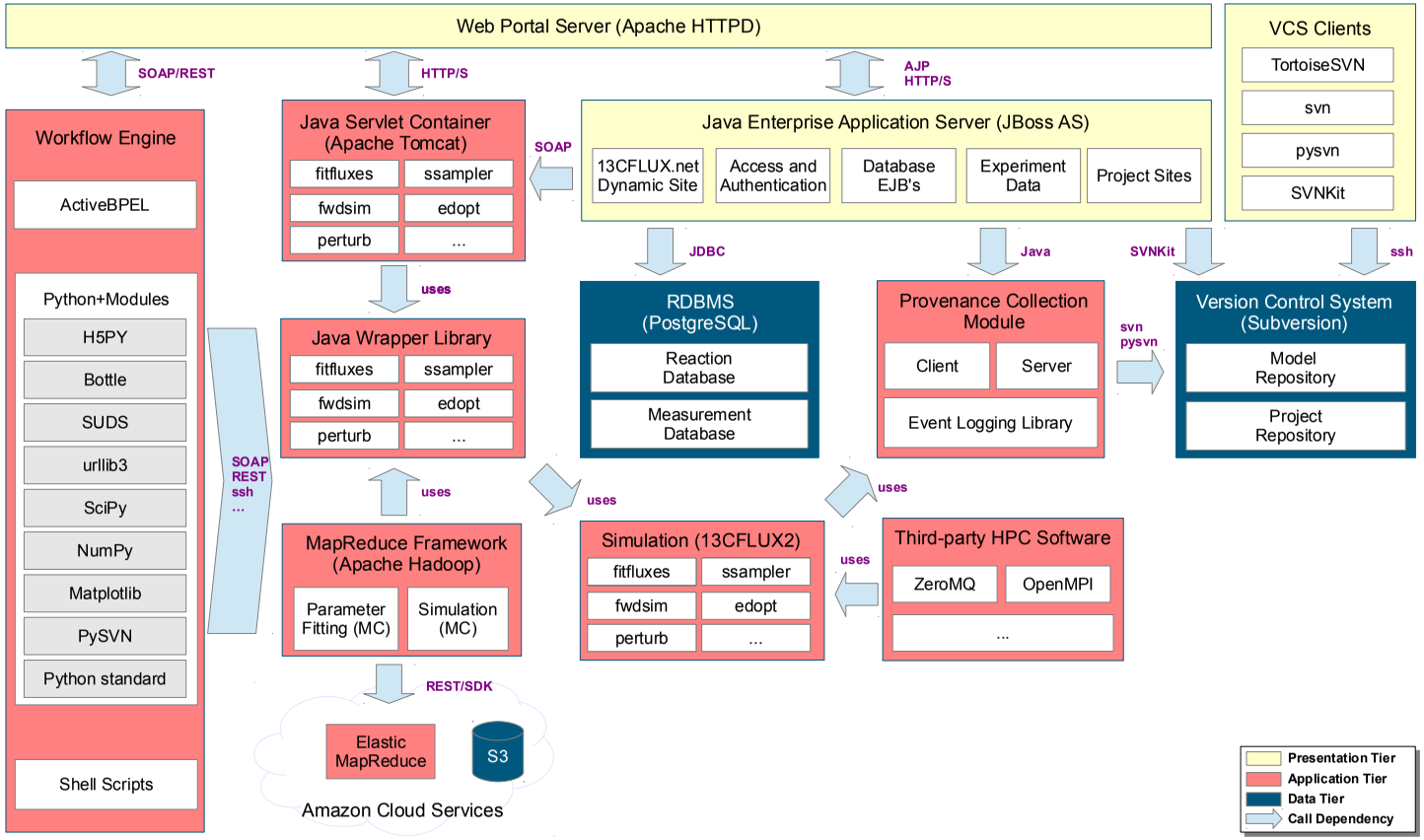


Figure 4.1.: Design of the SWF for ^{13}C-MFA. The SWF is divided in three layers. The *presentation tier* (yellow boxes) includes an Internet web site that is realized by a static and a dynamic HTML portal server. The workflow processing, data provenance and execution are part of the *application tier* (red boxes). Web services and interfaces to storage solutions are also part of this layer. The SWF supports both SOAP and REST web services, e.g., BPEL shares the WS-* technology with SOAP while Amazon's services are based on REST. Various storage solutions, i.e., relational databases, cloud storage, and document repositories are part of the *data tier* (blue boxes). The arrows specify call dependencies.

4.1. Data Tier

^{13}C-MFA workflows involve both, raw and pre-processed, inputs and outputs, i.e., experimental data, biochemical information, metabolites, reaction pathways, atom mappings, models, and simulation data (Wiechert, 2001; Yang, 2013; Zamboni, Fendt, et al., 2009). The employed tools and the data, have both their own data formats. Hence, the SWF needs to support – at least to some extent – facilities to load and convert these formats. Besides the data, biological and technical meta-information is vitally important (how is the experiment performed? What simulation parameters are used in the study?). To cope with this heterogeneous information, different data storage technologies are used: relational databases for bulk data, a version control system for documents, and a cloud-based storage solution.

4.1.1. Storage of Scientific Raw Data in Relational Databases

Because scientific raw data (e.g., measurements) is typically recorded once and accessed many times thereafter, relational SQL databases are employed to store this kind of information persistently.

- **Design Decision 3:** Experimental data is managed in the open source PostgreSQL software, which offers SQL:2008 compliance, database clustering, and transactions (Obe and Hsu, 2014). In addition, PostgreSQL provides a JDBC driver which allows the integration with a Java Enterprise application server (Rubinger and Burke, 2010).

- **Design Decision 4:** Scientific raw data is distributed on three databases which are identified in the ^{13}C-MFA context: a database for experimental data and measurements, a model database, and a unified database to represent reaction information.

4.1.1.1. Measurement Database

This database is part of *JuMeDaS* (Jülich Measurement Data Selector), a management system for analytical raw data from middle- and high-throughput experiments with ^{13}C isotope tracers (developed by Miebach, 2012). Figure 4.2 summarizes the database schema which consists of eight tables grouped in three categories: analytical data, process data, and measurement meta information.

4.1.1.2. Model Database

The model database provides interfaces to organize and access model documents. The following tables are defined in the database (cf. fig. 4.3).

- The `models` tables organizes the model metadata, i.e., the associated project, name and description of the model, owner, and insertion date.

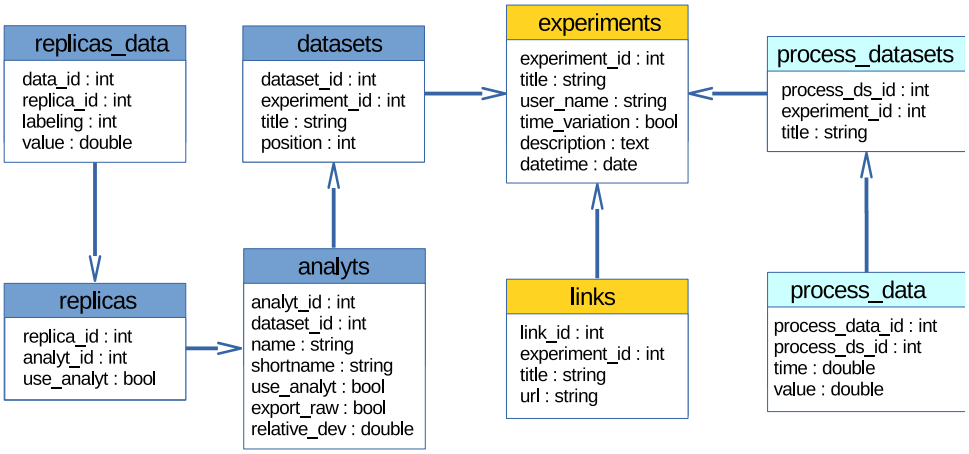


Figure 4.2.: UML diagram of the JuMeDaS measurement database schema. The tables are grouped in three categories: measurement data (blue), process data (yellow), and metadata (cyan).

- Models are organized in `projects`, where the access permissions are organized per project in the `permissions` table.
- Because typical ^{13}C-MFA workflows frequently modify a model, the `model_versions` table tracks the various changesets of a model including a history log.
- The `files` table contains metadata of the actual model file, i.e., name, type, and location in the filesystem (which is backed by a VCS project repository; cf. § 4.1.2).

The access to the model database is realized in the `ModelManager` EJB[1]. The management of experiments is realized in the `ExperimentManager` EJB[2].

4.1.1.3. Reaction Database

JuMetReD (Jülich Metabolic Reaction Database) unifies the access to various knowledge sources in the context of metabolic reactions and pathways (Fuhrmann, 2010). Because the tracking of atom transitions is required for ^{13}C-MFA, this information is annotated to the reactions in addition. The database consists of 21 tables and is grouped in four

[1] The WSDL of the model manager web service provides details on the available operations: `https://www.13cflux.net/ModelDB-ejb/ModelManager?wsdl`; last accessed: May 22, 2017

[2] The WSDL of the experiment manager web service provides details on the available operations: `https://www.13cflux.net/MeasurementDB-ejb/ExperimentManagerImpl?wsdl`; last accessed: May 22, 2017

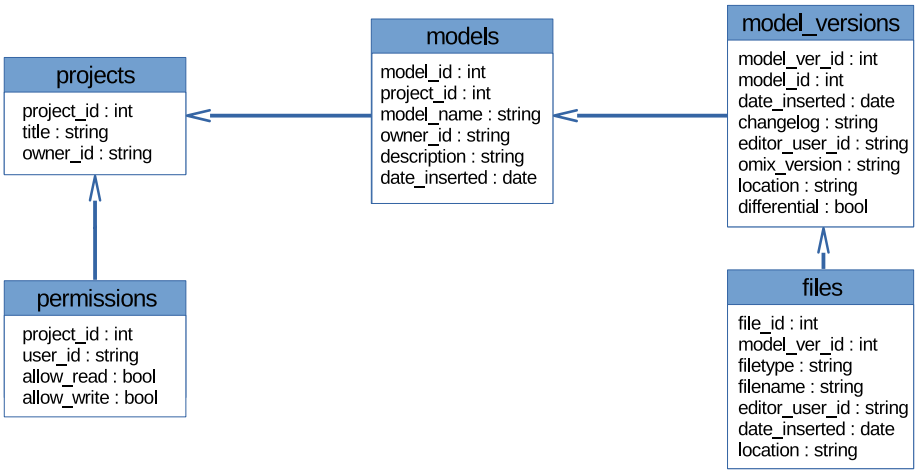


Figure 4.3.: UML diagram of the JuMetReD model database schema. Five tables are defined: `models`, `projects`, `permissions`, `model_version`, and `files`.

parts: compounds, reactions, hierarchical entities, and literature. Originally developed as an independent Java Enterprise tool, JuMetReD is fully merged into the SWF[3].

4.1.2. Version Control Systems for Storing Document Data

Unlike bulk data, document-based files (especially FluxML models, but also spreadsheet files containing evaluated raw data) are constantly reviewed and changed in the course of a typical ^{13}C-MFA study. While it is technically possible to store documents in a relational database, performance degradation is to be expected in the long run, especially for larger documents (Sears, Ingen, and Gray, 2006). In addition, especially text-based documents will consume the full file data space in the RDBMS, even for small changes.

Design Decision 5: A *Version Control System* (VCS) is employed to organize these documents efficiently, e.g., by tracking each modification with a change history per file and per folder (Tichy, 1982). Modern VCS solutions store files incrementally, sometimes even binary data which is beneficial for space consumption and performance (Hudson, 2002).

4.1.2.1. Subversion and Other VCS Software

Table 4.1 compares five popular VCS solutions that are selected from existing open source and commercial software packages[4] for inclusion into the SWF: Subversion, CVS, Perforce,

[3] The integration into the SWF is realized by Runkel, 2009 as part of his diploma thesis

[4] A comprehensive list of other VCS software is found at Wikipedia: `https://en.wikipedia.org/wiki/List_of_version_control_software`; last accessed: May 22, 2017.

	Subversion	CVS	Perforce	Mercurial	git
GUI client	yes	yes	yes	yes	yes
console client	yes	yes	yes	yes	yes
API support	yes	yes	no	yes	yes
Java client lib.	SVNKit	no	no	no	JGit
Python client lib.	PySVN	no	no	native	GitPython
atomic operations	yes	no	yes	yes	yes
directory tracking	yes	no	no	no	no
open source	yes	yes	no	yes	yes
VCS style	client-server	client-server	client-server	distributed	distributed
comm. support	CollabNet	none	yes	none	GitHub Inc.

Table 4.1.: Comparison table of popular VCSs. The decision criteria include the availability of GUI and console clients, Java and Python API support, atomic committing of a set of changed files, and the ability to track empty folders in the VCS.

Mercurial, and git. The most important decision criteria for the VCS software in the context of ^{13}C-MFA are: the availability of GUI and console clients, support for writing custom clients in Java and Python, atomicity (i.e., check-ins of multiple documents are performed as whole or not at all), and tracking of empty directories (which is useful for tracking project templates).

Design Decision 6: The Apache Subversion software is employed to store document-oriented data (Collins-Sussman, Fitzpatrick, and Pilato, 2008). Table 4.2 summarizes the available clients for Subversion: the Microsoft Windows Explorer plug-in *TortoiseSVN*, the *svn* command-line, the Java programming library *SVNKit*, and the Python SVN client library *PySVN*.

4.1.3. Cloud Storage

Complementing traditional local devices, cloud computing infrastructures provide storage renting services that have become popular in computational biology, most commonly the Amazon S3 cloud service (Palankar et al., 2008). One of the pillars of cloud computing are web services which are used to directly integrate the Amazon S3 storage (`https://aws.amazon.com/s3/`). The Amazon S3 command-line utility *s3cmd* or a web service interface can be employed to access, modify, or delete data in the cloud. As Amazon S3 provides a web-based user interface, published data files can be instantly shared with other scientists via the Internet.

Design Decision 7: Amazon's S3 service is used as cloud storage.

	TortoiseSVN	svn	SVNKit	PySVN
add/rm/ci/co/diff/log	yes	yes	yes	yes
check-in per file	yes	yes	yes	yes
check-in per folder	yes	yes	no	yes
check-in of file selection	yes	no	no	yes
programmable	no	no	yes (Java)	yes (Python)
usability	easy	intermediate	easy	difficult
allows partial checkouts	yes	yes	n/a	yes
requires check-out	yes	yes	no	yes
search revision history	yes	yes	yes	yes
search in repository	yes (Explorer)	yes (grep)	no	no

Table 4.2.: Comparison of four Subversion VCS clients. (a) TortoiseSVN, a graphical tool with advanced access features; (b) the standard console tool *svn* for performing expert tasks; (c) custom Java client application the SVNKit library; and (d) custom Python client scripts utilizing the *PySVN* Python library. The usability aspect is a subjective measure and reflects the required user experience to use this tool effectively.

4.2. Application Tier

Within the application layer, the overall business logic of the SWF is realized. The very core of the SWF is the workflow engine, which is responsible for managing and executing the workflow applications. In particular, the engine orchestrates access to data layer services, computational services, and allows for the deployment of simulation tasks on cluster or cloud computing resources. These services (including those provided by the workflow engine) are then utilized by the presentation layer (cf. § 4.3).

4.2.1. Workflow Engine

In the SWF, the workflow engine is responsible for executing control-flow programs. An expressive language (i.e., Turing-complete) is required to represent scientific workflows, which are typically non-deterministic, iterative, and recursive (W. v. d. Aalst and Stahl, 2011). The SWF is designed as a system of loosely-coupled applications, therefore, it must be possible to compose workflows from standard SOAP and REST web services (cf. Design Decision 1). Because the business web services are usually executed instantaneously, synchronous service interfaces are considered to be sufficient. However, in the ^{13}C-MFA context, support for long-running tasks is a must.

In practice, a general-purpose programming language suffices to realize even complex ^{13}C-MFA applications. Conversely, a specialized workflow engine (i.e., BPEL-based) offers many benefits for large-scale modeling of complex workflows, e.g., with graphical tools, BPMN compatibility, and cloud-awareness (Dörnemann, 2013). For the SWF, the following design decisions are made:

- **Design Decision 8:** The general-purpose programming language Python is employed in the SWF to compose ad-hoc workflows. Several reasons are factored into this decision:
  - Python is widely used in the life sciences community (Cock et al., 2009).
  - Because Python is among the five most popular programming languages[5], a vast amount of literature, internet resources, and add-on modules are available.
  - Python is one of the best languages in terms of interoperability with other programming languages, e.g., with Shell, C or C++, which is in particular relevant for the SWF (Bissyandé et al., 2013).
- **Design Decision 9:** Because ActiveBPEL is designed to seamlessly integrate in any SOA environment, the SWF provides optional support for BPEL-based workflows.

The Python workflow approach as well as a brief comparison with ActiveBPEL (including the scientific workflow extensions discussed in Dörnemann, 2013) is presented next.

4.2.1.1. Python Tools and Libraries in the ^{13}C-MFA context

To facilitate the ad hoc realization of ^{13}C-MFA workflows, several open source Python libraries and tools are integrated into the SWF. Specifically, the workflow engine interoperates with any installed application tool by accessing web-service wrappers that have well-defined I/O interfaces. By employing the Python programming language as the workflow engine, ^{13}C-MFA applications can be flexibly composed with the immediate availability of a plenitude of standard libraries and third-party components. Specifically, the following libraries are essential for the development of most ^{13}C-MFA workflows (see also use cases in chapter 6):

- *NumPy* and *SciPy* provide high-level scientific computing functionality, e.g., numerical and statistical methods (J. Stewart, 2014).
- The HDF5 Python library *h5py* (Collette, 2013) as well as the Python modules *csv* and *xml* support the most important ^{13}C-MFA file formats.
- *urllib3* (`urllib3.readthedocs.org/`) is chosen to implement REST clients, while SOAP clients are implemented with *SUDS* (`pypi.python.org/pypi/suds-jurko/`). To provide REST web services, the Python *Bottle* module (`bottlepy.org/`) is employed.
- *PySVN* is used to access Subversion repositories out of a workflow program.

[5] The TIOBE programming community index aggregates the popularity from various internet search engines such as Google, YouTube, Bing, or Amazon. See `http://www.tiobe.com/tiobe_index?page=index`; last accessed: May 22, 2017

Requirement	ActiveBPEL	Python
Turing-complete language	yes	yes
SOAP web services	yes	library
REST web services	no	library
long-running tasks	WSRF	yes
HPC support	yes	library
cloud computing support	yes	library
DBMS access	SOAP only	library
document repository access	SOAP only	library
graphical modeling	yes (BPMN)	yes (UML)

Table 4.3.: Comparison of ActiveBPEL and Python as workflow engine. Using a broad range of freely available extensions, Python largely covers the requirements of the SWF. By providing SOAP web service interfaces, ActiveBPEL is easily integrated into the SWF.

4.2.1.2. Comparison of Python and ActiveBPEL

A specially-tailored version of the open source software ActiveBPEL, which includes support for long-running processes, scalability, reliability (in terms of fault tolerance), and data security is included as part of the SWF (Dörnemann, 2013). As ActiveBPEL runs in a Java Servlet Container (such as Apache Tomcat), it fits well into the overall software architecture of our SWF. Table 4.3 compares ActiveBPEL with Python according to the requirements of the SWF.

4.2.2. 13CFLUX2 and Other Simulation Tools

Simulation programs are integrated into the SWF using a dedicated interlayer that wraps the original application and provides suitable interfaces (e.g., web services or Java). Because most of the workflows discussed in this thesis employ 13CFLUX2, the legacy wrapper component needs to provide a mechanism to handle the specific parameters of these programs. While most parameters are program-specific, some options are shared among 13CFLUX2 tasks in the workflow. For instance, all programs of a simulation workflow should use the same provenance store settings to enable complete workflow traceability (cf. § 4.2.5).

Design Decision 10: Develop a novel lightweight legacy software wrapper with Java and web service interfaces.

An analysis of existing legacy wrappers reveals that existing approaches are either complicated to use, or too restricted in their functionality (the results of the research are detailed in Dalman, Juhnke, et al., 2010). Therefore, a new Java library called *FluxCore* is developed that fills this gap. *FluxCore* is designed to provide object-oriented interfaces for all programs of the 13CFLUX2 software collection (cf. fig. 4.4). Using JAX-WS, the Java Servlet *FluxWS* provides SOAP web service interfaces in an Apache

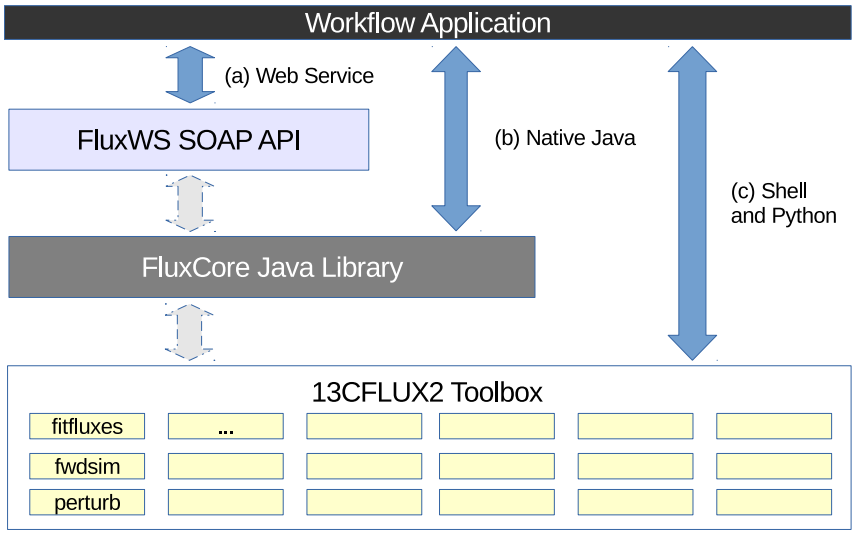


Figure 4.4.: Layers of the 13CFLUX2 legacy wrapper architecture. A workflow application accesses 13CFLUX2 by three successive interfaces (from left to right): (a) FluxWS is a SOAP web service API; (b) FluxCore is a thin wrapper that provides a native Java interface; and (c) 13CFLUX2 can be invoked directly using Shell or Python scripts.

Tomcat container for all public methods in *FluxCore*. With this extensible and lightweight design, *FluxCore* is also compatible with third-party command-line tools as well. Because *FluxCore* and *FluxWS* easily enable the integration of (legacy) simulation tools into the SWF in a service-oriented manner, § 5.1 provides an in-depth discussion about the specific realization of 13CFLUX2 web services.

4.2.3. Third-Party Software

Computationally demanding and data-intensive applications are best executed on high-end workstations and computer clusters. While the SWF imposes no technical restrictions on employing traditional HPC tools and libraries (as opposed to cloud-based solutions using web service interfaces), the following third-party tools are selected to realize HPC applications in combination with the SWF.

- **Design Decision 11:** MPI is a well-known event-driven distributed programming interface (Pacheco, 2011). Using *mpi4py* (`mpi4py.scipy.org/`; part of *SciPy*), MPI is easily used with Python programs.
- **Design Decision 12:** ZeroMQ is a networking library that is designed around a set of powerful communication design patterns such as request-reply or

publish-subscribe (Hintjens, 2013). PyZMQ (`github.com/zeromq/pyzmq/`) provides Python interfaces for ZeroMQ.

Both solutions can be used with the 13CFLUX2 libraries to realize performance-optimized ^{13}C-MFA simulations. Standalone third-party tools, such as Omix, access the SWF services (e.g., databases or 13CFLUX2) via web service interfaces. Conversely, third-party web services are easily accessed from Python or BPEL workflow programs. Typically, the realization of web service clients is trivial in most modern programming languages. For instance, the following code presents a Python program that accesses KEGG REST web services (Kanehisa and Goto, 2000):

```
1 import urllib3
2 url = 'http://rest.kegg.jp/info/kegg'
3 http = urllib3.PoolManager()
4 r = http.request('GET', url)
5 print("content: %s" % r.read())
```

Similarly, a SOAP web service client using SUDS can be reviewed in appendix B.5.

4.2.4. Cluster and Cloud Computing with Hadoop MapReduce

While clusters and grids continuously evolved to satisfy the ever increasing requirements of scientific computing in the last decades, only recently commercially successful cloud computing services emerged in a pervasive manner. By employing Amazon's cloud offering, the availability of virtually unlimited resources to perform computations with large-scale ^{13}C-MFA models is now in reach. The primary motivation is to provide robust and failure-tolerant cloud-aware ^{13}C-MFA applications using Hadoop MapReduce. Performance analysis of Hadoop MapReduce and its parameterization has been subject to various studies in different contexts (e.g., Zaharia et al., 2008; Jiang et al., 2010; Ibrahim et al., 2010; S. B. Joshi, 2012). In summary, these general design decisions are made for the SWF:

- **Design Decision 13:** Hadoop MapReduce is employed to perform large-scale simulation tasks.
- **Design Decision 14:** Amazon AWS (including EMR and S3) is employed to outsource simulations on virtual cloud resources.

The remainder of this paragraph discusses the design decisions made to integrate 13CFLUX2, and the choice of configuration parameters to optimally run ^{13}C-MFA simulations in Hadoop MapReduce environments.

4.2.4.1. Using ^{13}C-MFA Simulation Tools with Hadoop MapReduce

The Hadoop MapReduce framework is optimized to execute programs using plain strings as input and output. This *Hadoop MapReduce streaming API* is very convenient because

simulation jobs are realized using the simulation programs directly. The applicability of this approach is demonstrated on the example of a raw MS chromatogram alignment script in § 6.1. Although 13CFLUX2 simulation programs operate on XML string files, Hadoop MapReduce requires each data item in a specific format. While XML data is easily converted using Shell or Python scripts, there is a significant effect on the performance as discussed in § 6.5.

As an alternative to this straightforward approach, a family of Java programs (called *FluxHadoop* and *FluxHadoop2*) is developed as part of the SWF which employs the *Hadoop MapReduce Java API* (cf. § 5.2). 13CFLUX2 programs are accessed by Java applications via the API of the *FluxCore* library. In addition, this implementation allows the use of binary data types, e.g., HDF5. Both, the straightforward and the Java approach, have in common that the realization of Hadoop jobs is possible with minimal programming effort, while the original 13CFLUX2 software is untouched.

Design Decision 15: Two strategies to execute existing simulation tools with Hadoop MapReduce are realized in the SWF, where each approach has its own challenges and advantages. Depending on the concrete problem, the use of either the streaming API, or the native Java bindings to realize MapReduce jobs is advertised.

4.2.4.2. Configuring the Hadoop MapReduce Software for Simulation Tasks

The runtimes of 13CFLUX2 simulations are non-deterministic, while the MapReduce architectural design pattern is typically used to uniformly process large amounts of data (i.e., in terms of gigabytes or terabytes) (Dean and Ghemawat, 2004). ^{13}C-MFA models often require only several kilobytes of storage space, and currently even the most comprehensive models take up only a few hundred kilobytes. Fortunately, Hadoop MapReduce offers with more than 200 parameters a way to tune the runtime performance for our type of applications (White, 2009).

In particular, the following parameters have been selected to conduct 13CFLUX2 simulation tasks.

- `mapred.task.timeout`: by raising the default maximum task timeout from 10 minutes to a sufficient maximum (e.g., 24 hours), long-running simulations are no longer aborted by Hadoop.

- `mapred.tasks.speculative.execution`: by default, Hadoop MapReduce performs speculative execution of parallel tasks. While this compute-ahead feature is beneficial for pure data processing tasks, performance issues with long-running simulation tasks with Hadoop have also been identified elsewhere in the literature (Wall et al., 2010). Hence, speculative task execution is disabled for this kind of simulation job.

- Rather than processing huge amounts of data, our primary use of Hadoop is to elegantly parallelize a job on a distributed infrastructure. `mapred.map.tasks`, `mapred.reduce.tasks`, and `mapred.max.split.size` allow the optimization of the data segmentation (chunking) for the `map` and `reduce` processors.

In summary, the performance of 13CFLUX2 simulations can be improved by carefully adjusting the selected default configuration parameters of the Hadoop MapReduce framework.

4.2.5. Provenance Collection Framework

Because ^{13}C-MFA studies consist of a number of non-deterministic and in parts recursive workflow steps, the researcher needs to keep track of a plenitude of *metadata* including input data files (e.g., measurement data, models), output data files (e.g., simulation outcomes, logs), and other documents (e.g., related studies or lab protocols). The information that is needed to reproduce and understand a scientific workflow task is summarized with the term *provenance data* (Moreau, Groth, et al., 2008). This section discusses the requirements and design criteria of a software module that supports the tracking and organization of scientific workflow provenance data.

4.2.5.1. Requirements for a 13CFLUX2 Provenance Framework

The ability to capture, manage, and query metadata is one of the major advantages of using scientific workflow solutions in general (Davidson and Freire, 2008) and the SWF in particular (the results of the research are detailed in Dalman, Weitzel, Wiechert, et al., 2011). As pointed out, typical ^{13}C-MFA workflows consist of several modeling, simulation, and analysis steps. As with other parts of the SWF, the provenance collection framework is a custom-tailored solution with the focus on ^{13}C-MFA workflow applications using 13CFLUX2. By supporting the scientist in organizing and tracing provenance messages for each of these steps, the provenance data of the whole workflow is made accessible.

Notably, inspection of the provenance data in the SWF is not limited to a retrospective analysis. Besides monitoring the current status (is the task pending, running, or finished?) of the workflow execution on distributed computing environments, the online provenance approach for the SWF also allows for life monitoring of analysis results on demand. This is a very effective decision-making feature for the scientist who invokes compute-intensive ^{13}C-MFA applications because it can save significant amounts of time and money. For instance, long-running flux estimation processes may come to nothing, which can be detected by an experienced modeler through continuous inspection of the specific process data. Additionally, as unintended abortions of simulation tasks can happen at any time, it is especially useful to track such premature program exits (e.g., due to program errors, resource shortage, or modeling issues).

4.2.5.2. Software Design after the Provenance Life-cycle Model

The *provenance life-cycle model* describes the phases that an item of provenance data perambulates in a computer system (Moreau, Groth, et al., 2008). This model is taken as a design guide for the design of the provenance collection framework. It consists of four parts (cf. fig. 4.5):

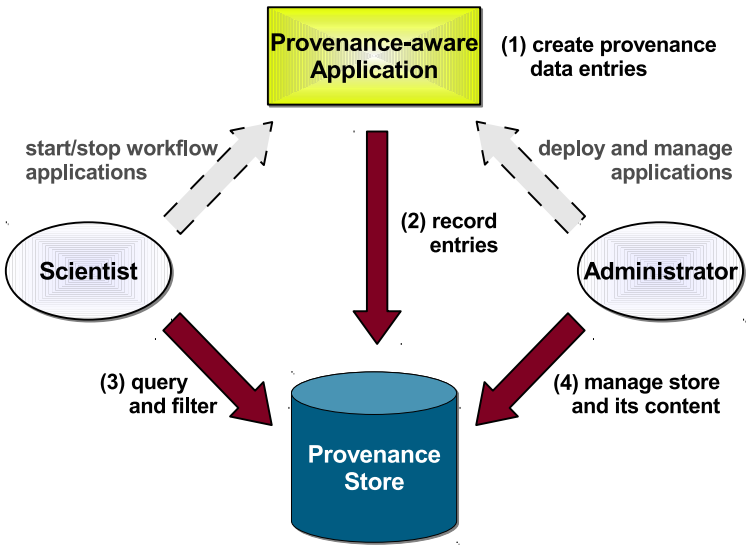


Figure 4.5.: The provenance life-cycle model consists of four phases: (1) creating, (2) recording, (3) querying, and (4) managing provenance data (Moreau, Groth, et al., 2008).

1. *Creating*: provenance data is created by a provenance-aware application. In the ^{13}C-MFA context, 13CFLUX2 tools (and other programs) need to be extended by the capability of generating provenance data.

2. *Recording*: the *provenance store* is an application that captures and manages provenance messages. Entries generated from provenance-aware applications are stored in-memory, or (if needed) persistently saved in a file.

3. *Querying*: after the recording, scientists often need to obtain selective information on the provenance data of specific items (e.g., log messages from a failed workflow step). Thus, the provenance store provides the ability to be queried and filtered for messages of interest. Eventually, the scientist also submits the extracted information of interest to the Subversion project repository.

4. *Managing*: the provenance life-cycle model distinguishes between user roles (working with scientific information in the store), and administrator roles (managing the store itself). While this differentiation is also important in the SWF approach (especially in a multi-user context), the actual persistence layer (and thus, the management of the store contents) is handled within Subversion repositories.

The provenance server collects metadata and log messages from simulation applications running on the compute nodes *online* (i.e., while running). The collected data can be

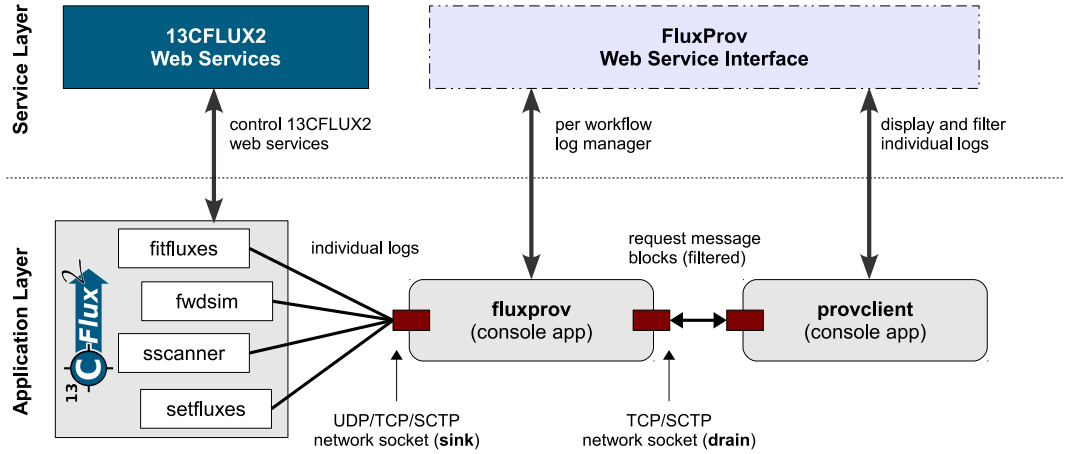


Figure 4.6.: The overall design of the log collector consists of two parts: a high-level web service API that is integrated into the ^{13}C-MFA workflow framework, and a low-level network-attached provenance framework. 13CFLUX2 command-line programs are accessed by a web service wrapper.

accessed by the scientist via a web-based interface. In summary, the following design decisions are derived for the provenance collection framework (cf. fig. 4.6):

- **Design Decision 16:** The design of the provenance collection solution of the SWF includes four basic building blocks following the provenance life-cycle model: (i) a library to emit provenance log messages (*fluxlog*); (ii) provenance store (*fluxprov*); (iii) a query client (*provclient*); and (iv) web service interfaces for managing provenance stores on demand.

- **Design Decision 17:** The framework is split in two layers. While the *service layer* offers standardized web service interfaces to other components of the SWF, the *application layer* is optimized for high-performance. Specifically, custom binary network transport protocols are employed for low-latency and high-throughput communication.

- **Design Decision 18:** Provenance stores in the SWF are private to the user which simplifies the overall design of the provenance collection framework. User management and persistence of provenance information is organized in Subversion project repositories.

- **Design Decision 19:** Several open source C or C++ logging libraries are available which support many features. For instance, Apache *log4cxx*[6] is a prominent logging

[6] See `https://logging.apache.org/log4cxx/`.

library with support for wide character Unicode and multiple inheritance-based logging contexts. However, existing libraries have a noticeable impact on performance (Synesis Software Pty Ltd, 2010). Hence, following the design of 13CFLUX2, a lightweight logging library is developed from scratch.

- **Design Decision 20:** 13CFLUX2 programs are extended by parameters to control the provenance collection behavior. This way, the scientist can decide whether a 13CFLUX2 task is to be monitored at the expense of runtime performance.

4.2.5.3. TCP, UDP, and SCTP Network Transport Protocols

Because typical 13CFLUX2 simulation workflows generate lots of log messages (especially when several simulation workflows run in parallel), it is worthwhile to optimize the performance of the provenance collection framework. In general, when a networking communication channel is highly loaded, the effect is observable in the transport layer, e.g., message losses or slow-downs are seen (Tanenbaum, 2002). While there are several transport layer protocols available in Linux and other modern operating systems, three of the most widely used protocols[7] with different usage characteristics are chosen for capturing provenance messages in the SWF (cf. table 4.4):

Features	UDP	TCP	SCTP
reliable transport	no	yes	yes
preserves ordering	no	yes	optional
connection-oriented	no	yes	optional
strict message boundaries	yes	no	yes

Table 4.4.: Comparison table of employed network transport protocols UDP, TCP, and SCTP. These protocols mainly vary in the reliability of the packet transport, whether messages arrive in the same order as sent, and whether an explicit connection between sender and receiver is required.

- **TCP** is a connection-oriented streaming protocol. Because the message transfer is reliable, TCP is employed as the default protocol in the provenance collection framework.

- **UDP** is selected in ^{13}C-MFA workflows where overall performance is more important than accuracy. For instance, it is tolerable to occasionally miss residual or flux values that are being monitored during a parameter estimation process (cf. § 6.7). Although the underlying UDP protocol gives no guarantees about the ordering of transmitted datagrams, the provenance framework ensures this by using timestamps in *fluxprov* and *fluxlog*.

[7] See also `https://en.wikipedia.org/wiki/Transport_layer`.

- **SCTP** is, despite being available for more than a decade on the Linux operating system, a lesser known alternative to both, TCP and UDP (R. Stewart, 2007). Specifically, SCTP addresses the reliability and blocking issues in TCP and UDP. In many private networking environments (such as the FZJ/IBG-1 IT infrastructure; cf. § 4.4), SCTP can be readily used as drop-in replacement for collecting provenance data in the SWF.

§ 5.3.5 discusses the performance differences of the three transport protocols with a use case employing 13CFLUX2.

Design Decision 21: TCP is chosen as default transport protocol for the SWF provenance collection framework. UDP and SCTP protocols are selectable as configuration parameter.

4.3. Presentation Tier

On top of the SWF, the presentation layer enables the scientist to compose, run, monitor, and steer workflow applications. To this end, services that provide static and dynamic web pages of the Internet domain `13cflux.net` are implemented in this layer. This way, this solution adapts well-established web technology that is also found in other general-purpose SWFs, e.g., Galaxy (Goecks et al., 2010) or Taverna (Hull et al., 2006) and specialized bioinformatics platforms such as MeltDB (Neuweger et al., 2008). The software design of the SWF web site is driven by the following requirements:

- Enable the presentation of information about research projects and related content to prevalent Internet web clients.
- Access to project-specific (or otherwise sensitive) web content needs to be protected by user-based login. At the same time, some information (e.g., imprint) need to be publicly accessible. Therefore, the web sites need to be designed with different *personas*[8] (i.e., with different access permissions and views on information) in mind.
- Because any internet web site is exposed to malicious attackers, dedicated security measures must be implemented to protect sensitive data such as project content and user credentials.

4.3.1. Software Components

It is common to distinguish between the *web server* and *application server* components, e.g., for security and performance reasons (Chopra, Li, and Genender, 2007). The following design decisions are made for the presentation layer:

[8] In this text, *persona* is used synonymously with the term *role*.

- **Design Decision 22:** The Apache HTTPD *web server* is the most widely used open source Internet server software[9] and is therefore chosen to handle internet web requests (Bowen and Coar, 2008). Appendix D summarizes the configuration details of the Apache HTTPD web server.

- **Design Decision 23:** The dynamic web content, i.e., research and non-scientific content (e.g., user-related credentials) is realized in the Java EE-compliant JBoss 6 *application server* (Rubinger and Burke, 2010).

4.3.2. Web Page Design of 13CFLUX.net

The FZJ corporate design is included by using a HTML template[10]. Each web page consists of a navigation menu (left), a page header (top) and footer (bottom). The page content is displayed in the remaining space in the center.

Figure 4.7 shows the front web page of the portal server. The corresponding source code is found in listing B.1 in appendix B. JSTL is employed to assemble HTML pages using the design template and to include server-side JSP tags. The following custom JSP tags are defined:

- `cms:page`: this is the root XML tag for the custom JSP content. The attribute `site` references the content settings XML file for the 13CFLUX.net web portal.

- `cms:pageAttr`: with this tag the page title, custom CSS, and JavaScript files can be specified.

- `cms:security`: page access is controlled using this tag. Setting the `boolean` attribute `requireLogin` to false allows anonymous access to the page contents. The attribute `requireRole` further restricts the access to the page to the specified group of users.

- `cms:securitySection`: while `cms:security` provides access control mechanisms for a whole page, this tag displays page sections depending on the user's login-state and role.

On client side, the pages are presented using HTML, CSS, JavaScript, and JQuery.

4.3.3. Content and Personas

The web portal of the SWF is used by different personas with varying user-interface workflows (Mulder and Yaar, 2006). Each user is assigned to one or more roles which are categorized as follows:

[9] See for example: `http://w3techs.com/technologies/cross/web_server/ranking/`; last accessed: February 9, 2016.

[10] The corporate identity design of the FZJ web sites from 2009 has changed in the meantime. Thus, the similarity between our web site and `http://fz-juelich.de/` is restricted to logos and color schemes.

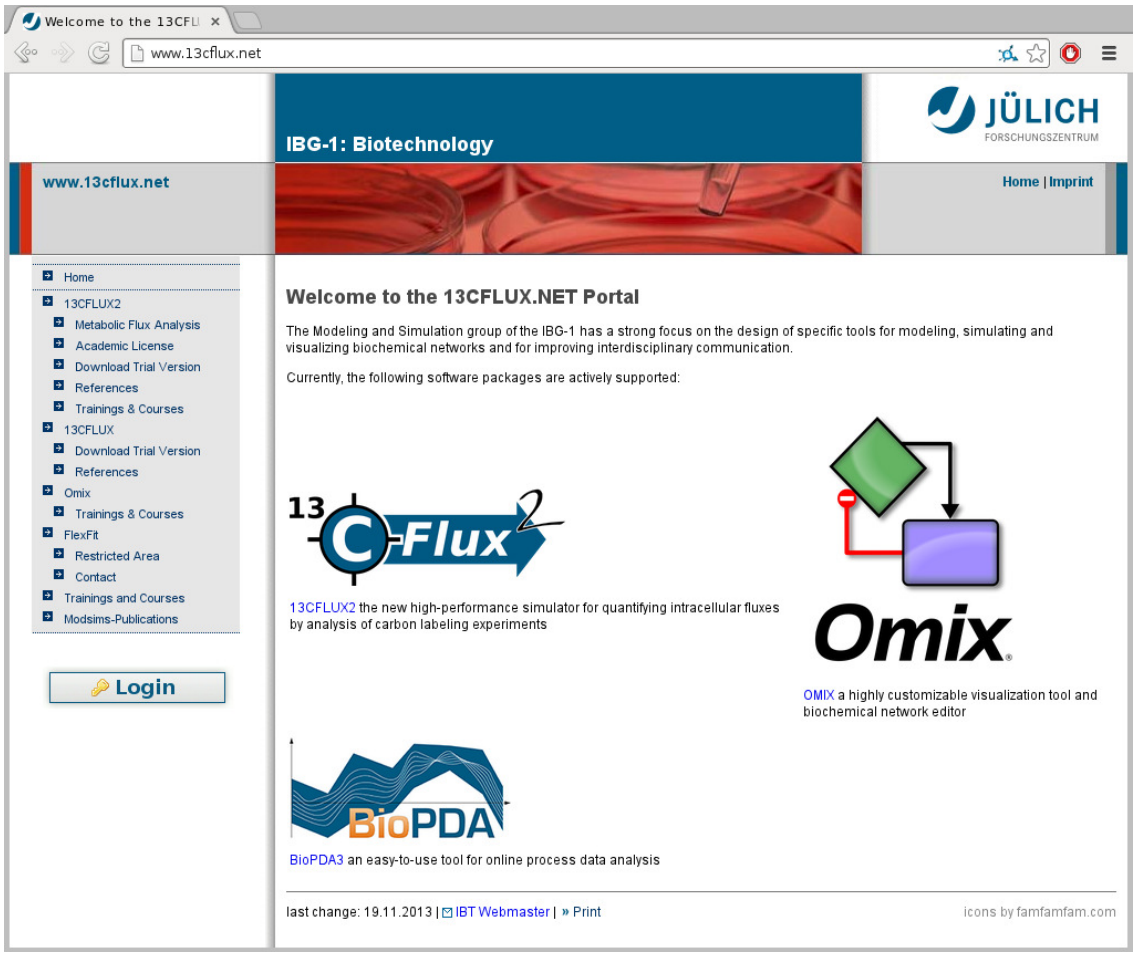


Figure 4.7.: Screenshot of the `www.13cflux.net` front web page.

- *Guests* visit the web site to obtain basic information about ^{13}C-MFA, publications, tools, or specific projects. This is the default role for any user, hence, all non-restricted pages are accessible by guests.

- *Scientists* have access to particular areas of the web portal. The `measurementdb-user` role, for example, is allowed to view contents of the measurement database.

- *Administrators* are permitted to perform tasks like user management.

The Java EE standard provides several facilities to realize a persona-oriented web portal.

Design Decision 24: Support for multiple roles that can be assigned to users. The basic roles are: guests, scientists, and administrators. More roles can be added on-demand, e.g., to control permissions for specific research projects.

4.3.4. Ticket-based Authentication System

Workflow applications can be defined by a series of independent (web service) invocations. To avoid a full authentication at each step, a suitable security mechanism is desired (Todorov, 2007). With *Stateful Session Beans*[11], Java EE provides a standardized way to handle session tickets. However, these beans are normally memory-resident (i.e., non-persistent) and designed to run a short time period (Jamae and P. Johnson, 2009). By contrast, scientific workflow applications are typically long-running, i.e., in the order days or weeks.

The *Central Authentication and Authorization System* (CAAS) is the ticket-based access and authentication component of the SWF web portal. It features the role-based ticket authentication system CAAS which is based on CAS[12], a single sign-on Java library. The `CAASHandler` EJB provides public interface methods to check the validity of a ticket, to prolong the ticket's expiry, and to get information about the ticket. The tickets are persistently stored in the CAAS database. Figure 4.8 depicts the database schema for the user management which consists of seven tables.

Design Decision 25: Realize a ticket-based authentication mechanism to grant access to web portal services. Extend the life-time of such an authentication ticket until the workflow execution is finished.

4.3.5. SVNKit Web Integration

The design of the Subversion repository access within the web portal using SVNKit is shown in fig. 4.9. Following the three-tier design, the user frontend for accessing and managing the repository is realized in several JavaServer Pages. The logic is implemented by two management (`ModelRepository` and `RepositoryManager`) and helper classes (`DirEntry` and `LogEntry`). The actual repository operations (e.g., creating a new file revision) is provided by the SVNKit library in the Java namespace `org.tmatesoft.svn.core`. The `net.x3cflux.repository.entity` interfaces the database EJBs to store repository meta information (such as user access mapping).

These repository operations are implemented in the web portal: browse the repository tree, download, modify, create new file or new file revision, and delete. By mapping the Subversion (SSH) user keys to CAAS credentials, the scientist is relieved from explicitly authorizing to the repositories.

Figure 4.10 depicts a web portal screenshot viewing the above template repository structure. With this web interface, a scientist accesses the repository content and history with a web browser.

[11] In the Java Enterprise context, a *bean* refers to an application running in the web application server context.

[12] CAS web site: `https://apereo.github.io/cas/`; last accessed: May 22, 2017

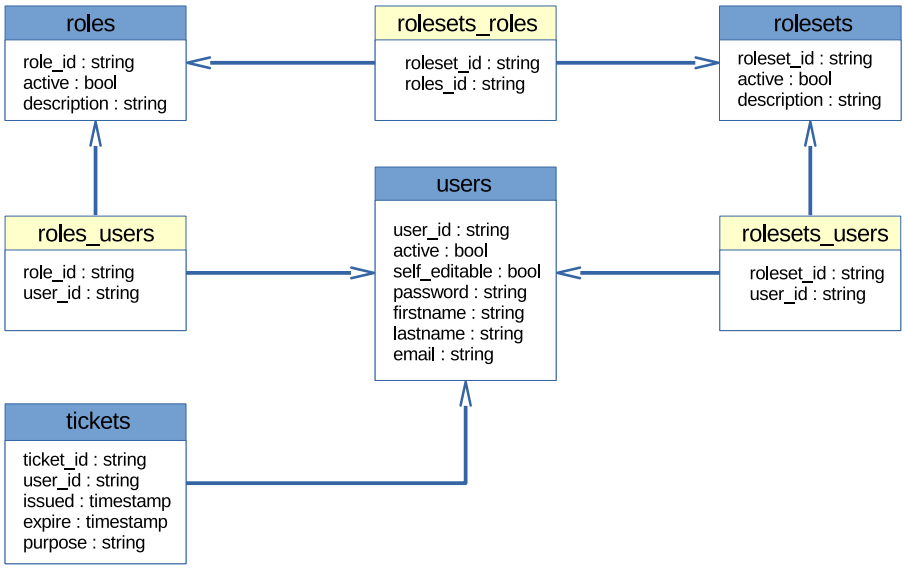


Figure 4.8.: UML diagram of the CAAS database schema. User-related data is stored in the `users` table. While the `roles` table associates a list of users to a role (via `roles_users`), the `rolesets` tables can arbitrarily group roles (via `roleset_roles`) and users (via `rolesets_users`) together. The `tickets` table is used to provide a time-limited authentication to a particular user.

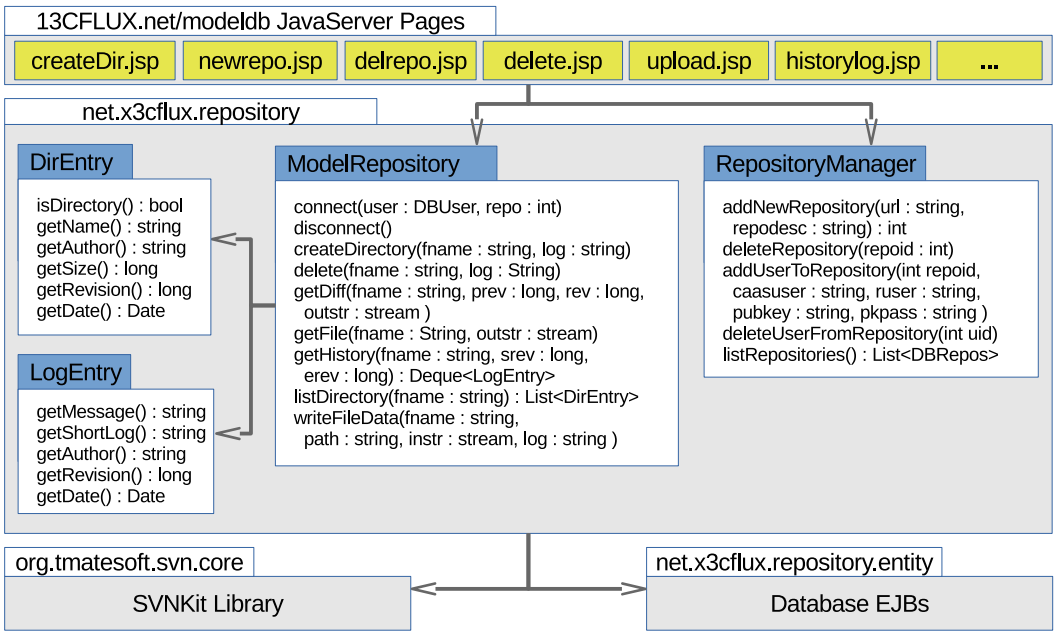


Figure 4.9.: Diagram of repository integration classes using SVNKit.

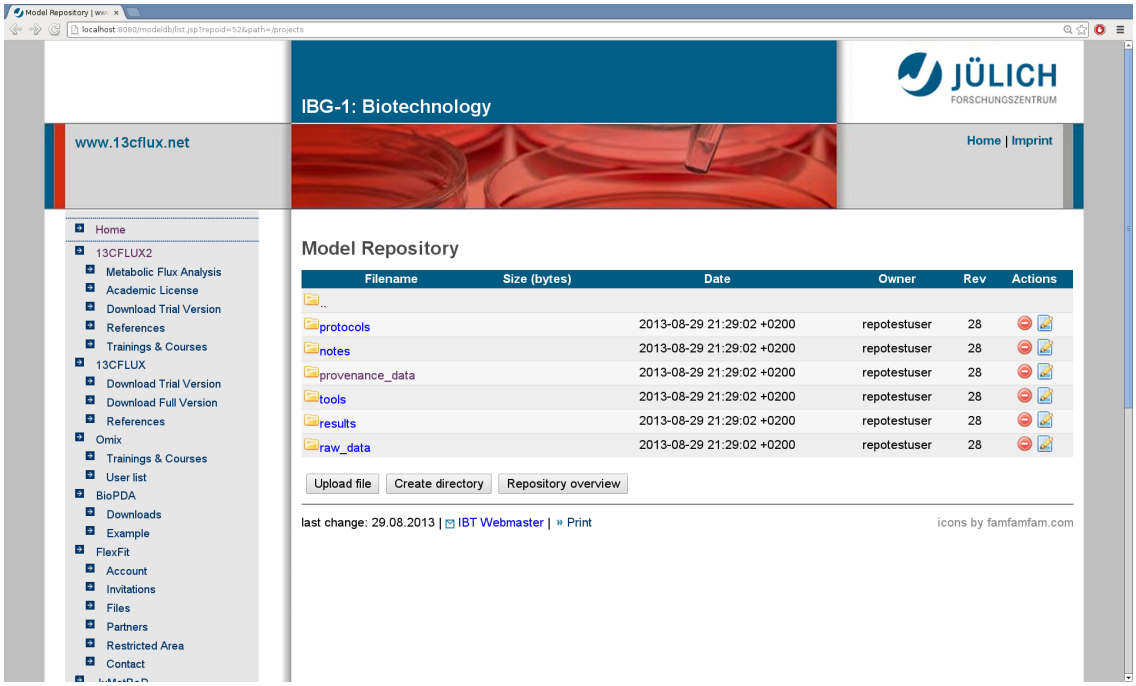


Figure 4.10.: Screenshot of the SVNKit web site. Using this interface, a scientist can download, add, update, delete, or read the changelog of a particular file from any web browser.

4.4. Deployment Considerations

The SWF is designed to be accessible from public Internet web clients. Therefore, various security and service reliability measures are identified and implemented in the SWF deployment architecture:

- **Secure Transport**: Because web servers are generally scanned by malware applications for open networking ports, all service accesses except HTTP and HTTPS are restricted to trusted hosts (i.e., from the intranet). To protect web-based communication between client and web service, the use of HTTPS is forced for the exchange of sensitive data[13] because *a security system is only as strong as its weakest link* (Ferguson, Schneier, and Kohno, 2010, page 5).

- **Virtualization**: While it is possible to deploy the SWF on a single host, the distribution of the services on multiple nodes offers advantages in terms of security, scalability, and service reliability. By separating publicly accessible Internet services from internal services on dedicated (virtual) machines, compromised hosts only have a confined impact on the whole system (Spector, 2000).

- **Service Reliability**: Service robustness and data integrity are crucial requirements for storage clusters like database systems and repository servers. When a node or a service fails, it is desirable to keep the downtime at a minimum. To ensure robustness, the cluster monitoring service Nagios[14] is employed. Thus, when a node fails, automatic recovery scripts are executed which restore the service operation. Data integrity is ensured by utilizing various levels of redundancy, i.e., by employing RAID systems, NAS storages, and external backup solutions (Hick and Shalf, 2009).

- **Performance**: The web server needs to be able to handle several hundred simultaneous HTTP requests per second. Because the Java Enterprise application server hosts the dynamic web site, client requests typically consume considerable amounts of CPU and memory resources. Typical web clients of the web site are malware bots, automatic web crawling robots, or actual users (Milstein, Biersdorfer, and MacDonald, 2006, chapter 8). Thus, to mitigate possible web performance issues, the web server is separated from the application service. Because adding compute nodes to the Hadoop cluster is easily possible, load bottlenecks in the simulation cluster are eliminated with new or additional hardware (Sammer, 2012). Alternatively, performance peaks can be handled by employing on-demand cloud computing resources, e.g., with ActiveBPEL (Dörnemann, 2013).

In summary, these design decisions are derived for the SWF deployment:

[13] The Electronic Frontier Foundation advocates the general use of HTTPS. Thus, they offer the free web browser plugin *HTTPS Everywhere* to prefer encrypted transport over plain HTTP on client side. The SWF provides HTTPS for all web sites. See `https://www.eff.org/https-everywhere/`; last accessed: May 22, 2017

[14] Nagios web site: `http://nagios.org/`; last accessed: May 22, 2017

- **Design Decision 26:** Implement basic security measures, i.e., use HTTPS for web clients and (REST) web services whenever possible, expose only necessary networking services, and grant access to data only for authenticated users.
- **Design Decision 27:** Use virtualization and redundancy techniques to minimize service downtimes. In addition, modern virtualization stacks employ Kernel-based Virtual Machine (KVM) or operating system level virtualization, which have practically no added performance impact (Newman, 2015, chapter 6).
- **Design Decision 28:** To ensure scalability and performance stability, split the services on different physical hosts (e.g., by using virtualization technology).

To demonstrate these considerations for the deployment of the scientific workflow framework, fig. 4.11 depicts an overview of the hardware infrastructure at FZJ/IBG-1.

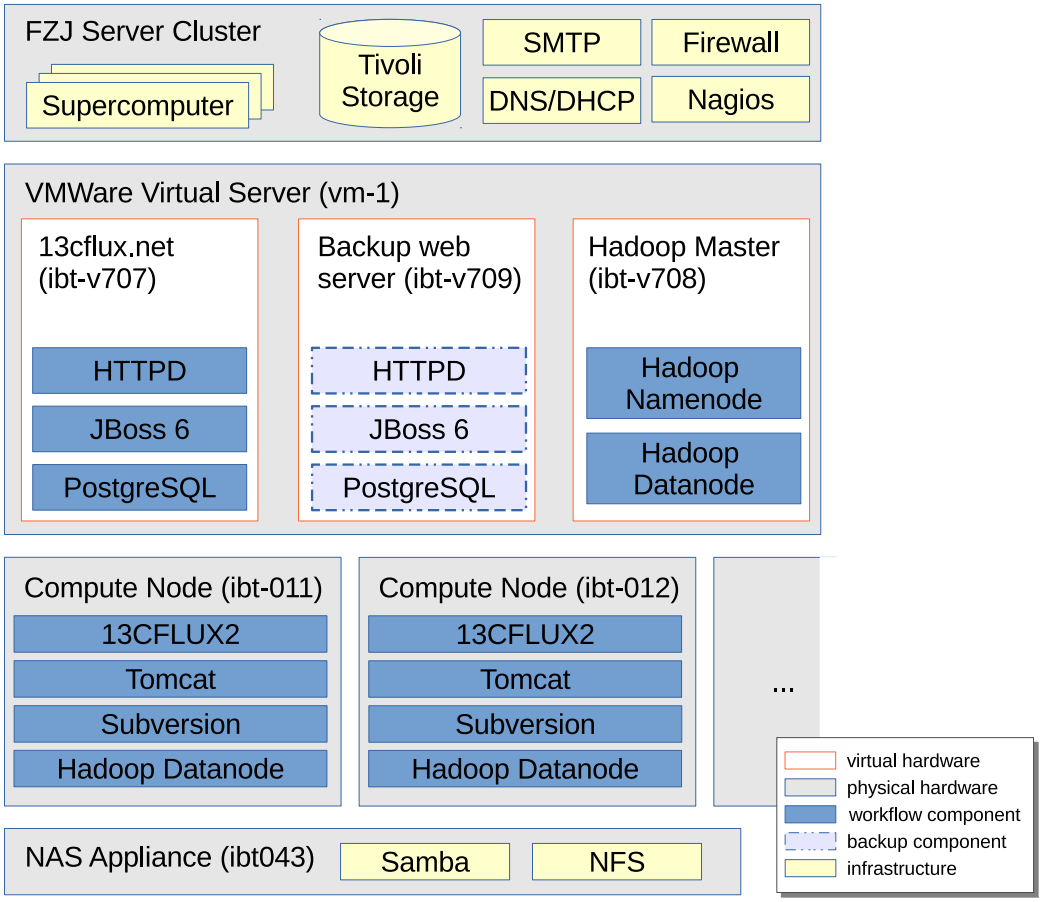


Figure 4.11.: Overview of the local cluster infrastructure at FZJ/IBG-1. The data center operates the supercomputer cluster and provides general IT services, i.e., backup, storage, networking, e-Mail, firewall, and monitoring. The SWF uses three virtual machines on a two-host VMware ESX server cluster: the web and application server (ibt-v707), a hot-standby backup server (ibt-v709), and the Hadoop master (ibt-v708). The IBG compute cluster currently consists of three identically configured cluster nodes (ibt-011, ibt-012, and ibt-013). 13CFLUX2, a Tomcat Servlet Container, a Subversion Repository, and HDFS distributed storage services are running on each node.

4.5. Chapter Summary

For the SWF, 28 design decisions are made that are valid for ^{13}C-MFA in particular and model-based metabolic engineering approaches in general. Taken individually, many of these design decisions are also found in other scientific workflow approaches (in fact, some decisions are inspired by existing solutions). The aggregation of all design decisions, which are derived from the computational challenges C1–C5 of ^{13}C-MFA applications, forms the overall design of the SWF.

A quantitative look at the distribution of the individual design decisions reveals that the SWF mainly focuses on the application tier, i.e., 15 decisions out of 28 are defined in the middleware layer. Conversely, the design of data-driven or choreographed workflow frameworks (which are popular especially in bioinformatics) have primarily emerged from data management[15] and visualization requirements (Curcin and Ghanem, 2008). Moreover, these (mostly monolithic) workflow frameworks integrate the scientific visualization as part of the system, whereas the SWF approaches the visualization of ^{13}C-MFA models and simulation outcomes as decoupled steps (though, the automation of visualization and simulation cycles is – at least in parts – still possible as shown in §§ 6.2, 6.4 and 6.8).

[15] For instance, most of the services listed in the Taverna Workbench User Manual are designed to perform data extraction, querying, and format conversion. See: `http://dev.mygrid.org.uk/wiki/display/tav250/User+Manual`.

Chapter 5.

Service-oriented ^{13}C-MFA Solutions

Due to its universality by design and its extensive functionality, 13CFLUX2 is deemed as the heartpiece of the SWF. Its modular and "headless" architecture eases the composition of computational tasks to larger ^{13}C-MFA workflows. However, to transfer the complete ^{13}C-MFA procedure to the SWF, the aforementioned challenges need to be tackled, namely, data organization (C1), standardization of tools and data (C2), interactive workflow steering (C3), distributed computing (C4), and service orientation (C5). This chapter displays how the SWF addresses these challenges by employing a service-oriented approach.

By realizing Java and web service interfaces, a modern and standardized method to access 13CFLUX2 programs is provided by the SWF (§ 5.1). With these interfaces, computationally demanding jobs are deployed on cluster and cloud computing resources using the Apache Hadoop MapReduce framework (§ 5.2). § 5.3 presents the provenance collection services, a framework that enables the gathering of metadata. The SWF document storage concept improves the traceability and reproduction of ^{13}C-MFA studies in general (§ 5.4). Finally, auxiliary visualization, simulation, and data conversion tools complement the 13CFLUX2 services in combination with the SWF (§ 5.5). Figure 5.1 depicts the tools that are proposed in this work to realize the steps of the ^{13}C-MFA procedure. Although 13CFLUX2 plays a central role in the SWF implementation, the underlying ideas are readily transferable to other ^{13}C-MFA tools that provide appropriate APIs.

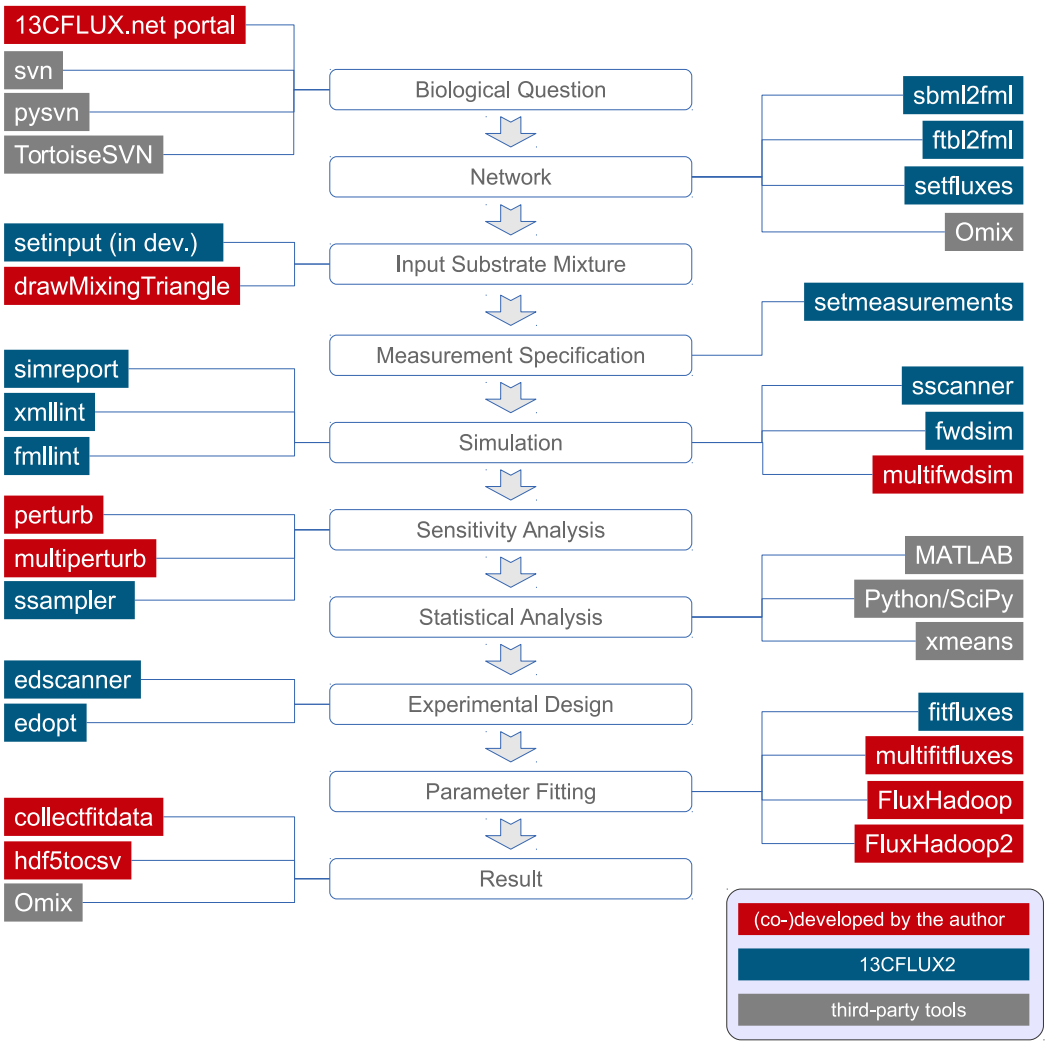


Figure 5.1.: The ^{13}C-MFA workflow – revisited. Software used in this thesis to realize typical ^{13}C-MFA applications include the 13CFLUX2 toolbox (blue boxes), third-party tools (gray boxes), and newly developed programs (red boxes). The central arrows indicate principal transitions from ^{13}C-MFA workflow steps (rounded boxes), while the straight connector lines show the typical utilization of a tool within a particular workflow step. Some tools are used in more than one workflow step, however, in the drawing software components are only drawn once (Omix being the exception due to its dual role as modeling and visualization tool).

5.1. Making 13CFLUX2 Service-oriented

To use existing programs in composite applications, they need to provide suitable interfaces to communicate within the SWF. In this work, the *FluxCore* and *FluxWS* libraries implement lightweight Java and web service interfaces for legacy programs. On the example of 13CFLUX2, the adequacy of the taken approach to make legacy command-line tools service-oriented (and thus, to integrate them into the SWF) is described.

5.1.1. Extending Legacy Tools with Java Interfaces using FluxCore

A command-line program is understood as a user-space process which performs a non-interactive computation task (Peterson and Silberschatz, 1985). Linux command-line programs usually interact with the operating system environment (e.g., the login shell) via the following *standard I/O* conventions (Love, 2013):

- *Standard input stream* is used by the program to read application data (such as file content).
- Program output text[1] is written to the *standard output stream*.
- User text messages are written to the *standard error stream*.

All 13CFLUX2 programs are designed to follow these conventions. Because the *standard input* and *standard output* streams of 13CFLUX2 programs are compatible to each other (i.e., FluxML or FWDSIM formats are used), the composition of these tools to simple workflows is easily possible, e.g., using the *pipes and filters* architectural design pattern (Hohpe and Woolf, 2003). To extend command-line programs with a Java interface using the *FluxCore* library, a constructor (realized as static initializer method) and a configuration method are implemented (cf. fig. 5.2). This way, the *FluxCore* library provides a variety of features:

- *Extensibility*: by inheriting from the `FluxServiceBase`, the access to command-line parameters which are common to all 13CFLUX2 programs (e.g., logging options) is centralized for all 13CFLUX2 program classes. Likewise, the internal handling of a program (e.g., starting, stopping, setting input and output streams) is realized in the abstract `ServiceBase` class. Thus, Java interfaces for command-line programs are easily added to the SWF by inheriting from `ServiceBase`.
- *Virtual workspace*: the heart of the *FluxCore* library is `ExecutionContext`. This class manages a virtual workspace for each active program, including input and output files, and an error code if the program has terminated. Because the virtual workspace is backed by a dedicated folder on the filesystem, it may be used to exchange file data between subsequent workflow steps.

[1] Some programs, such as *GNU tar*, also support writing binary data to the console output to allow "chaining" of programs using pipes.

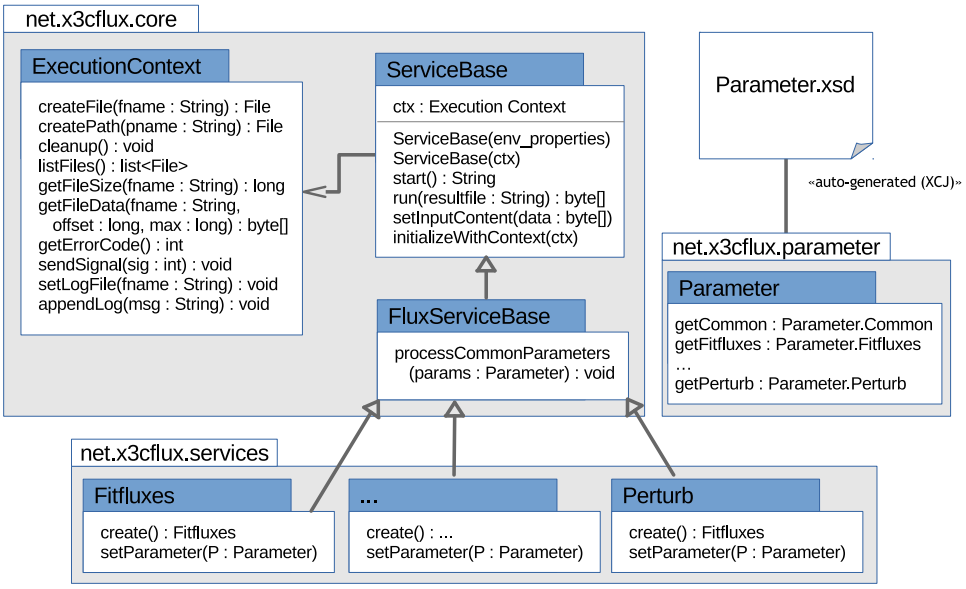


Figure 5.2.: UML class diagram of the *FluxCore* library. Each 13CFLUX2 program is wrapped by a Java class from the `net.x3cflux.services` package (bottom). These classes inherit from a more generalized execution and processing engine in the `net.x3cflux.core` package (top). Thus, the integration of new command-line tools is comparable easy to realize. The command-line arguments are automatically generated from an XML Schema file which eases the utilization of workflow arguments (right).

- *Signal handling*: by utilizing the standard Linux operating system signal handling (i.e., POSIX signals), running applications may be immediately terminated (SIGKILL) or notified to gracefully interrupt the processing (SIGINT). Many command-line programs already implement a set of signals. For example, interrupting the execution of a console program with control-C, the SIGINT signal is passed by the Linux operating system to the program (Love, 2013). In *fitfluxes*, the current parameter optimization iteration is aborted when SIGINT is received and a FWDSIM file is printed based on the last valid optimization values.

5.1.2. Web Service Interfaces for Long-running Simulation Tasks

Services (and in particular web services) are understood as a standardized and interoperable set of interfaces which expose functions of a software component (Josuttis, 2007). Because web services are typically stateless, the execution of (potentially long-running) simulations has two consequences:

1. The web service call itself is blocking, i.e., the web service call by the client will return after the simulation is finished.

2. State management has to be realized on top of the web service execution.

The Web Services Resource Framework (WSRF) addresses the realization of industry-grade standardized stateful web services and, thus, is ideally suited to work with other WS-* standards such as WS-Addressing or WS-Notification (Banks, 2006). Notably, WSRF is also used together with BPEL (Dörnemann, Smith, and Freisleben, 2008). § 5.2.3.2 presents an example where WSRF is used in a BPEL workflow.

However, well-tested and easy-to-use open source WSRF libraries outside the Java programming language are (to the author's best knowledge) rather uncommon. To fill this gap in the SWF, *FluxWS* is developed as a lightweight library to realize stateless and stateful web services using plain SOAP. As an outstanding feature, *FluxWS* provides a custom-tailored parameterization concept for configuring 13CFLUX2 web service application. Exemplarily, the realization of the *Fitfluxes* web service within *FluxWS* is shown at the end of this paragraph.

5.1.2.1. FluxWS Web Service API

FluxWS provides two SOAP web service interfaces to invoke 13CFLUX2 programs:

1. *Synchronous calls* are blocking until the web service processing is finished (in which case the standard output result from the *FluxCore* call is returned), or an error occurred and a `SOAPFault` exception is returned.

2. *Asynchronous calls* immediately return with a string-based handle which refers to a unique `ExecutionContext` entity. In the context of *FluxWS*, this string is called `JobTicket`. A web service client may perform the following operations on the web service using the `JobTicket` object:
   - `delete`: stop the job with a POSIX `SIGKILL` signal and remove the execution context data.
   - `stop`: send a graceful job termination signal (realized as POSIX `SIGINT` signal).
   - `isRunning`: determine the job execution status.
   - `listFiles`: return a list of file names in the execution context.
   - `getFileData` and `getFileSize`: return file data.
   - `waitFor`: wait for a specified amount of time until the job is terminated. This method synchronizes the client-side execution of the service call.

The utilization of *FluxWS* using a Python SOAP web service client with the SUDS library is demonstrated in appendix B.5.

5.1.2.2. Workflow Parameterization Concept of FluxWS

While 13CFLUX2 programs are parameterized independently from each other in a composite simulation, some program arguments have in practice the same value for all tasks within a workflow run. The selection of FluxML configurations, the provenance collection behavior, and logging parameterization are identified as common 13CFLUX2 program arguments. In addition, several arguments are shared between simulation programs, e.g., optimizer parameters and solver configuration are uniformly defined in 13CFLUX2. *FluxCore* provides a consistent mapping to ease the parameterization and composition of complex workflow applications:

$$S : (P, C) \mapsto R$$

Here, the 13CFLUX2 service S maps a XML-based parameter string C and the service parameter P to the service result R. Typically, P is a FluxML model and R is XML or HDF5 data comprising simulation data. The workflow-specific configuration C is passed throughout each workflow step. Using this parameter XML document[2], 13CFLUX2 programs are configured individually. This parameterization concept leverages the composition of ^{13}C-MFA workflows with stateless web services.

5.1.2.3. The *Fitfluxes* Web Service

To demonstrate the lightweight character of *FluxWS* and the underlying *FluxCore* libraries, the differences of the synchronous and asynchronous realization of the `Fitfluxes` web service are shown. The synchronous `Fitfluxes` service endpoint implementation consists of 11 lines of code (comments, logs, and error handling omitted):

```
1  public byte[] Fitfluxes(byte[] fmlcontent, Parameter P) throws Fault {
2    Fitfluxes srv = Fitfluxes.create();
3    srv.setInputContent(fmlcontent);
4    srv.setParameter(P);
5    try {
6      return srv.run("stdout");
7    }
8    finally {
9      srv.getExecutionContext().cleanup();
10   }
11 }
```

Line 2 instantiates the `Fitfluxes` service object. After passing the input data (line 3) and workflow parameters (line 4), the program is run with `stdout` as program output file name in the execution context (line 6). This is a synchronous program execution, hence the result data is immediately returned and the execution context is cleaned up (Line 9).

Besides the common asynchronous web service functions (e.g., `waitFor` or `stop`), the specific `Fitfluxes` realization for this API consists of a "start" method:

[2]The `FluxCore` parameter XML format is defined in the following XSD schema: `http://13cflux.net/fluxparameter`; last accessed: May 22, 2017

```
1 public JobTicket FitfluxesStart(byte[] fmlcontent, Parameter xmlparams) throws
      Fault {
2   Fitfluxes srv = Fitfluxes.create();
3   srv.setInputContent(fmlcontent);
4   srv.setParameter(xmlparams);
5   String ctx_id = srv.start();
6   JobTicket ticket = new JobTicket(ctx_id);
7   return ticket;
8 }
```

While the `Fitfluxes` call blocks until the simulation is finished, `FitfluxesStart` immediately returns with a `JobTicket`. The identifier of the execution context is retrieved after asynchronously starting the task execution (line 5). With this context identifier, a `JobTicket` instance is created and returned to the caller (lines 6-7).

5.2. 13CFLUX2 in the Cloud with Apache Hadoop MapReduce

This section presents the solution of this thesis for deploying computationally demanding ^{13}C-MFA simulations on cloud computing resources using Hadoop MapReduce. On the example of 13CFLUX2, our primary simulation tool in the ^{13}C-MFA environment, three realized variants of the MCB algorithm are presented: a straightforward approach which calls 13CFLUX2 programs from the command-line, an improved approach (called *FluxHadoop*) using *FluxCore* and the Hadoop Java API, and a specialized version (called *FluxHadoop2*) that utilizes local CPU cores of a compute node. The general life-cycle of a 13CFLUX2 simulation in a Hadoop MapReduce cloud setup is described first.

5.2.1. Life-Cycle of a 13CFLUX2 simulation using Hadoop MapReduce

In the SWF realization, the simulation life-cycle with Hadoop MapReduce in Amazon's cloud consists of five steps:

1. *Upload scientific data to the S3 cloud storage.* The cloud storage is used to exchange data between the SWF and the EMR service. Amazon provides a variety of ways to upload scientific data, i.e., a web GUI, a command-line program, and a REST web service interface as part of the AWS SDK[3].

2. *Reserve virtual Hadoop cluster resources.* The on-demand initialization of virtual resources includes the specification of the node type, the number of virtual nodes, and the selection of an operating system image. For our purposes, the official Debian *Amazon Machine Image* (AMI)[4] is chosen as base operating system for all simulation tasks in the cloud.

[3] See `https://aws.amazon.com/tools/`.
[4] See `https://wiki.debian.org/Cloud/AmazonEC2Image/`.

3. *Virtual machine initialization (bootstrapping).* Because the AMI is a general-purpose Linux operating system, the specific software tools (e.g., 13CFLUX2) need to be installed during the VM startup. This process is called *bootstrapping*. The AWS credentials to access the scientific data from the S3 storage are as well installed by the bootstrap startup script.

4. *Perform the simulation job.* The input data is split into chunks and processed by the Hadoop software. Because the data chunks are pair-wise independent, the user-specified `map` and `reduce` programs can be invoked on all available VMs in parallel.

5. *Retrieve the workflow outcome.* After the job is finished, the output data is placed at the specified URN for further processing by subsequent workflow tasks. With the exception of log files, all allocated resources are disposed by the Hadoop framework. These logs can be inspected by the scientist in case of a Hadoop failure.

While Amazon cloud resources require the on-demand initialization of the VM resources, local Hadoop clusters are readily available to perform simulation jobs.

5.2.2. Straightforward MCB with 13CFLUX2 and Hadoop MapReduce

A straightforward approach to realize the MCB algorithm with Hadoop MapReduce is to use references to FluxML model files in the Hadoop distributed storage (HDFS) instead of passing file content as values. The deployment script performs five steps to prepare and deploy the job in the Hadoop cluster:

1. Create directories on the HDFS storage with the command `hadoop -dfs -mkdir fmlfiles`.

2. Prepare FluxML files with randomized samples locally using the 13CFLUX2 programs `ssampler`, `perturb`, and `setfluxes`.

3. Upload the FluxML documents with the command `hadoop dfs -put fmlfiles ${HDFSROOT}/fmlfiles`.

4. Invoke Hadoop MapReduce with this command:

```
hadoop jar hadoop-streaming.jar
   -input "${HDFSROOT}/fmlfiles"
   -output "${HDFSROOT}/tmpoutput"
   -mapper "mc_mapper"
   -reducer "mc_reducer"
```

   `mc_mapper` and `mc_reducer` are simple Python wrapper scripts that invoke `fitfluxes` and `collectfitdata`. For example, the key-value pair `(0,`

`${HDFSROOT}/model0.fml`) is passed to `map`. Likewise, the intermediate pair (`0`, `${HDFSROOT}/fwd0.fml`) is processed by `reduce`, which eventually returns (`0`, `${HDFSROOT}/result0.hdf5`).

5. The computed result is downloaded from the HDFS storage with the command `hadoop dfs -get ${HDFSROOT}/collect.hdf5 collect.hdf5`, while the results of failed computations are dropped.

When the map tasks are distributed within a network of nodes (step 4), the communication overhead has to be taken into account (Pacheco, 2011). Therefore, to balance between the number of parallel tasks and the expected computation of a simulation, the granularity parameter G is introduced to the MCB algorithm, i.e., every task computes G Monte Carlo iterations (cf. algorithm 3). To keep the implementation as simple as possible, only one `reduce` task per Hadoop job is employed, however, this as a significant effect on the performance (cf. § 6.5).

Algorithm 3 MCB algorithm with Hadoop MapReduce utilizing a granularity parameter. The key difference between the original `map` function is the outer loop which processes $G \geq 1$ Monte Carlo iterations. The Hadoop MapReduce function `context.write` is called to emit a key-value pair for each iteration. The keep the code readable, the error-handling has been omitted. The error variable `SUCC` is set to FALSE if either *fitfluxes* or *setfluxes* fails.

```
MAP(INFNAME, FML)
 1  for k ← 1 to G
 2       do
 3          ▷ Model with perturbed measurements
 4          FML_P ← perturb(FML)
 5          for j ← 1 to M
 6               do
 7                  ▷ Parameter estimation
 8                  FWDSIM ← fitfluxes(setfluxes(FML_P, SAMPLES[j]))
 9                  ▷ Write FWDSIM result to shared folder with unique name
10                  FNAME ← makeFileName(INFNAME, j, k)
11                  storeFile(FWDSIM, FNAME)
12                  ▷ SUCC indicates whether fitfluxes was successful
13                  context.write(SUCC, FNAME)
```

5.2.3. FluxHadoop: Improved Implementation of the MCB

Beside the tuning parameter G, two aspects of the straightforward streaming implementation critically influence the parallel execution of the MCB algorithm. Firstly, using file

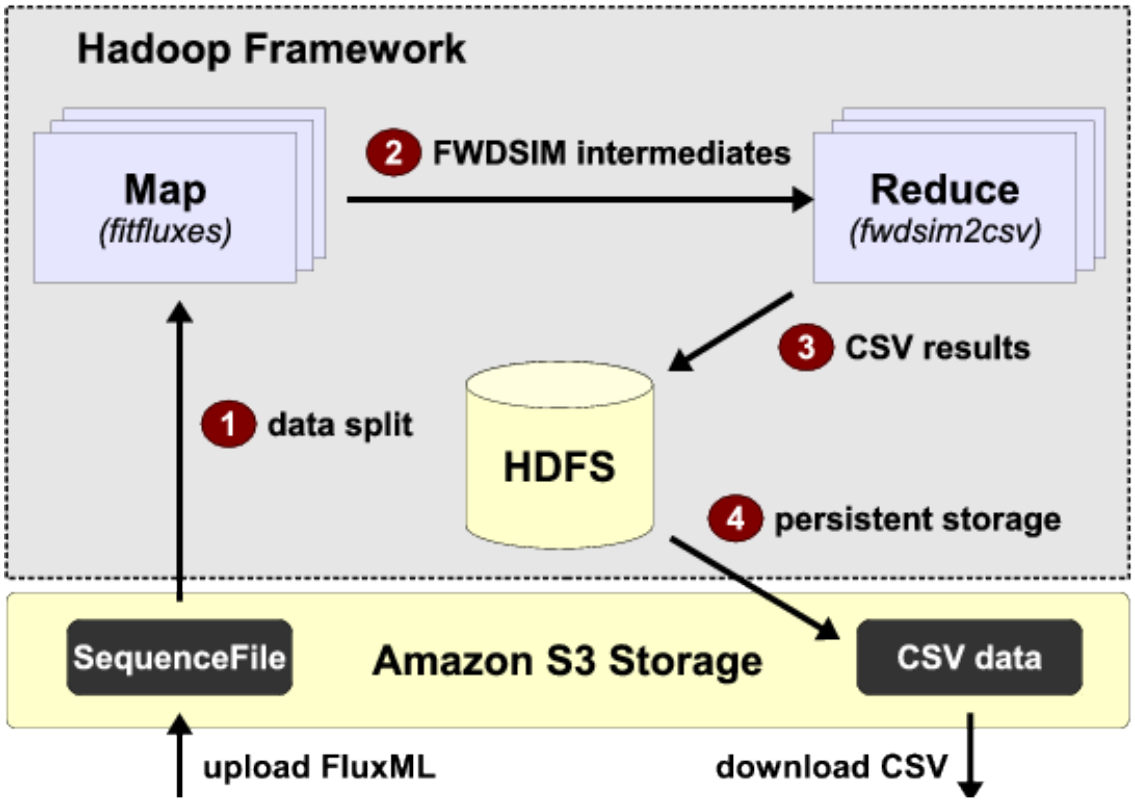


Figure 5.3.: *FluxHadoop* architecture overview. *FluxML* documents are stored in the Amazon S3 repository. Data is passed through the Hadoop framework in four steps: (1) Data splits from the sequence file are passed to the `map` tasks. (2) *FWDSIM* files generated from *fitfluxes* are passed to the `reduce` tasks. (3) CSV results are written to the HDFS storage. (4) The CSV data is transferred back to S3, where the user can access the result files.

references (which consume only small amounts of storage) causes many read and write operations on the Hadoop MapReduce filesystem HDFS (Sammer, 2012). Notably, HDFS is by default configured to replicate 64 MB chunk blocks on three HDFS nodes, i.e., three blocks are completely written for every write operation. Secondly, employing only one `reduce` task per Hadoop job limits proper scaling of the MCB implementation. Clearly, Hadoop jobs with a large number of nodes will suffer from a noticeable performance loss[5]. Addressing these issues of the MCB streaming version, a new Java-based implementation called *FluxHadoop* is provided with three major improvements (cf. fig. 5.3):

[5] The question of optimally partitioning of Hadoop jobs into suitable chunks is discussed on the Hadoop MapReduce wiki: `http://wiki.apache.org/hadoop/HowManyMapsAndReduces`; last accessed: May 22, 2017

1. The FluxML files generated in the preparation phase of the algorithm are packed into the Hadoop-specific input format `SequenceFile` using the original file name as key and the file content as value. The *HadoopPacker* program combines multiple FluxML files into a single sequence file.

2. Because *HadoopPacker* creates native `SequenceFile` objects, the data file is directly readable by the Hadoop framework. After performing the computation, the mapper (using *fitfluxes* via FluxCore) emits the original file names as input keys, along with the corresponding *FWDSIM* result data as intermediate pairs. Because *fitfluxes* and *fwdsim2csv* employ standard input and standard output streams, the realization of *FluxHadoop* benefits from a reduced number of I/O operations in the HDFS storage (Miner and Shook, 2012).

3. Compared to merging multiple binary HDF5 files is a complex operation, the concatenation of CSV files is simple. Therefore, the reducer is implemented by the 13CFLUX2 program *fwdsim2csv* that emits CSV files consisting of lines of comma-separated values of interest in the format: error-code, input file name, residual values, and estimated flux values. In case the subsequent workflow steps require the data format to be HDF5, CSV files can be easily converted using h5py (Collette, 2013).

5.2.3.1. Realization of FluxHadoop

To illustrate the interfaces and interactions between 13CFLUX2 and Hadoop, implementation details of *FluxHadoop* are presented in the following. The Java package *FluxHadoop* consists of the classes `MCBMain`, `Map` and `Reduce`. Implementing the Hadoop `Tool` interface, `MCBMain` is responsible for setting up the Hadoop job. Specifically, input and output types are assigned with the methods `setInputFormatClass`, `setMapOutputKeyClass`, `setMapOutputValueClass`, `setOutputKeyClass` and `setOutputValueClass`. In *FluxHadoop*, the `Map` Java class performs the actual `map` operation with an input key/value pair. The computation is executed by the following code fragment:

```
String xmlpath = context.getConfiguration().get(paramfile);
Parameter fluxparams = FluxParameter.getParameter(xmlpath);
Fitfluxes flux = new Fitfluxes(value, fluxparams);
byte[] res = flux.run();
```

In the SWF, 13CFLUX2 is parameterized with an XML file. Using the Hadoop configuration mechanism, the parameter file (`paramfile`) contains the constant string `net.x3cflux.hadoop.mcb.fluxparamfile`. With the *HadoopJarStepConfig*, the file *fluxparameter.xml* is passed to the Hadoop framework with the Amazon SDK. The *EMRinvokeHadoop.sh* script is used to pass this parameter as argument to Hadoop.

With `context.write(key, value)` at the end of the *Map* class, the result of this operation is emitted to the Hadoop framework, where *context* is an instance of the Hadoop

`Context` class. Likewise, the `Reduce` class is implemented using the 13CFLUX2 Java wrapper class `FWDSIM2CSV`. An error-code (i.e., 0 for success, a negative number otherwise) and the processed FML file name is prepended to the output CSV data. The Hadoop framework transfers CSV files from finished `reduce` tasks to the persistent S3 storage. The number of output files corresponds to the number of `reduce` tasks.

5.2.3.2. Monte Carlo Bootstrap Master Application on Amazon EMR

Figure 5.4 presents *MCMapReduce*, a BPEL realization of the MCB algorithm using Amazon's EMR cloud service. These Java classes are realized in *MCMapReduce*:

- `AmazonElasticMapReduceClient` is responsible for setting up and configuring the EMR cloud resources. AWS credential access and compute nodes are configured, i.e. number and type of nodes are specified. The Amazon SDK provides a comprehensive REST API to perform this task.
- EMR jobs are configured and executed with the `RunJobFlowRequest` class.
- The API class `HadoopJarStepConfig` is utilized to define *FluxHadoop* as implementation of the Hadoop `map` and `reduce` functions.

Beside initializing the cloud resources and defining the EMR job steps, `MCMapReduce` provides a WSRF web service interface. After starting a MCB job, *MCMapReduce* polls the Amazon service for state changes. The workflow then registers as a state change listener, being automatically notified using WS-Notification when the services have finished computation (fig. 5.4, step 5). In the meantime, the workflow (or workflow branch if there is more than one) is suspended.

5.2.4. Hybrid-Parallel Parameter Estimation

In recent years, the number of cores per CPU has drastically increased, and this trend is likely to continue (Pacheco, 2011). Although Hadoop MapReduce not only distributes computational tasks on cluster nodes, but also on all cores of a node, $N \times M$ *fitfluxes* tasks are scheduled for the MCB algorithm with *FluxHadoop*. This has two consequences on the performance: firstly, each *fitfluxes* call is traversed through a stack of Java code (i.e., the *FluxCore* library, the Hadoop MapReduce Java API, the Hadoop TaskTracker, and the Java Virtual Machine) which adds increased runtime and memory usage (Sammer, 2012). Secondly, each *fitfluxes* has an overhead of computational tasks for loading the FluxML file, verifying the integrity of the standard equations, computing the matrices, and (after finishing the parameter estimation) writing out the FWDSIM XML files.

Both problems are addressed by *FluxHadoop2*, which employs *multifitfluxes*, a multi-process variant of *fitfluxes* (§ 5.5.3 provides a detailed description of *multifitfluxes*). *FluxHadoop2* introduces the configuration option K as the number of simultaneous parameter estimation processes per `map` task. Hence, by combining Hadoop MapReduce with a shared memory parallelization approach, only $N \times M/K$ tasks need to be scheduled

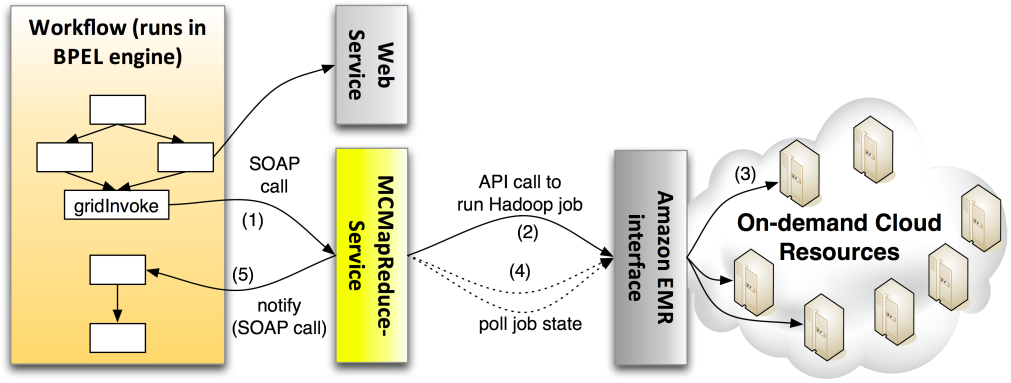


Figure 5.4.: Embedding the Hadoop MCB algorithm into a BPEL workflow. A workflow executed in a BPEL engine invokes the MCMapReduce service via a SOAP call (1). Using the AWS API, a Hadoop job is started (2) and the job is executed on cloud computing resources (3). The MCMapReduce service polls for the completion of the simulation job (4) and eventually returns a SOAP notification (5).

by the Hadoop framework. Each `map` task performs M multi-start optimizations, thus, $K \leq M$ and $K \geq 1$ tasks are executed in parallel on a node. Because *multifitfluxes* emits M flux vectors as HDF5 formatted data, these intermediate outputs need to be converted to CSV results. Therefore, a different converter is employed to implement the `reduce` step (i.e., *hdf5tocsv* instead of *fwdsim2csv*; cf. § 5.5.5). Table 5.1 summarizes the findings of the three introduced MCB realizations using the Hadoop MapReduce framework. In § 6.6, the performance improvements of *FluxHadoop2* are presented.

	Streaming	FluxHadoop	FluxHadoop2
13CFLUX2 binding	Python	FluxCore	FluxCore
`map` program	fitfluxes	fitfluxes	multifitfluxes
`reduce` program	collectfitdata	fwdsim2csv	hdf5tocsv
Number of mappers	$N \times M$	$N \times M$	$N \times M/K$
Number of reducers	1	≥ 1	≥ 1

Table 5.1.: Comparison of the MCB realizations with Hadoop and 13CFLUX2. Only one reducer is employed in the straight-forward streaming version. Because the hybrid version employs k cores per node using `multifitfluxes` instead of `fitfluxes`, only $N \times M/K$ `map` tasks need to be scheduled by the Hadoop framework.

5.3. Provenance Collection Services

In the SWF, the collection of provenance data can be controlled by employing the four basic building blocks of the framework in ^{13}C-MFA workflows: (i) the *fluxlog* library, (ii) the provenance store *fluxprov*, (iii) the provenance query client *provclient*, and (iv) a web service interface for managing provenance stores on demand. This section describes how to integrate and use these services efficiently in ^{13}C-MFA workflows. Because the SWF provenance collection components allow the operation with TCP, UDP, or SCTP, suggestions for choosing the optimal transport protocol is given at the end of this section.

5.3.1. Creating Provenance Data using the Fluxlog Library

The *fluxlog* library is used to extend applications with the ability to emit messages to a provenance store. Although *fluxlog* is shipped as part of the 13CFLUX2 toolbox[6], the library can be used with any C++ program to create provenance data. The following features are offered by *fluxlog*:

- *Immediate output*: provenance messages are created and published immediately. Thus, the execution of workflow steps can not only be observed post-mortem but online as well.
- *Various log types*: support for level of detail for different logging types, i.e., error, warning, notice, information and five additional types of debug messages.
- *Multiple streams*: emit provenance messages to multiple output streams (e.g., a file and a networking interface).
- *Performance optimized*: The *fluxlog* library is developed with special focus on memory and runtime efficiency. Depending on the application's parameterization, logging

[6] The API design and implementation of *fluxlog* is jointly developed with the 13CFLUX2 original author Michael Weitzel.

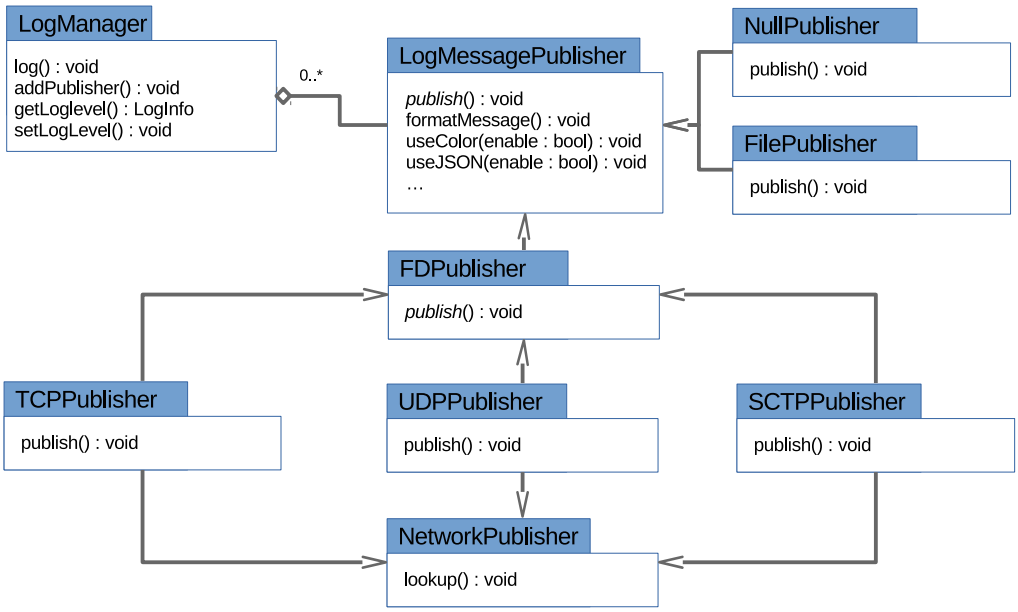


Figure 5.5.: Simplified UML diagram of the *fluxlog* class library. Publisher classes are registered in the `LogManager` singleton instance. By calling the `log` method, provenance messages are committed to all registered log publishers which subsequently decide on the log level whether to emit the messages. Further attributes (e.g., message formatting or coloring) can be configured in the publisher base class `LogMessagePublisher`.

can be activated at runtime for certain log types. By employing C/C++ macros, logging types can be disabled at compile time and, thus, impose zero runtime overhead on performance-sensitive simulations.

The UML class diagram of *fluxlog* is shown in fig. 5.5. To make an application provenance aware using *fluxlog*, the following steps need to be realized:

1. *Select a logging publisher method*: the abstract `LogMessagePublisher` class provides interfaces for concrete provenance log publishers. Besides the `publish` function, which emits the provenance messages, various commonly used methods are provided, e.g., to define the output format (`formatMessage`). There are several concrete publisher classes available: `FilePublisher` emits messages in files or the standard console output. To disable the output entirely on runtime, the `NullPublisher` class is employed. Under Linux, the file descriptor publisher `FDPublisher` is used to send messages to files. Specifically, because networking interfaces are accessed via file descriptors on Linux, this class is employed as base class for the networking publishers (`TCPPublisher`, `UDPPublisher`, and `SCTPPublisher`).

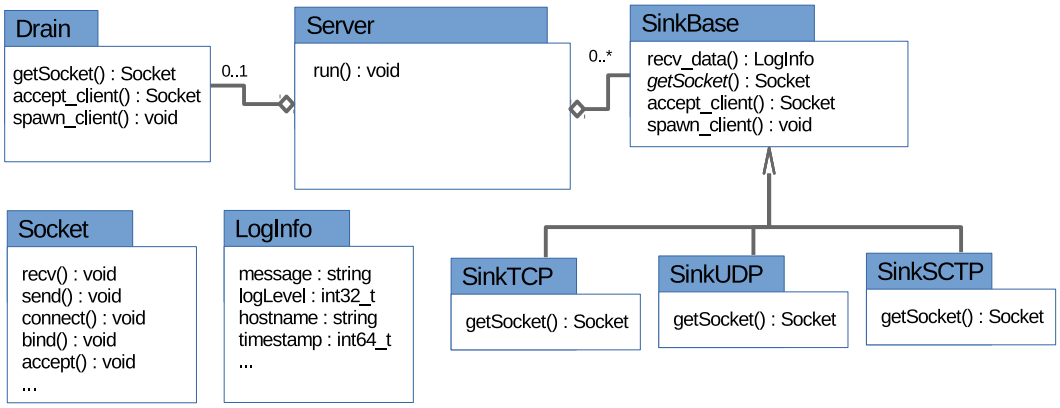


Figure 5.6.: Simplified UML diagram of the provenance store *fluxprov*. The `Server` class represents the provenance store. While multiple provenance sources (i.e., sinks from TCP, UDP or SCTP protocols) can be connected at a time, at most one query client is allowed to be active which is represented by the `Drain` class. Query clients are connected via TCP. The `Socket` class is a simple abstraction of the Unix socket interface (Stevens, Fenner, and Rudoff, 2003). The `LogInfo` class describes provenance messages and is imported from the *fluxlog* library.

2. *Configure the logging behavior*: the `LogManager` class is a singleton class that is used to configure an application for its logging behavior. In 13CFLUX2 programs, the parameter $-l$ controls the output of the provenance data. In *fluxlog* this is performed with the `SETLOGLEVEL` macro which accepts the most significant ordered enumerator type (i.e., ERROR, WARNING, NOTICE, INFO, and five DEBUG levels).

3. *Emit provenance information*: *fluxlog* offers two ways to emit logging messages: a macro is used to emit general messages (e.g., `fINFO` or `fTHROW`); secondly, by inheriting from a publisher, a custom stream for emitting specific data can be realized. Instead of using the aforementioned macros, messages are emitted directly using the class `publish` method.

5.3.2. Recording of Provenance Data

Beside the creation of log messages in an application, both, smart capturing and fast transport of provenance data to a provenance store are performance critical tasks. Capturing provenance messages from distributed 13CFLUX2 applications requires an efficient communication between a *fluxlog* program and the provenance store *fluxprov*. Hence, the

fluxprov program sits in-between programs using the *fluxlog* library and the provenance query client *provclient*. Figure 5.6 summarizes the UML class design of *fluxprov*.

The provenance store can be queried while at the same time log messages from 13CFLUX2 programs are captured. We call this monitoring feature *online capable*. To allow such an online monitoring of provenance data and to keep the design of the provenance framework simple, *fluxprov* provides two distinct networking sockets. Because the first socket collects the provenance data, it is termed *sink*. 13CFLUX2 provenance messages are erased from the store after being queried, therefore, the second socket is called *drain*.

The binary representation of a provenance message (of type `LogInfo`) is defined in the *fluxlog* library and shared with the provenance store. The `LogInfo` structure consists of the following metadata types: log type, process ID, hostname, file name, line and function name in the application source, and a record creation timestamp.

5.3.3. Querying the Provenance Store

provclient is a command-line query tool that sends requests for provenance messages to the *fluxprov drain* socket. A request can be parameterized with specific filters:

- *Hostname*: Designated simulation hosts can be selected by a network address or a hostname.
- *Message type*: Specific messages can be filtered by log type, e.g., only error and warnings can be requested.
- *Timestamp*: Messages captured within a time window are requested.
- *Message*: This option allows to filter messages by a regular expression.

An additional filter parameter, *discard*, controls the retention policy of queried messages. The user may keep queried messages in the database to perform multiple queries to the provenance store. Request filters are implemented as command-line parameters of *provclient*. POSIX regular expressions are used to implement this feature (Goyvaerts and Levithan, 2012). Finally, *provclient* allows flexible formatting of any selection of message and metadata fields as customized output string. This way, captured provenance messages can be easily converted to CSV or XML files.

5.3.4. Managing Provenance Services

To work with the provenance collection framework, the scientist needs to be able to perform these tasks:

- Start and stop the *fluxprov* program.
- Execute 13CFLUX2 programs that are parameterized to log into a provenance store.

- Query the provenance store contents.
- Submit queried provenance information persistently to the Subversion project repository.

Performing these tasks on a single machine is easily possible, e.g., by using the command-line. Because the framework transparently supports communication via TCP, UDP, or SCTP protocols, the use of the SWF provenance collection solution is likewise easy in a private networking environment. However, in a web environment (e.g., the Internet), the access to arbitrary networking ports is usually restricted by strict firewall rules. To overcome this issue, the provenance framework provides web service interfaces, i.e., *fluxprov* and *provclient* is started or stopped remotely using the *FluxCore* library.

FluxCore not only provides the ability to control 13CFLUX2 programs, but also a mechanism to pass common parameters within a workflow. Specifically, the provenance collection behavior is set-up in the flux parameter XML document. For instance, the following configuration of a *fluxprov* publisher sends the provenance data to the provenance store loghost.13cflux.net on port 9099 via UDP:

```
1 <parameter>
2   <common>
3     <logpublisher>
4       udp:loghost.13cflux.net:9099
5     </logpublisher>
6   </common>
7 </parameter>
```

5.3.5. Choosing the Network Transport Protocol

The provenance collection framework uses TCP as default transport protocol between the provenance-aware applications using *fluxlog* and the *fluxprov* service. To evaluate the influence of the network transport protocol on the overall performance of 13CFLUX2 simulations, a series of simulation experiments is conducted. Using UDP, TCP, and SCTP protocols, the provenance data emitted from a *multifwdsim* workflow is captured by a *fluxprov* service.

The experiments are conducted on two servers with 64 GB RAM and four Xeon X7350 processors, each with four CPU cores which are connected via Gigabit Ethernet (exclusive usage; with only a Layer-2 switch in-between). The first server creates 16 parallel simulation jobs with 1,000 samples each of a medium-scale *FluxML* model, while the second server runs *fluxprov*. The simulations are performed with the highest verbosity level producing 6,688,000 log data packets with a total size of 297,010,295 bytes.

As a reference, the simulation runs in 9:40 minutes with local file provenance data collection. Sending provenance data via TCP takes 10:05 minutes, i.e., compared to the local reference, a slowdown of approximately 5% is measured. The fastest network based transfer is achieved with UDP (9:43 minutes, less than 1% overhead). With a runtime of

9:49 minutes, SCTP is faster than TCP. The difference between UDP and SCTP, however, is surprisingly small. The number of packets not being captured by the *fluxprov* service was found to be less than 2% via SCTP. Table 5.2 summarizes the results.

	local file	**UDP**	**TCP**	**SCTP**
elapsed time (mm:ss)	9:40	9:43	10:05	9:49
packets (recv/send ratio)	100%	99.04%	100%	100%
slow-down ratio	–	0.51%	4.31%	1.55%

Table 5.2.: Comparison between local file and networking protocols. To obtain the reference execution time, the provenance messages are redirected into a file on the local system. The other measurements are performed in a local networking environment (two compute nodes with Gigabit Ethernet interfaces). While the second best performance is achieved with UDP (with only 3 seconds overhead), there is a measurable amount of lost packets. TCP is 25 seconds slower (4.3% slow-down) with no packet losses. Likewise, the simulation is performed reliably, but with only 9 seconds slow-down (1.5%).

From the tests it can be concluded that the provenance solution is fast. Although 283 megabytes of provenance data is generated in the experiment, the total simulation time increases by less than 1% in the best case. The measurements show that SCTP is superior to UDP and TCP in the given environment. Because the availability of SCTP is currently restricted to specially-confined computer networks (Tanenbaum, 2002), TCP is a default that works everywhere and has only little performance overhead on simulations with a high volume of provenance data.

5.4. Version-Controlled ^{13}C-MFA Workflows

In the ^{13}C-MFA domain, until now only loose guidelines for publishing results have been proposed (Crown and Antoniewicz, 2013). These guidelines concentrate exclusively on end-point reporting and lack information that enables the repetition of the computational workflow or the tracing of a model's evolution. Going beyond this, the SWF document storage concept is inspired by the MIASE guidelines (minimal information about a simulation experiment[7]) (Waltemath et al., 2011). Similar to MIAPE for proteomics experiments (C. F. Taylor et al., 2007) and the MIRIAM for annotating models (Novère et al., 2005), MIASE guidelines aim at defining the necessary information to perform reproducible simulation experiments.

5.4.1. Standard Project Template

For Subversion, a standard template has been developed to organize all information and data generated and used in a ^{13}C-MFA research project. Intermediate results, programs

[7]See `co.mbine.org/standards/miase/`

(as source code or binary), metadata (e.g., which and how programs are invoked), and further information is collected there. Thus, when starting a new ^{13}C-MFA project, a fresh repository is created that contains the following sub-folders:

- `raw_data`: raw measurement data files and intermediate results (e.g., .xls or .csv files).
- `results`: the final outcome of the study (e.g., .fml files).
- `protocols`: laboratory protocols that were used in the study.
- `provenance_data`: all log files, program execution parameters, and other technical metadata generated during the study.
- `tools`: custom executable scripts or programs utilized by the scientist.
- `notes`: the researcher's verbatim notes, comments, screenshots, and other media data that are captured during the study.

In this structure, the version-controlled files accumulate throughout the ^{13}C-MFA project. Superfluous data (e.g., logs that are only relevant for an ongoing simulation process) can be curated by the modeler, if appropriate.

5.4.2. Accessing SWF Projects via Web Service Interface

In the SWF, services access exchange files (input, output, and metadata) using the corresponding project repository. This way, all information is preserved while at the same time the exchange of data is standardized. The access to the project repositories is again realized as web service. *RepoManager* is a RESTful web service to manage and access SWF project repository data. The application employs Python and *PySVN* to provide the following services (cf. appendix B.6):

- */init/<repo>*: initialize a repository with name <repo>. Thereby, all sub-folders of the standard project template are created.
- */list*: return a list of all available repositories.
- */list/<repo>*: list the content of the repository <repo>.
- */add_project/<repo>/<name>*: add a new project with name <name> to the previously initialized repository with name <repo> (see /init above).

The above operations are invoked as a REST URL with a base URL, e.g., `http://localhost:8080/`. Using the open source cURL software[8], the REST service is easily accessed from the command line, e.g., `curl http://localhost:8080/list` executes the */list* operation.

[8] see: `https://curl.haxx.se/`

5.5. Revisiting the ^{13}C-MFA Workflow: Auxiliary Tools

Beside the standard tools 13CFLUX2 and Omix, a couple of features are identified for general usage (and further standardization) of ^{13}C-MFA workflows using the SWF:

- Visualization of input substrate mixtures (`drawMixingTriangle`).
- Measurement resampling in FluxML documents (`perturb` and `multiperturb`).
- Simulation tools that exploit multi-core compute nodes (`multifwdsim`, `multifitfluxes`).
- Collection and extraction of simulation data (`collectfitdata`).
- A conversion tool from FluxML HDF5 data to CSV (`hdf5tocsv`).

These tools are distributed as part of the 13CFLUX2 toolbox (Weitzel et al., 2013). Figure 5.1 highlights the software components that are (co-)developed as part of the SWF to complete the realization of semi-automated ^{13}C-MFA workflow applications.

5.5.1. drawMixingTriangle: Using Python instead of MATLAB

drawMixingTriangle.m is a MATLAB^{TM} program to visualize input substrate mixture triangles resulting from an in silico experimental design study and published as part of the 13CFLUX2 toolbox (Weitzel et al., 2013). MATLAB^{TM} is very popular in the scientific community, however, several issues need to be considered when integrating MATLAB^{TM} programs into a service-oriented environment such as the SWF (see also the official MATLAB^{TM} site; Edelhofer, 2014):

- MATLAB^{TM} is commercial and the standard licensing model is per user. Thus, to provide a web service that executes the MATLAB^{TM} interpreter, strict authentication of all users of such a web service must be provided. MathWorks provides a special *Distributed Computing Server* which is instead machine-bound.
- It is possible to integrate MATLAB^{TM} programs into other applications, by employing the C/C++ code generator[9,10].
- The integration of graphical MATLAB^{TM} programs with other software components can involve architectural and implementation effort (Lamprecht, Margaria, and Steffen, 2009).

On the example of *drawMixingTriangles*, it is demonstrated that the SWF approach to employ Python and NumPy instead of MATLAB^{TM} is a viable option. These steps are conducted for the new implementation:

[9] MATLAB^{TM} Coder: `https://mathworks.com/products/matlab-coder/`
[10] Coder supports only a subset of the MATLAB language (Edelhofer, 2014).

- Because *matplotlib* provides a MATLAB^{TM}-compatible syntax, a large amount of visualization could be adapted with only minor syntactical changes.
- The conversion of MATLAB^{TM} matrix operations to NumPy involves several pitfalls that need to be considered carefully. For example, the following operations are functionally equivalent:

```
# MATLAB
R = [0, -1, -min( T(:, 2)); 1, 0,-min(T(:, 1))]
# Python/NumPy
R = numpy.array( [[0., -1., -min(T[:,1])], [1., 0., -min(T[:,0])]]);
```

  Aside from being slightly more verbose, the NumPy version has some minor semantic changes, e.g., array indexes are starting with 0 instead of 1. A collection of all differences between MATLAB^{TM} and NumPy is found on the SciPy web site[11].
- *drawMixingTriangle* processes data in HDF5 format generated by the *edscanner* tool. To access HDF5 data out of a Python program, the H5Py library is used (Collette, 2013).
- The computation of quantiles is realized with `scipy.stats.mstats.mquantiles`.

In comparison, the new implementation offers several improvements: (a) the design is completely object-oriented, thus, its parts (i.e., the drawing and the computation part is separated from each other in the new implementation), are reusable; (b) the application is a stand-alone command-line program with the 13CFLUX2 I/O and parameter conventions; and (c) the implementation is functionally superior because different image formats and quality options are offered in addition.

5.5.2. perturb: Completion of the MCB Implementation

To implement the bootstrap method in 13CFLUX2, original measurements need to be replaced with randomized samples (Efron and Tibshirani, 1993). Algorithm 4 presents the `perturb` procedure in pseudo-code. The `generateNoise` function (Line 11), which is part of the 13CFLUX2 C++ library, overwrites the measurement group according to the measurement model with randomized values. Thereby, the normal distribution function with the parameters μ (measured sample value) and σ (the standard deviation obtained from the measurement device) is employed in the measurement model. An in-depth description of the 13CFLUX2 measurement model is found elsewhere (Weitzel, 2009)[chapter 4].

5.5.3. Multitools: Exploiting Simulation Performance on Local Nodes

Up until now, a researcher has three options to perform Monte Carlo simulations using 13CFLUX2 and the SWF: (i) small-scale simulations are easily performed with *fitfluxes*

[11] "NumPy for Matlab Users" porting guide: `http://wiki.scipy.org/NumPy_for_Matlab_Users`

Algorithm 4 PERTURB ALGORITHM

```
PERTURB(FMLIN, FMLOUT, GROUPTYPES)
 1  ▷ Read FluxML document from file
 2  FLUXML ← readFluxML(FMLIN)
 3  MGROUPS ← FLUXML.getMGroups()
 4  for each MG in MGROUPS
 5       do
 6          if MG ∈ GROUPTYPES
 7            then
 8                 ▷ Resample selected measurement group using the 13CFLUX2
 9                 ▷ library method generateNoise. It is ensured that the sum
10                 ▷ of all measurement values within MG equals 1.
11                 MG.generateNoise()
12  ▷ Write randomized data into a new FluxML file
13  FLUXML.writeFluxML(FMLOUT)
```

on a local node utilizing a single compute core; (ii) large-scale simulations are deployed on a Hadoop MapReduce cluster or cloud computing resource; and (iii) medium-scale simulations can be performed on multiple cores on a local compute node using the 13CFLUX2 program *multifit* (Weitzel et al., 2013).

Multifit is realized as Perl script that performs multi-start parameter estimations by invoking *ssampler*, *fitfluxes*, and *setfluxes* in a multi-tasked fashion. Due to this design, for M multi-start samples the FluxML input needs to be parsed and analyzed once for *ssampler*, and M times for *fitfluxes* and *setfluxes*, respectively (i.e., $1 + 2 \times M$ times in total).

Multifitfluxes addresses these issues by using the 13CFLUX2 C++ library (and, similarly, multi-core variants for *fwdsim* and *perturb* are realized as well). Figure 5.7 depicts the class diagram of *multifwdsim*, *multifitfluxes*, and *multiperturb*. The realization of these tools is summarized as follows:

- The classes `MultiFitFluxes`, `MultiForwardSimulation`, and `MultiPerturb` implement the functionality of the three tools.
- Inspired by the MapReduce design pattern, the `JobManager` class provides an environment for processing the abstract methods `map` and `process` (which are implemented by the above classes) in parallel.
- The helper classes `FluxMLManager` and `CollectDataStore` provide access to FluxML and HDF5 data files.

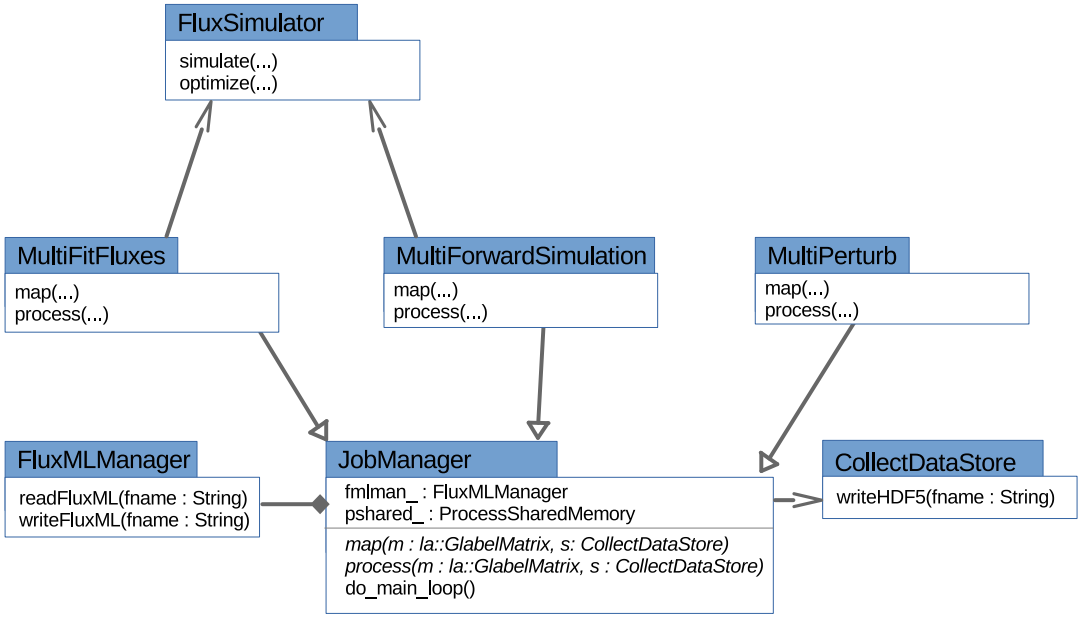


Figure 5.7.: UML class diagram of *multifwdsim*, *multifitfluxes*, and *multiperturb*.

- Because `FluxSimulator` utilizes `FluxMLManager` to process input FluxML files (instead of parsing each input separately), this class is employed by `MultiFitFluxes` and `MultiForwardSimulation`. The 13CFLUX2 library methods are used to perform the optimization and simulations, respectively.

These tools demonstrate the feasibility of custom-tailored ^{13}C-MFA applications in the SWF using the 13CFLUX2 libraries. The performance advantages of this approach (in terms of runtime and memory consumption) compared to the utilization of *fitfluxes* are demonstrated in § 6.6.

5.5.4. collectfitdata: Consolidate the Outcome of MCB Simulations

A full MCB simulation job often generates millions of FWDSIM files out of a single FluxML input model. For this vast amount of data, it is desirable to perform a pre-analysis where flux data from a smaller batch of MCB simulation files (e.g., with 30,000 files) is extracted and summarized and in a single HDF5 file. With this HDF5 data, fluxes are visualized (using Python and matplotlib) and subsequent workflow steps are decided by the researcher, e.g., update the model or perform a full-scale MCB simulation.

collectfitdata is a specially-tailored C++ helper tool to extract fluxes from multiple FWDSIM files and consolidate them into a single HDF5 file. Given a directory containing the FWDSIM files ($PATH$) and a selection of collection items (C) as input, the

collectfitdata procedure is described in algorithm 5. 13CFLUX2 library functions (e.g., `SimDocMultiReader`) are used to implement *collectfitdata*. To reduce the computational effort depending on N, the FWDSIM validation is performed only once in line 5.

Algorithm 5 COLLECTFITDATA Algorithm

COLLECTFITDATA($PATH$, C)

```
 1  ▷ Acquire and sort list of FluxML documents
 2  FILES ← readDirectory(PATH)
 3  ▷ Create reader (assuming non-empty list of regular files)
 4  ▷ Only the first file is fully parsed and validated
 5  S ← SimDocMultiReader(FILES[0], C)
 6  ▷ Create HDF5 file and create data fields
 7  H5 ← initializeHDF5Output(C)
 8  for each F in FILES
 9       do
10           ENTRY ← S.extractFWDSIMData(F)
11           H5 .appendData(ENTRY)
12  H5 .Close()
```

5.5.5. hdf5tocsv: A High-Performance Data Conversion Tool

Although HDF5 is an highly efficient format for storing and organizing scientific bulk data, it is sometimes necessary to stick to a simpler text format such as CSV. For instance, the CSV-based outcome of Hadoop MapReduce tasks can be merged using a simple file concatenation operation. Furthermore, spreadsheet processing tools like Microsoft Excel 2013 support textual input in the CSV format only.

The *hdf5tocsv* tool is developed as part of the SWF to convert 13CFLUX2-compatible HDF5 files, such as the output from *ssampler*, *multifitfluxes*, or *collectfitdata*, to CSV text files. By using the 13CFLUX2 and HDF5 libraries, this program requires 80 lines of C++ code (excluding argument parsing code and help texts; cf. appendix B.2). *hdf5tocsv* supports various formatting options, e.g., omitting the CSV header (which eases the concatenation operation), and changing the default delimiter from ',' to another character.

Chapter 6.

Use Cases

This chapter illustrates the utilization of the SWF on concrete use cases. As a guideline for this chapter, selected examples are taken from the complete ^{13}C-MFA workflow as presented in fig. 1.2. § 6.1 describes parallel data processing of raw MS measurement data. By analyzing the steps a scientist performs during the creation of ^{13}C-MFA models, the principles of hiding technical details in the SWF are covered (§ 6.2). A large-scale simulation involving 50,000 samples of two variants of a BCG vaccine model is presented in § 6.3. § 6.4 demonstrates the non-deterministic nature of typical ^{13}C-MFA applications on an exploration workflow that essentially narrows down the *flux solution space* of a model (Schuster, Dandekar, and Fell, 1999). The runtime performance of the Hadoop MapReduce MCB algorithm on purely local, cloud computing resources, and mixed environments is explored in §§ 6.5 and 6.6. Using the provenance collection framework, § 6.7 covers the online tracking of residual data, which is created in the course of a ^{13}C-MFA simulation workflow. Finally, the chapter closes with a workflow that combines multiple phases of modeling, simulation, analysis, and visualization with the interaction of a researcher (§ 6.8).

6.1. Mass Spectrometer Data Analysis with Hadoop DTW in R

To obtain usable data for ^{13}C-MFA analyses from raw MS measurements typically a series of processing steps need to be performed (Miebach, 2012). *Hyphenated methods* (e.g., GC-MS or LC-MS), which are typically used in ^{13}C-MFA experiments, yield chromatograms that are composed of large sets of consecutive mass spectra (Gross, 2011). Each chromatogram dataset has three dimensions: retention time, intensity, and m/z value. However, because the obtained mass spectra contain experimental errors, one step of the MS data processing workflow is to perform an alignment correction of the mass spectra retention time shifts in the chromatogram. `PA_alignAndChooseSpectra.R` is a script implemented by Max von Haugwitz[1] in the R programming language that performs the alignment correction using the *Dynamic Time Warp* (DTW) method (Giorgino, 2009).

On the example of this script, the integration of legacy tools in the scientific workflow and the parallelization of otherwise serial processing tasks is demonstrated in this use case.

[1] This script is provided by Max von Haugwitz as part of his doctoral thesis (Haugwitz, 2016).

Methods

The input data is provided by Stephan Miebach as part of his doctoral thesis (Miebach, 2012). A series of *C. glutamicum* wild type measurements consisting of a pre-processed list of 101 measurements are stored in CSV files with a total of 7,171 alignment entries. The analytical data is graphically pre-processed using the *JuMeDaS* tool. *JuMeDaS* is connected to the SWF by accessing the measurement database service using SOAP web service interfaces. The R script processes the input chromatogram and generates aligned data. Because the mass spectra are pair-wise independently, parallel processing using Hadoop MapReduce is possible.

DTWHadoop, the parallel version of the alignment script, is implemented using the Java Hadoop MapReduce API and the *FluxCore* library (cf. §§ 5.1.1 and 5.2.3). The Hadoop *map* function simply executes the script with the following command:

```
Rscript PA_alignAndChooseSpectra.R key thres filterwidth
```

Here, `Rscript` is the R interpreter that executes the `PA_alignAndChooseSpectra.R` script with its parameters. `key` is a unique text identifier, `thres` a correlation threshold, and `filterwidth` the smoothing width of a Gaussian filter. Because the outputs from *map* are adequate for further processing (CSV formatted results), the *reduce* function is realized as an idempotent function, i.e., the data is passed through as-is.

Results

The computation of all data sets takes 28 minutes on a single core. Employing two nodes with a total of 32 cores, the simulation time is reduced to less than three minutes. The R script is reused as-is, and only 150 lines of Java code is needed to implement *DTWHadoop*.

Discussion

This use case highlights two aspects. Firstly, the software components of the parallel MCB use cases are mostly reused. Although originally designed to integrate 13CFLUX2 tools into the SWF, the *FluxCore* library is reused for a (non-13CFLUX2) R script implementing the DTW method. Secondly, The realization with the SWF saves 90% of the total computation time. However, the relatively low parallel efficiency of about 0.30 indicates that the taken approach has a measurable performance overhead.

6.2. Metabolic Reaction Network Modeling Workflow

The core of any ^{13}C-MFA study is the *metabolic network modeling* workflow (cf. fig. 6.1). It consists of sub-tasks involving model set-up and acquisition of measurement data sets within a graphical environment, the evaluation of the model equations, and the visualization of data and simulation results. Human intervention and the integration of various knowledge and data sources is crucial in each step of the modeling workflow (Dalman, Droste, et al., 2010).

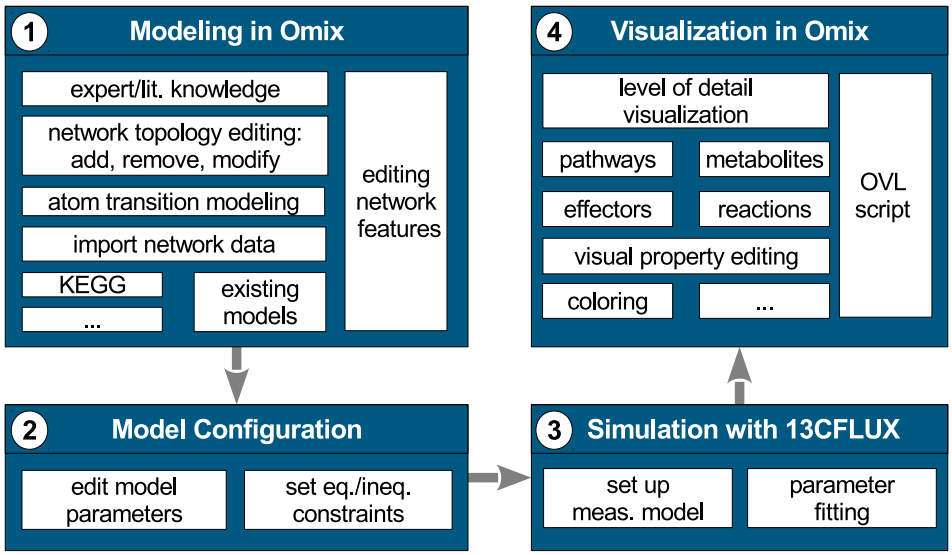


Figure 6.1.: A simple ^{13}C-MFA workflow. This workflow consists of the four steps model editing, configuration, parameter estimation with measurement data and visualization of parameter estimation results.

The realization of such a ^{13}C-MFA study involves several software tools and bioscientists often have to carefully handle subtle technical (i.e., IT-related) problems. For instance, 13CFLUX2 is a collection of command-line programs that exclusively runs on Linux operating systems. However, researchers are often working with a Microsoft Windows software environment where the explorer-like tool WinSCP[2] is used to exchange files with a remote Linux compute node and the PuTTY terminal emulator[3] to open a Secure Shell (SSH) command-line session to such a Linux computer. During the modeling process with Omix in such a Microsoft Windows environment, intermediate models frequently need to be run with *fitfluxes* or other 13CFLUX2 programs on a remote Linux node. From the user's perspective, one such run consists of the following six steps:

1. Save and export the model to the local filesystem during the editing process in Omix.
2. Copy the model file on a Linux workstation with WinSCP.
3. Open command-line console to the Linux workstation by starting a SSH session using PuTTY.
4. Invoke *fitfluxes* and store the resulting FWDSIM file locally.

[2]WinSCP web site: `https://winscp.net/`.
[3]PuTTY web site: `http://www.putty.org/`.

5. Transfer the FWDSIM results back (again using WinSCP).

6. Load the FWDSIM file into Omix to visualize the results.

In the following, a metabolic network modeling and visualization workflow is presented which frees the scientist from technical tasks by using the web services of the SWF.

Methods

The modeling and visualization workflow is realized by interconnecting Omix with 13CFLUX2 (Nöh, Droste, and Wiechert, 2015). By using the same file formats (e.g., FluxML, FWDSIM, or HDF5), the employed tools are made compatible to each other. The SWF provides access to the 13CFLUX2 toolbox with the *FluxWS* component. Hence, by realizing an Omix plugin interface, 13CFLUX2 programs are accessed via SOAP web services (Dalman, Droste, et al., 2010). Omix offers user-driven dialogs that improve the usability of ^{13}C-MFA workflows even for inexperienced users (Nöh, Droste, and Wiechert, 2015). For instance, the network model being edited in Omix is sent to the 13CFLUX2 *fitfluxes* service, which returns the simulated flux values. These values are immediately shown in the model view of Omix, e.g., the arrow thickness represents the computed flux values by utilizing the OVL capabilities (Droste et al., 2010).

Results and Discussion

The SWF serves for organization of complex analysis processes involved in ^{13}C-MFA applications. By encapsulating technical details and avoiding recurrent issues, sources for errors are minimized, the evaluation procedure for ^{13}C labeling experiments is accelerated and, moreover, becomes documentable. Being able to access the services provided by the SWF out of Omix allows the researcher to perform many ^{13}C-MFA steps out of a GUI application like Omix. The presented solution liberates the researcher from performing six purely technical and error-prone steps.

6.3. Simulating and Comparing BCG Vaccine Models

To demonstrate the SWF on a large-scale example, a combined simulation study of two models of the H37Rv vaccine strain of *M. tuberculosis* (BCG) organism is conducted. The models are structurally identical, however, they differ in their growth rate as discussed elsewhere in detail in the literature (Beste, Bonde, et al., 2011). The two models are henceforth called *BCG_fast* and *BCG_slow*. While this use case focuses on the general feasibility of a large scale ^{13}C-MFA simulations in the cloud, § 6.5 presents several 13CFLUX2 in Monte Carlo Bootstrap simulation studies in HPC and cloud environments in detail.

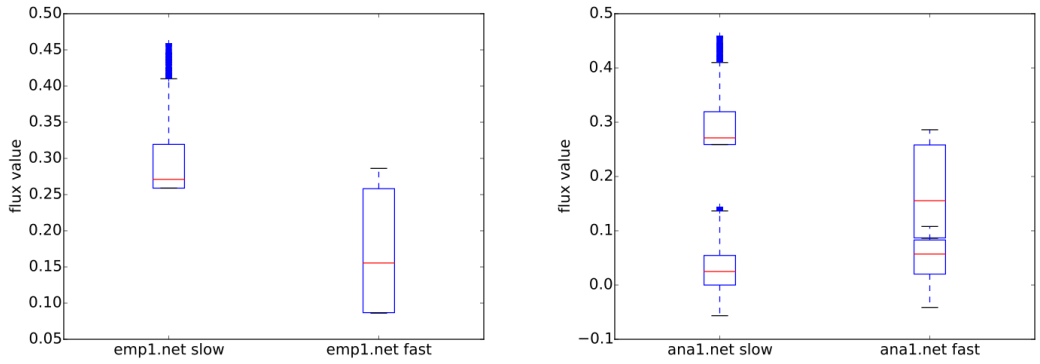


Figure 6.2.: Boxplot comparing distributions of *emp1 net* (left) and *ana1 net* (right) fluxes from the *M. tuberculosis* vaccine models *BCG_slow* and *BCG_fast*. The complete ^{13}C-MFA study results of the *M. tuberculosis* H37Rv organism is found in the literature (Beste, Nöh, et al., 2013). The script for this drawing is displayed in appendix B.7.

Methods

The models consist of 41 pools and 74 reactions with a mean forward simulation time of 0.34 seconds. However, performing parameter estimation requires a high number of optimizer iterations, i.e., more than 1,200 iterations with a runtime of about 9 minutes per flux vector on the IBG-1 server. For each model, $N = 1,000$ artificial measurements with $M = 25$ initial flux distributions are used resulting in 50,000 optimization runs in total. *FluxHadoop* is employed to perform the simulations on 20 Amazon EMR cloud *High-CPU extra large* instances with 152 CPU available for simulations.

Results

The total experiment runtime takes 24 hours and 23 minutes. The sequence file generated from 50,000 packed FluxML files takes 418 MB hard disk space and the resulting CSV files have a total size of 52 MB. The total cost of the computation using the Amazon EMR services is $384. Differences of specific flux values between the two BCG variants can now be identified using the SWF services. For example, fig. 6.2 depicts boxplots of the *emp1 net* and *ana1 net* flux distributions which are created using Python and matplotlib (cf. appendix B.7).

Discussion

To demonstrate the usefulness of the SWF, a large-scale Monte Carlo bootstrap simulation on 20 instances in the Amazon cloud is conducted on a biological meaningful model pair. Exemplarily, two flux distributions are extracted and visualized from the resulting

CSV files with a Python script. Because all steps – the invocation of the Hadoop cloud deployment, the download from the Amazon S3, and the generation of a boxplot image per flux value – are available as command-line calls, it is possible to automate the whole workflow chain with a Python script (this issue is covered in § 6.4).

There are several areas of future work. By extending the above workflow, visual tools (i.e., Omix) will be able to show boxplots and other graphics by interactively selecting a flux of interest. Two technical prerequisites (i.e., augmenting an application by a web service interface and organizing simulation results) are presented in subsequent use cases (cf. §§ 6.7 and 6.8). Having implemented the MCB algorithm for a scalable on-demand cloud solution employing 20 virtual instances, the next step is to perform studies with even larger models. For instance, the availability of metabolic network models with several hundred dimensions hints at future trends towards even larger problems that should be approached by *MapReduce* in a cloud.

6.4. Data-intensive Exploration of Flux Solution Spaces

The very first step for a modeler after assembling and proofreading the ^{13}C-MFA network is to get an overview of possible flux solutions that are compliant with the underlying network structure. While the stoichiometry and the imposed flux constraints define the *flux solution space* (Schuster, Dandekar, and Fell, 1999), labeling measurements effectively narrow this theoretical solution space. However, at this point, it is not clear whether there is only one or there are various flux that explain the measured data well. Obviously, this information is crucial for further analysis. One way of generating a rough impression of the effective flux solution space and its geometric characteristics (e.g., size, eccentricity) is to apply an explorative-oriented flux sampling workflow.

The proposed workflow consists of the following steps:

1. Sample n randomized flux vectors (samples) in the flux solution space using a Gibbs sampler as described in Weitzel et al., 2013. It should be noted that due to dimensionality, the volume of the flux space increases fast and flux samples rapidly get sparse (Bellman, 1957). Clearly, the larger the number of samples n chosen, the better the exploration coverage of the flux space.

2. For each flux sample, the emerging in silico labeling patterns are simulated (*forward simulation*).

3. The discrepancy between the in silico and the actual measurement data is calculated, which gives rise to the residual sums of squares value. The smaller the residual value, the better the model fit to the real measurements.

4. The fluxes and residual values are extracted from the simulation outcome. The results are sorted by residual value and the samples with the m smallest residuals are kept (m is a user-specified number with $m \ll n$).

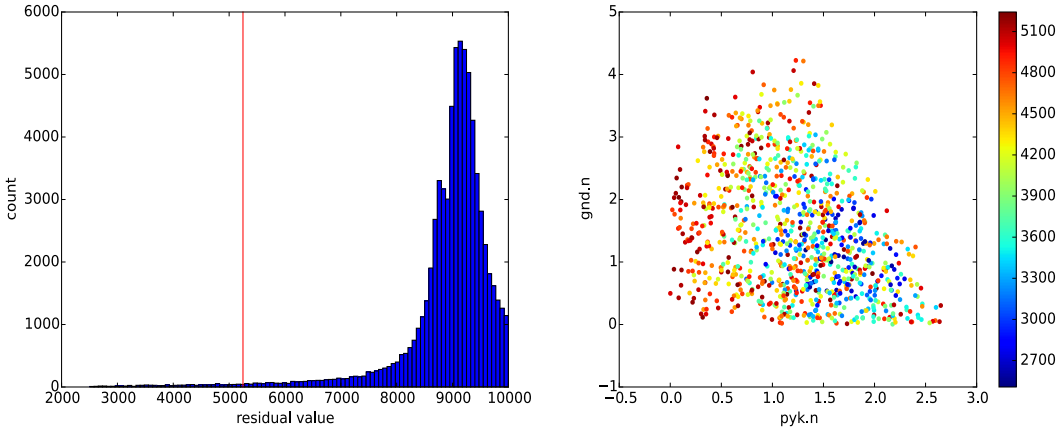


Figure 6.3.: Explorative ^{13}C-MFA simulation data with the *C. glutamicum* network model. Left: histogram showing the distribution of residuals generated by random sampling simulations. Exemplarily, the $m = 1,000$ best performing flux samples (to the left of the red line) out of a total of $n = 100,000$ are selected for further exploration. These samples may be suitable seeds for flux fitting later in the course of the analysis. Right: Scatter plot of the selected 1,000 flux samples to visually investigate the residual landscape. Color ranges from blue (low residual values) to red (high residual values). Note, that due to the high dimensionality of the flux space, trends rather than clear-cut separations by residual value can be expected in low dimensional projections. Flux names: gnd – glucose-6-phosphate dehydrogenase, pyk – pyruvate kinase. Values are given relative to the substrate uptake rate.

5. The m flux samples with the smallest residuals are visualized to identify alternative flux solutions. Selected sampling results are shown in fig. 6.3.

With the dimensionality of the problem (23 in our case, but possibly several hundred for large scale networks (Ravikirthi, Suthers, and Maranas, 2011)), this type of explorative analysis easily becomes data-intensive. Therefore, two solutions are proposed: a single-node implementation that is adequate for small-to-mid-sized models and a Hadoop MapReduce variant for large-scale models with a high-dimensional flux space. Starting with the single node solution, specific code changes are highlighted that are necessary for deployment in the cloud.

Methods

The single-node implementation of workflow consists of two parts (cf. fig. 6.4):

(a) *Sampling and simulation*: in essence, a Python program executes the 13CFLUX2 tools *ssampler* and *multifwdsim* to generate random samples and forward simulation

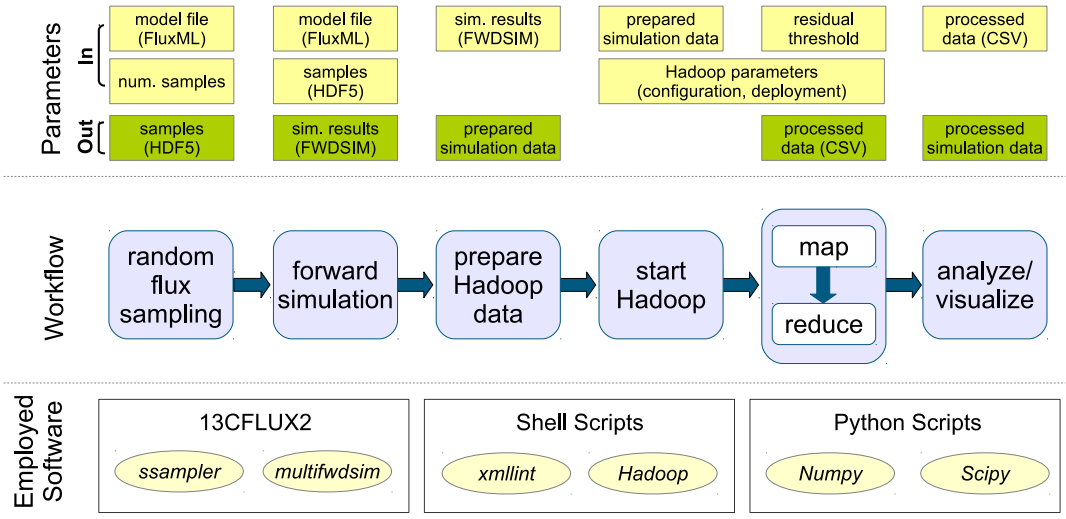


Figure 6.4.: Overview of the sampling-oriented flux space exploration workflow. The individual workflow steps are presented in the middle row, while the employed tools are displayed in the bottom row. The input and output parameterization of these tools is shown on the top layer.

results, respectively (`sample_and_filter_multi`, cf. appendix B.8). By reading the simulation results using the Python modules numpy and h5py, the workflow yields a CSV file as output containing the m best flux solutions along with their residual values.

(b) *Scripted visualization*: using the Python modules numpy, matplotlib, csv, and gzip, two visualization programs are realized to analyze the data output (`plot_histogram`, `plot_scatter`; cf. appendices B.9 and B.10).

The Hadoop MapReduce variant of the workflow is similar with respect to the inputs and outputs of the workflow. Hence, the realization is a drop-in replacement for the sampling and simulation step (a), while the scripted visualization step (b) can be reused. Nevertheless, there are technical differences between the single-node and the Hadoop variant of the simulation workflow (the employed scripts are displayed in appendix B.11):

- Hadoop deployment is prepared with a shell script (`prepare_hadoop_uc1`). Specifically, the FluxML file (*value*) is prepended with a serial number (*key*), transformed to fit in a single line, and passed to the Hadoop Streaming API. The xmllint tool, which is part of the open source Libxml library[4] is used to canonicalize the FluxML content. Here, canonicalize means that the simulation results are combined in a

[4]Libxml version 2.9.2 is employed; project web site: `www.xmlsoft.org/`.

Hadoop-compatible text file with one string line per flux sample and tab-delimited key-value pairs per line.

- Sampling and simulation is performed with the Hadoop script `map_uc1`. Because the input of a map processor is always a pair of identifier (*key*) and FluxML content (*value*), the latter needs to be extracted before the 13CFLUX2 program sequence is invoked. Here again, the code of the single-node implementation is largely reusable.

- Instead of extracting the m best random samples, the Hadoop script `reduce_uc1` script filters the samples by residual value. To this end, a provisional threshold is determined by employing the aforementioned single-node variant of this workflow with a small batch size (e.g., with $n = 10,000$ and $m = 1,000$). Samples exceeding a certain threshold are discarded. The reason for this change is performance: while filtering n samples by threshold is an $O(n)$ operation, extracting the best m results requires a sorting step of all outputs which involves $O(n\,log(n))$ operations (Cormen et al., 2009). Thus, the complexity is significant for the runtime, especially when millions of samples are processed.

Results and Discussion

Using the tools and services provided by the SWF allows a realization of the computational part of the exploration algorithm in a single script that consists of essentially 37 lines of comprehensible Python code. Notably, the conversion of a single-node to a distributed computing variant on a Hadoop MapReduce HPC resource (e.g., Amazon's cloud) needs only a few changes: an additional preparation script for the deployment and map reduce scripts with 35 lines of code in total are added to transform the input/output data to the necessary form. As demonstrated by the semantic change from extracting by threshold instead of the best m flux vectors, tailored algorithmic adaptations are needed to effectively translate computational problems to larger scale. Apart from this, the use case demonstrates the benefits of deploying data-intensive simulation jobs on cloud computing resources.

6.5. Cloud Monte Carlo Bootstrap ^{13}C-MFA Workflows

In the ^{13}C-MFA context, MCB is an exemplary application for computationally demanding simulation job. By parallelizing the MCB, a significant amount of time can be saved. This section investigates two computational MCB workflows that are deployed on virtual nodes in the cloud. Firstly, a series of MCB simulations with three models are conducted on four different VM setups using Amazon's EMR cloud offering. Thereby, the principle feasibility of the taken approach by deploying 13CFLUX2 on cloud computing resources is studied (cf. Dalman et al., 2010). Although the achieved time savings are impressive, an in-depth analysis of the results reveals that there is potential for further optimization. Hence, the second use case focuses on improving the MCB algorithm (Dalman et al.,

2013). Compared with the first approach, an improvement of the performance by 64% could be achieved with a large-scale model and virtual cloud setup. In absolute numbers, the total simulation time improved from about 23 hours on a single compute core down to less than half an hour on a cloud setup with eight VM nodes with eight cores each.

6.5.1. Common Methods

In the following, the similarities of the cloud experiments are presented, i.e., employed network models, the parameterization of the MCB algorithm, and the Amazon EMR setup. To gain a better understanding of the runtime behavior of the models, a series of single core (i.e., non-parallel) measurements are conducted on a local cluster node first. Likewise, T_1 measurements are performed on Amazon virtual machines that are used to compare the performance with the distributed Hadoop cloud implementation of the MCB algorithm.

6.5.1.1. Network Models

Three metabolic network models of different size have been selected (cf. § 3.2.4):

1. *Model 1*: a small toy network consisting of 6 pools, 8 reactions (Antoniewicz, Kelleher, and Stephanopoulos, 2006).
2. *Model 2*: an essential central metabolism of *C. glutamicum* with 34 pools and 65 reactions (Petersen, 2001).
3. *Model 3*: an extended *C. glutamicum* network with 76 pools and 117 reactions (Petersen, 2001).

6.5.1.2. MCB Parameters

The choice of the number of independent optimization runs from M random initial flux distributions depends on the degree of nonlinearity and conditionedness of the parameter fitting problem. Thus, no general guideline can be given for the number of parameters and the size of the search space. As suggested in the literature, $M = 10$ random initial flux vectors and $N = 1,000$ pseudo-measurement data sets are chosen as parameters for the MCB algorithm (Chernick, 2007).

6.5.1.3. Local Single Core Simulations

Before deploying the model simulations on the Amazon cloud, the MCB algorithm with the above setup is run on a single core on an IBG-1 server node. These measurements are similar to the benchmark performed in § 3.2.4 and have the purpose to provide a deeper understanding of the simulation times and the model behavior (cf. table 6.1). Note that, while using the same methods, the average runtimes per model differ from the simulation times in table 3.2. This difference is explained by the choice of a different optimizer.

Here, the open source non-linear optimizer software IPOPT[5] is selected. Due to licensing reasons, the commercial NAG-C[6] optimizer is restricted to simulation experiments on IBG-1 compute nodes.

In summary, several factors are identified that constitute the non-deterministic nature of ^{13}C-MFA parameter estimation runtimes:

- The model size (defined by the number of pools and reactions) influences the simulation runtime.
- The number and quality of measurements have an impact on the robustness of the optimization procedure, thus, directly influencing the runtime of the parameter estimation step.
- The parameterization and choice of the optimization routine influences the runtime behavior.

	Model 1	Model 2	Model 3
T_1 total time (h)	0:30:42	1:51:57	19:40:40
total file size in (MB)	82	277	1,371
total file size out (MB)	79	470	2,081
optimization mean (s)	0.02	0.13	1.79
optimization min (s)	0.0016	0.1280	1.47
optimization max (s)	2.4711	0.7994	5.57
optimization total (h)	0:03:16	0:22:17	4:57:47

Table 6.1.: Computation times on local machine (single core usage). Raw total input (*FluxML*) and output (*FWDSIM*) data set sizes also reveal the differences between the three models: a toy network (*Model 1*), a basic *C. glutamicum* network (*Model 2*), and an extended *C. glutamicum* network (*Model 3*). The last four rows show pure optimizer runtimes, i.e. without data processing, network setup, or stoichiometric and topological analysis.

6.5.1.4. T_1 Reference Measurements on Amazon Cloud Virtual Machines

To compute the *speedup* and *parallel efficiency* (PE) of a distributed algorithm, the single core measurements (T_1) have to be obtained. Table 6.2 shows the simulation time on a single core virtual machine (T_1). The reference simulation times are obtained from single core measurements of Amazon EMR machine types from the instance class *High-CPU*

[5] https://projects.coin-or.org/Ipopt
[6] http://www.nag.com/numeric/CL/CLdescription.asp

	Model 1	Model 2	Model 3
simulation time (h)	0:42:03	2:01:09	22:44:13
collectfitdata (h)	0:00:17	0:01:04	00:13:51
fwdsim2csv (h)	0:07:18	0:07:07	00:09:22
total time T_1(collectfitdata) (h)	0:42:20	2:02:13	22:58:04
total time T_1(fwdsim2csv) (h)	0:49:21	2:08:16	22:53:35

Table 6.2.: Single core benchmark results on a local node. While Table 6.1 presents single core measurements on the IBG-1 server, runtimes on *High-CPU extra large* Amazon instances are shown here (10,000 flux vectors). Smaller network models are significantly faster using *collectfitdata* (T_1(collectfitdata)) than processing the simulation results with *fwdsim2csv* (T_1(fwdsim2csv)). This effect is reversed with the extended *C. glutamicum* model.

extra large. VMs from the instance classes *High-CPU extra large* and *Standard extra large* are compared in § 6.5.2 with respect to their runtimes in a distributed Amazon EMR cloud setup. Because 13CFLUX2 has a comparably low memory footprint, the *High-Memory* instance class is omitted in the comparison studies. The T_1 simulation time varies between 42 minutes and almost 23 hours on an Amazon instance, depending on the model.

6.5.2. Hadoop MapReduce Streaming Approach to the MCB Workflow

The purpose of this section is to investigate the general feasibility of deploying 13CFLUX2 and the MCB algorithm on Amazon's EMR cloud computing service. Thereby, the machine types *High-CPU extra large* and *Standard extra large* are compared to each other with respect to runtime performance and price per simulation.

Methods

The straightforward MCB realization using the Hadoop streaming API is employed for the experiments. In the streaming experiments, S3 is used as the distributed file storage, and the granularity parameter has been set to $G = 10$.

Results

Table 6.3 depicts the measurement results for the test networks. Compared to the overall simulation time, the overhead to provide the VM cluster running Hadoop is low: depending on the machine type and the number of requested resources, it varies between 2:40 minutes and 8:30 minutes. Among various factors, this effect predominantly depends on the virtual cluster load and the locality of the virtual resources. Likewise, runtime performance fluctuations are measured on the cloud resources which are also observed in other studies (see, e.g., Zaharia et al., 2008; Jiang et al., 2010; Sehgal et al., 2011).

	Model 1	Model 2	Model 3	Model 1	Model 2	Model 3
	Standard 4 machines, 4 cores			*Standard* 8 machines, 4 cores		
total time (h)	0:09:46	0:19:15	2:34:35	0:08:49	0:15:22	1:54:58
comp. time (h)	0:06:44	0:16:08	2:31:17	0:05:08	0:11:41	1:50:39
bootstrap (h)	0:03:02	0:03:07	0:03:19	0:03:41	0:03:07	0:04:19
total price ($)	2.78	2.78	8.34	5.56	5.56	11.12
	High-CPU 4 machines, 8 cores			*High-CPU* 8 machines, 8 cores		
total time (h)	0:08:08	0:13:54	1:48:53	0:07:18	0:11:56	1:22:44
comp. time (h)	0:04:58	0:10,53	1:45:31	0:04:00	0:08:56	1:19:09
bootstrap (h)	0:03:10	0:03:01	0:03:22	0:03:19	0:02:59	0:03:34
total price ($)	2.78	2.78	5.56	5.56	5.56	11.12

Table 6.3.: Computation times on Amazon instance types *Standard extra large* (upper rows) and *High CPU extra large* (lower rows). Three network models of different size are selected for the simulations: a toy network (*Model 1*), a basic *C. glutamicum* model (*Model 2*), and an extended *C. glutamicum* model (*Model 3*). The MCB algorithm is executed with $N = 1,000$ and $M = 10$ (i.e., 10,000 flux vectors in total). Simulations on *High CPU extra large* take less runtime than on *Standard extra large* Amazon instances. At the same time, the cloud service costs are almost equal.

The simulation results reveal that the performance gain (decrease of runtime) is only *25%* when the core count is *doubled*. A *High-CPU extra large* instance with 32 cores (4 machines, 8 cores) is only 7.5% faster than a corresponding *Standard* instance (8 machines, 4 cores). Because a *High-CPU extra large* machine has 0.5 ECU more per core than a *Standard* machine (i.e., 2.5 instead of 2 ECU), a performance gain of 25% is expected. However, the measurements show that *High-CPU extra large* machines are clearly preferable since the cost per core is half – therefore, the computation takes less time and costs 50% less.

Discussion

These observations indicate that I/O throughput might be a bottleneck hampering the expected 25% gain. Amazon Simple Storage Service (S3) performs poorly when small files are read and written (Palankar et al., 2008). This problem occurs in these experiments, since during one simulation run thousands of kB-sized files are processed from and to S3. On that score, Amazon S3 is replaced by HDFS. Using the same setup as above (despite replacing the storage technology), further measurements are conducted. It turns out that Hadoop performs even *worse* than S3 in our scenario (about 12% higher total runtime).

The reason is, again, the relatively small size of the simulation files, while Hadoop is designed to process *big data*, i.e., total file volumes in the order of several hundreds of terabytes and more (Sammer, 2012).

In this application, the intermediate results, the FWDSIM data (which averages to about 79 kB per file), the full chunk size per I/O operation needs to be processed which is 64 MB by default (White, 2009). Hence, each read and write on the HDFS filesystem requires more than 800 times more I/O operations than required for the raw data. Furthermore, to improve the fault-tolerance in jobs with hundreds of compute nodes, every chunk written to HDFS is replicated on at least three nodes. In summary, this application suffers from the fact that I/O operations on small files are expensive using S3 and HDFS.

Nevertheless, even with sub-optimal I/O performance, the time savings are impressive: for example, 1:22:44 hours instead of 19:40:40 hours (18:17:56 hours less) are required to compute the extended *C. glutamicum* network. The total cost for using the Amazon service is $11.12 – clearly, one has to take into account that local resources also generate cost (acquisition costs, electricity, administration, etc.). Further tuning of the algorithm implementation should lead to even lower computation times. This is subject to the revised MCB workflow in the following.

6.5.3. Cloud Monte Carlo Bootstrap Workflow – Revised

The previous use case showed that the deployment of 13CFLUX2 and a real-world application on cloud computing resources is possible using Amazon's EMR cloud computing offering. Although the absolute time savings are indeed impressive, the parallel efficiency is at best only 0.408 on 32 cores and 0.272 on 64 cores. This revised use case discusses the measures taken to improve the parallel efficiency.

Methods

The revised MCB implementation, called *FluxHadoop*, addresses the shortcomings of the streaming version. Functionally, the *reduce* step uses *fwdsim2csv* instead of *collectfitdata*, thus, enabling the execution of parallel *reduce* steps. Consequently, other technical changes, such as using the Java interface, employing the *HadoopPacker* application, and fine-tuning Hadoop MapReduce parameters are as well implemented (cf. § 5.2.3).

Results

Table 6.5 summarizes the cloud runtime measurements. The extended *C. glutamicum* network has a lower simulation runtime with *FluxHadoop*, but the smaller networks require more time in comparison to the streaming version. With Model 1, the simulation time on 32 cores is 7:02 minutes (41% regression) and 5:05 minutes on 64 cores (27% regression). Model 2 takes 10:20 minutes on 32 cores (5% improvement) and 5:46 minutes on 64 cores (35% improvement). The best improvement is seen with Model 3. On 32 cores, the simulation only takes 53:09 minutes (50% improvement) and 28:40 minutes on 64 cores (64% improvement).

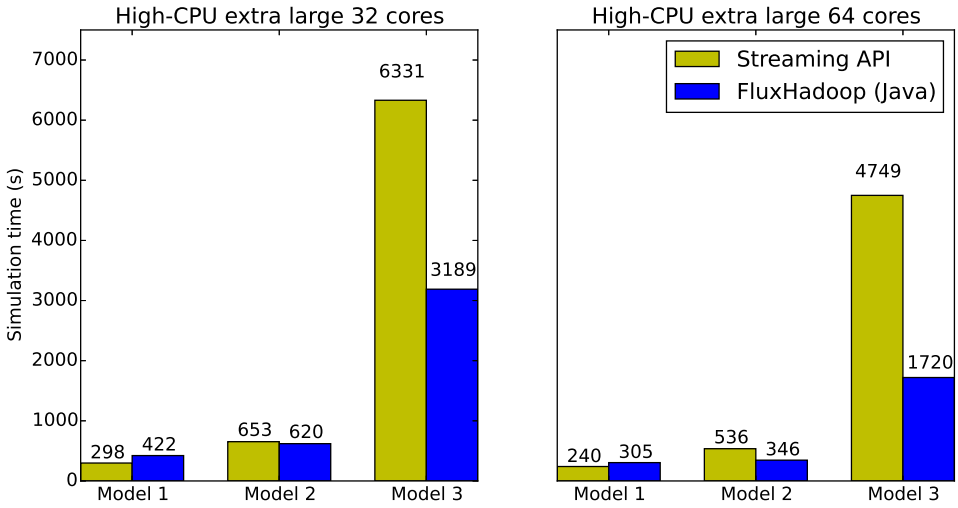


Figure 6.5.: Barchart of the benchmark results on Amazon instances. The figure depicts the simulation times on High-CPU extra large Amazon instances (left: four instances (32 cores total), right: eight instances (64 cores total). The streaming API version (yellow bars) performs faster with the smaller networks, however, the revised *FluxHadoop* approach (blue bars) scales well with larger network models. The parallel efficiencies of both approaches are easily computed using the T_1 measurements (using *collectfitdata*) from table 6.2 for 32 cores (Streaming API, FluxHadoop): Model 1 (0.266, 0.188), Model 2 (0.351, 0.370), Model 3 (0.408, 0.810). Similarly, the results with 64 cores: Model 1 (0.165, 0.130), Model 2 (0.214, 0.331), Model 3 (0.272, 0.751).

Discussion

One reason for the runtime difference of the investigated MCB cloud implementations is caused by the execution of *collectfitdata* and *fwdsim2csv*, respectively (cf. table 6.1). *collectfitdata* and *fwdsim2csv* vary in several aspects of their software design, i.e., implementation language, input parameters, and output. The simulation runtime performance of the MCB workflow has been measured for three examples of different complexity. Parallelizing the *reduce* step of the MCB algorithm leverages the pressure of data collection on a single node, and combined with tuning of Hadoop-specific parameters of *FluxHadoop* eliminates the I/O performance bottleneck and leads to acceptable scaling behavior.

With the revised implementation, significant performance improvements are achieved, in absolute and relative scales. While the new MCB implementation is a great enhancement, there is still room for improvement: the best parallel efficiency achieved is 0.810. One possible area for potential improvements is further tuning of Hadoop-specific pa-

rameters. Optimal Hadoop parameters, however, depend on influencing factors that are problem-specific like network model sizes or optimizer performance.

6.6. Hybrid Parallelization Approach for MFA Simulation Workflows

This study aims at improving the performance of parallel simulations by extending the MCB algorithm with a hybrid parallel implementation that combines Hadoop MapReduce with shared memory parallelization techniques (cf. Dalman, Weitzel, Freisleben, et al., 2011). Compared to the purely Hadoop-based *FluxHadoop* solution, the hybrid-parallel realization (*FluxHadoop2*) is significantly faster while it simultaneously benefits from reduced memory requirements.

This use case exploits the MCB algorithm by performing all M multi-start flux vectors locally, while still using the Hadoop MapReduce framework to distribute N cloud tasks (cf. § 5.2.4). It is shown that the algorithm and the Hadoop MapReduce Java program are basically unchanged, i.e., this use case especially focuses on the challenges of replacing the underlying 13CFLUX2 tool *fitfluxes* by *multifitfluxes*.

Methods

FluxHadoop is compared with *FluxHadoop2* on the two-node institute cluster with 16 cores each. The MCB simulation is performed with $N = 1,000$ and $M = 20$ on a small-scale network (20 dimensions, 2 model parameters). While *FluxHadoop* is executed with 16 parallel Hadoop tasks, only four `multifitfluxes` tasks are executed per node with *FluxHadoop2*. With $k = 20$ parallel parameter estimation threads per task, the server nodes' computing capacity is fully exploited.

On virtual Amazon *High-CPU extra large* nodes, *FluxHadoop* and *FluxHadoop2* are compared. The simulation runtime is computed on a node employing a single core (T_1), four (T_{32}), eight (T_{64}), and 16 nodes (T_{128}). An experimental series employing a large metabolic network model of *Penicillium chrysogenum* (328 dimensions, 18 model parameters) is conducted (Niedenführ, 2014)[7]. By calling *fitfluxes* and *fwdsim2csv* using a Linux shell script, the serial simulation runtime (T_1) of 20,000 Monte Carlo bootstrap flux vectors is computed on a virtual node. Three *multifitfluxes* map tasks per node are invoked with $k = 8$ parameter estimation processes per task.

More than 200 adjustable parameters are provided to tune and customize the Hadoop framework. The focus is on modifying few parameters, instead of optimizing the applications for specific Hadoop versions. For this experiment, the adjusted Hadoop parameters are summarized in table 6.4, defaults from Hadoop and Amazon are retained otherwise.

[7] A preliminary, unpublished version of the *P. chrysogenum* network model is used for the runtime measurements. Niedenführ, 2014 discusses even larger variants of this model.

Hadoop parameter	value
mapred.child.java.opts	-Xmx512m
mapred.map.tasks.speculative.execution	false
mapred.max.split.size	262144
mapred.map.tasks	200
mapred.reduce.tasks	40
mapred.task.timeout	86400000

Table 6.4.: Custom Hadoop parameter settings used for the runtime benchmarks with *FluxHadoop* and *FluxHadoop2*.

Results

With *FluxHadoop2*, the simulation with the small network consumes 3 minutes and 44 seconds and 1.3 GB memory in total. Compared to the non-hybrid *FluxHadoop* implementation the runtime is decreased by 33% (5:39 minutes) and 75% less total memory is required (5.2 GB). fig. 6.6 depicts the results.

The runtime results for the *P. chrysogenum* model in the cloud are summarized in fig. 6.7. On a single Amazon node the 20,000 vectors are simulated in 302 hours (T_1). While *FluxHadoop* performs faster than *FluxHadoop2* on four and eight nodes, *FluxHadoop2* performs 20% faster than *FluxHadoop* on 16 nodes, i.e., 6 hours 20 minutes instead of 7 hours 57 minutes. Notably, the parallel efficiency of the new hybrid parallel approach is almost constant for the conducted cloud experiments (varying between 0.37 and 0.39).

Discussion

This study clearly shows a scalability improvement over the original implementation of the MCB algorithm. The hybrid parallelization approach is faster than *FluxHadoop* – both, locally employing a small-scale model, and a realistic simulation setup employing 16 virtual Amazon cloud nodes. While *fitfluxes* has a low memory footprint (cf. table 3.2), the Java-based Hadoop MapReduce tasks require up to 325 MB per process[8]. Being a shared memory application, `multifitfluxes` utilizes multiple compute cores on a node, resulting in a massively reduced memory consumption behavior. The experiments show that the parallel efficiency of the hybrid parallel approach is constant, i.e., runtime is halved by doubling the number of employed compute cores. Future work will investigate the low runtime performance on four and eight Amazon nodes of *FluxHadoop2* compared to *FluxHadoop*. In conclusion, the hybrid parallel ^{13}C-MFA MCB service paves the way to perform serialized statistical evaluations with even larger-scale networks in series.

[8] The so-called *residential set size (RSS)* is measured, hence all shared memory pages per process are measured (Love, 2013).

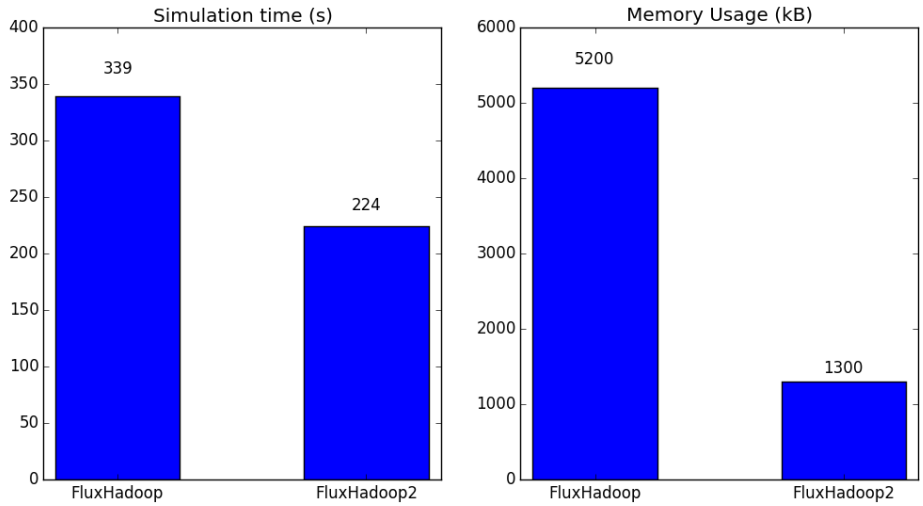


Figure 6.6.: Results of a microbenchmark comparing *FluxHadoop* and *FluxHadoop2*. A small test network model with 20,000 flux vectors in total is used to compare *FluxHadoop* and *FluxHadoop2* on two local cluster nodes. In this experiment, the simulation time decreased by 33% (left), while the total RSS memory usage decreased by 75% (right).

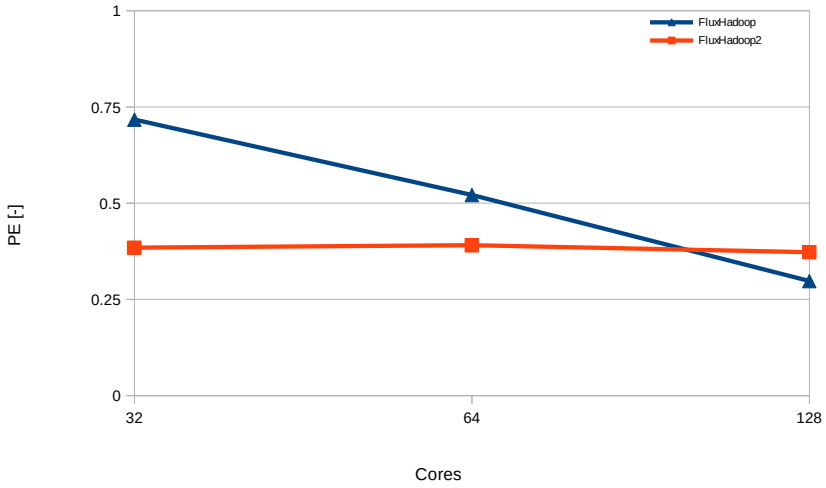


Figure 6.7.: Runtime comparison of *FluxHadoop* and *FluxHadoop2*. The parallel efficiency is defined as $PE(P) = T_1/PT_p$, where T_1 denotes the serial runtime and T_p the parallel runtime on P processing units.

6.7. Online Residual Tracking

During the modeling of a ^{13}C-MFA network, parameter estimation is performed interactively by a researcher to explore model variants and their performance (e.g., runtime behavior, goodness of fit, etc). With 13CFLUX2, parameter estimation is started with several deliberately chosen initial flux vectors spread over the whole feasible flux space. Using flux estimation, (hopefully) one solution is identified that explains the measured values best. Monitoring the residual value of the optimization process gives an immediate impression to the researcher especially compared to previous runs. For instance, it can be seen whether the optimization approaches a local sub-optimal flux space area. Therefore, observing ^{13}C-MFA simulations is an advisable tool for researchers.

The use case demonstrates the advantages of the provenance collection solution. Exemplarily, the execution of *fitfluxes* with a medium scale model is interrupted after reaching a predefined threshold. This experiment reveals simulation time savings of more than 80%.

Methods

The simulations are performed with the basic *C. glutamicum* model on two compute nodes of the local institute cluster (Petersen, 2001). Provenance data is captured on the second compute node with the *fluxprov* service. Using *provclient* on the scientist's workstation, the captured log messages are read from the *fluxprov* service. The residual value is extracted from the provenance message using the following command line: `provclient -M "residual" | cut -d ' ' -f 5`. By tracking the development of the residual value in the *provclient* console, the scientist interrupts the parameter estimation process at will.

Results

Figure 6.8 depicts the captured residual values from the *fitfluxes* run. Initially, the residual value of the network is 70,127. The global optimal residual value 1,748 (minimal distance) is achieved after 18 seconds. With time, the residual value does not further decrease considerably. In this scenario, an experienced scientist may choose to interrupt the *fitfluxes* run prematurely after 3 seconds to obtain a residual value of about 2,500. Thus, 15 seconds of time are saved, i.e., the total runtime is reduced by more than 80%.

Discussion

The use case demonstrates the provenance collection framework in action. It is a versatile approach tailored to the demands of high-performance ^{13}C-MFA simulation workflows and distinguishes itself from other provenance solutions:

1. This provenance solution supports online operation. Because continuously generated values from a ^{13}C-MFA simulation can be captured *on demand*, the provenance

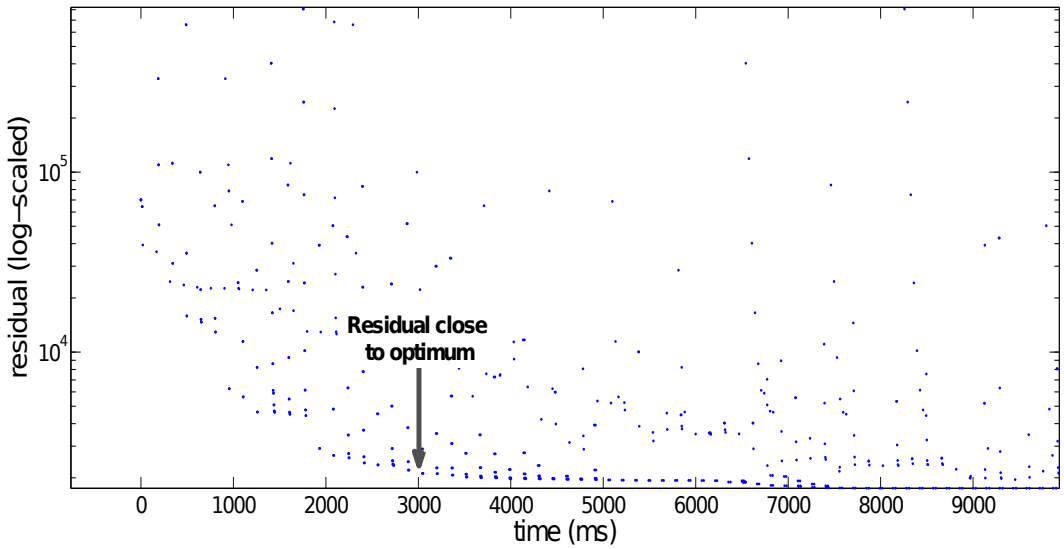


Figure 6.8.: Residual values collected from a *fitfluxes* run. The total calculation time is 18,000 ms, however after about 3,000 ms the residual no longer improve significantly. Thus, by monitoring online provenance data, the simulation could be interrupted prematurely and save a significant amount of the overall simulation time.

collection solution enables interaction with ^{13}C-MFA workflows that were out of reach before.

2. The provenance tools integrate well into the SWF and the employed toolchain. The provenance collection framework supports filtering messages by various metadata and regular expressions for the message content.

The provenance framework is designed to support the requirements of ^{13}C-MFA workflow applications. Because the presented provenance solution is by design non-intrusive and provides clean interfaces, future work includes the application in other scientific domains.

6.8. Interactive Hadoop-based ^{13}C-MFA Workflow

The exploration workflow described in § 6.4 provides the modeler information as to whether the model is – in principle – able to explain the observations. Moreover, as long as the number of flux samples (n) is sufficiently large, it may also yield an overview of regions in the flux space that are competitive in terms of the residual value. The next step in the overall ^{13}C-MFA procedure is the closer inspection and characterization of these regions, which is usually accompanied with modifying the model, e.g., by integrating additional

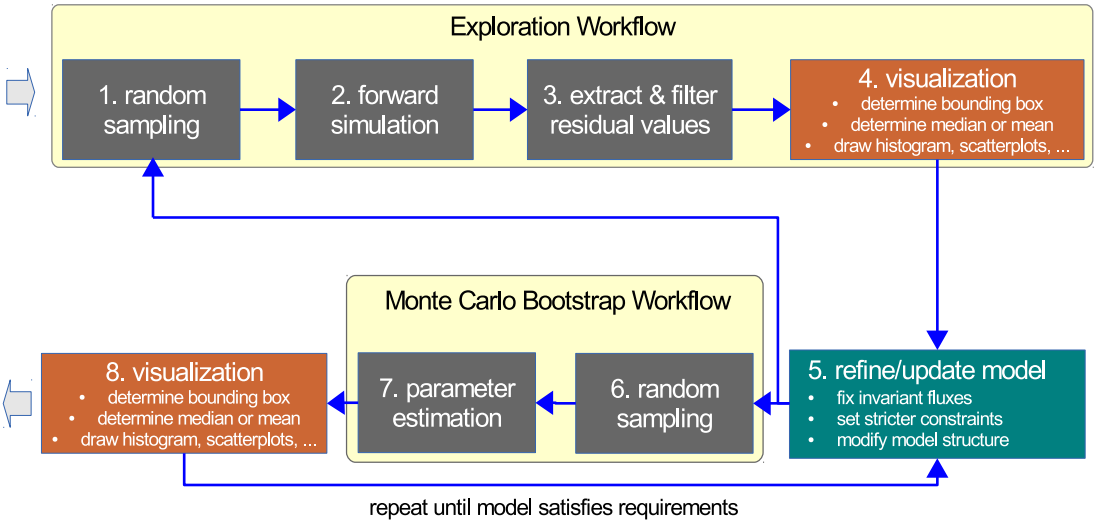


Figure 6.9.: Overview of the interactive ^{13}C-MFA workflow. Gray boxes represent automated tasks, while reporting tasks are colored orange. Arrows indicate the flow of control. Step 5 (turquoise) designates an interaction performed by the scientist. The exploration workflow described in § 6.4 is embedded as sub-workflow (upper branch). The Monte Carlo bootstrap sub-workflow (steps 6-7) is employed as described before (Dalman et al., 2013). The overall workflow includes several iterative steps and depends on the scientist's decisions upon inspecting intermediate results.

biochemical knowledge or updating measurements. In contrast to the sampling-based explorative workflow described before, here a targeted optimization-driven strategy is utilized which eventually supplies information on flux (non-)identifiabilities (Raue et al., 2011). The latter step is vitally important for any ^{13}C-MFA because it signals whether measured data actually contain the information that is needed to reliably estimate the unknown fluxes.

Due to its conceptual simplicity, the Monte Carlo bootstrap method is utilized as non-linear statistical method of choice in combination with a multi-start heuristic (Efron and Tibshirani, 1993; M. Joshi, Seidel-Morgenstern, and Kremling, 2006). If flux parameters are found to be non-identifiable, either new measurement information has to be added or, because such additional observations are rarely available in practice, the non-identifiable fluxes have to be eliminated from the model (Raue et al., 2011). As fluxes may be correlated, the elimination process must be done in an iterative manner. The proposed interactive ^{13}C-MFA workflow consists of the following steps, as shown in fig. 6.9:

- The exploration workflow (steps 1-4) is performed to gain a basic "familiarity" with the model as described before (cf. § 6.4).

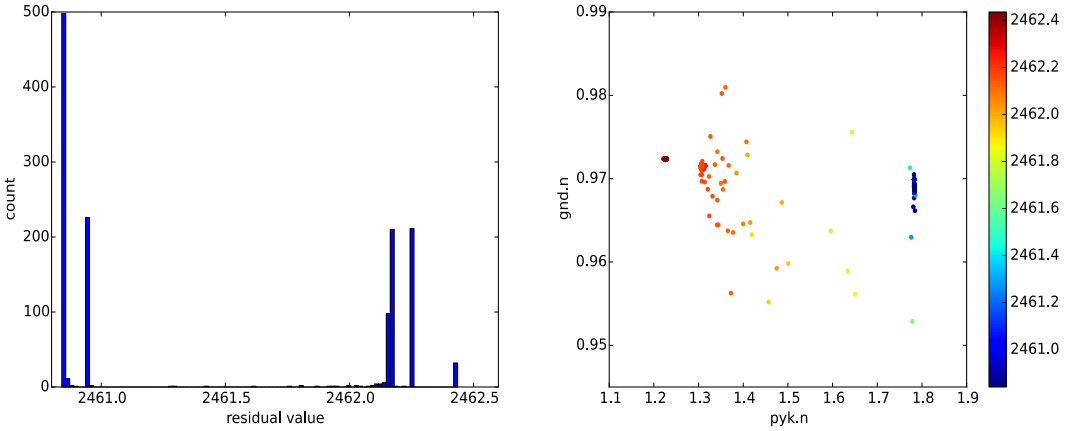


Figure 6.10.: Visualizations generated in the course of an interactive refinement workflow. Left: residual distribution of 1,000 flux fits with one of the interim *C. glutamicum* model variants. Two solution clusters are visible that differ only slightly in their residual values (2,460.8 and 2,462.2). Right: scatter plot of glucose-6-phosphate dehydrogenase (gnd), and pyruvate kinase (pyk) fluxes. The color of the dots codes the residual value. The visualization reveals that the flux solutions that underlie the two clusters have significantly different pyk fluxes. Moreover, while the flux value of pyk is dispersed for higher residual values, it is much more concise in case of the low residual solution cluster. The plot provides an indication for the multimodality of the nonlinear least-squares problem.

- With the visualizations of the exploration results, e.g., the distribution of residual values and various scatter plots (cf. fig. 6.10) at hand the researcher updates and refines the model according to the results if necessary (step 5). To assess the impact of changes made, the scientist may return to step 1 before continuing with step 6.
- The Monte Carlo bootstrap algorithm is applied with the best flux samples (steps 6 and 7).
- The results of the bootstrap are analyzed and visualized (step 8).
- After updating the model (step 5), the researcher may choose to restart from step 6 (or step 1) until the non-identifiable fluxes are removed from the model.

This use case highlights the iterative and interactive nature of typical ^{13}C-MFA modeling workflows. Notably, the single sub-workflow steps have quite heterogeneous runtime profiles: compute-intensive and long-running bootstrap executions alternate with inexpensive analysis tasks. Amazon's EMR cloud service is an elegant and straightforward way to solve compute intensive and embarrassingly parallel tasks like the Monte Carlo

bootstrap. Nevertheless, it does not make sense in all cases to await the final result of long-running processes. For instance, flux estimation processes often show no significant improvement in the residual value. In such cases, the run can be stopped prematurely. With the provenance logging module, it is possible to safely interrupt processes, saving time and money.

Methods

As indicated in fig. 6.9, the computational parts of the workflow are reused from other use cases, i.e., the exploration workflow and the MCB workflow. These sub-workflows are accessed via their web service interfaces. The model refinement and update task (step 5) is purely user-driven. Similarly, the researcher decides at the end of step 8 whether the outcome is sufficient, or further iterations are required. In addition to providing visualizations, the matplotlib scripts (`plot_histogram`, `plot_scatter`) also compute statistical moments of the results, e.g., minimum, maximum, mean, standard deviation, or median. Because the output file format (CSV) of steps 1-3 is equal to the MCB output (steps 6-7), the visualization scripts (steps 4 and 8) are effectively the same.

Results

Reusability is regarded as one of the most important drivers for service-oriented solutions (Josuttis, 2007). As already stated, ^{13}C-MFA workflows rely on many recurring tasks. These tasks may be seen as standard components either on atomic or already assembled level. In this particular use case, it took four iterations of the iterative ^{13}C-MFA workflow. In each iteration of the workflow, non-identifiable fluxes are identified and eliminated one by one from the set of unknowns. Additionally, information on (potentially locally) optimal solutions of the least-squares regression is continuously gathered.

Discussion

By clearly implementing web service interfaces it becomes easy to assemble these tasks in the wanted order, possibly by filling the missing gaps in between by writing scripts. In this use case, it was shown how previously developed workflows (namely, the MCB simulation workflow and the exploration workflow) are reassembled to a full-fledged interactive ^{13}C-MFA application. The described workflow can be seen as a chassis workflow, which is extensible by additional sub-workflows, e.g., the identification of fluxes is further automated by applying the X-means clustering tool (Pelleg and Moore, 2000) to the final outcome of the workflow (cf. fig. 6.11).

Likewise, by exchanging the simulation tool and visualization applications the overall workflow can be flexibly modified to fit to a different context. Hence, the scientist is supported in managing the sequence of "daily life" steps.

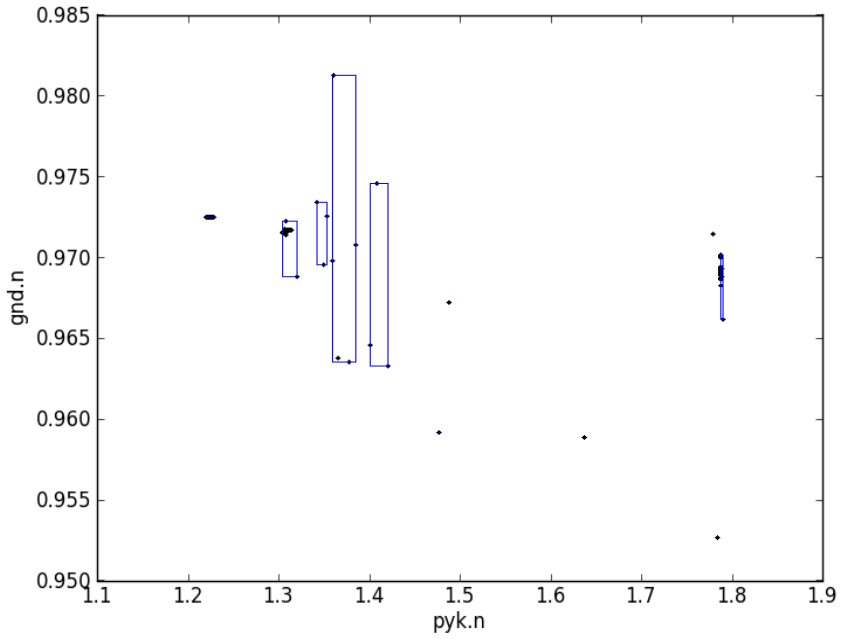


Figure 6.11.: Identified clusters (blue boxes) of the *C. glutamicum* model. The cluster image is generated using the *X-means* tool version 1.15 (Pelleg and Moore, 2000), while the frame and axes are drawn with matplotlib (cf. fig. 6.10; right). X-means is called with `kmeans kdtree in pyk_gnd_v4.csv -D_DRAWPOINTS -D_INTERACTIVE`, where the CSV file *pyk_gnd_v4.csv* contains the workflow outcome after four iterations.

Chapter 7.

Conclusions and Discussion

While the principle ^{13}C-MFA procedure is seemingly straightforward, the specific steps undertaken by a researcher are often driven by modeling decisions that are dictated by the specific biological question under study, the observed data at hand, and computational considerations (Niedenführ, Wiechert, and Nöh, 2015). Thus, building high-quality ^{13}C-MFA workflows requires modeling expertise and familiarity with a broad range of specialized software tools. The examples presented illustrate the diversity of ^{13}C-MFA applications, which range from the modeling of complex microorganisms to sophisticated statistical analyses of large-scale simulation data.

Following the conventional definition of a software framework (R. E. Johnson and Foote, 1988), the outcome of this thesis is *an abstract design for solutions to a family of related problems* (rather than a mere collection of libraries and tools), which aims at providing solutions to master the ^{13}C-MFA procedure. Exposing special-purpose ^{13}C-MFA tools and data sources as services with unified interfaces allows for flexibly composing computational pipelines and workflow applications. Hence, the approach proposed by this thesis – namely, designing the SWF as a collection of loosely-coupled modules that are glued together with web services – supports scientists in the realization of ^{13}C-MFA workflows. Specifically, the five challenges identified in § 1.3 are addressed in this thesis as discussed in the following.

7.1. Discussion

C1: Heterogeneous and flexible data and tool organization
A usable SWF for model-based evaluations needs to strike a balance between being a mere collection of services and providing fully-integrated functionality, such as graphical modeling and easy-to-use HPC deployment. The chosen design of the SWF allows that both, the use of web services, which allow the flexible integration of otherwise diverse applications, and the introduction of a VCS in addition to traditional databases, help to organize different knowledge sources, experimental data, and models.

However, this flexibility also comes at a price: often, the automation of ^{13}C-MFA workflows utilizing third-party components is only seldom possible (if at all) to full extent out of the box. Instead, the researcher needs to create a workflow that employs the software and determine which intermediate information requires an expert decision. In addition, input and output formats of some of the employed tools need to be adapted

to adjacent workflow steps. While especially third-party bioinformatics software often employ custom formats, a plenitude of data conversion libraries and utilities are available that are ready for inclusion into the SWF. For instance, the biopython package provides a broad range of Python interfaces for prominent bioinformatics tools including data conversion utilities (Cock et al., 2009). As a result, the utilization of third-party tools in ^{13}C-MFA workflows is supported by the SWF as shown in the various use cases.

C2: Standardization of processes and unifying software tools and data
Practical experience shows that many sub-tasks are repetitive and can be performed following standardized procedures. The SWF addresses this challenge with several measures:

- Organizing all electronic information of a ^{13}C-MFA study using the SWF project repository template allows researchers to quickly structure input, output, and other data of a workflow. Moreover, this standardization eases the exchange of information with other scientists.

- The unification of data formats to a core set (namely, FluxML, FWDSIM, HDF5, and CSV) in combination with the utilization of web service interfaces eases the composition of different tools (e.g., Omix and 13CFLUX2) to semi-automatic workflows.

- As demonstrated in the various use cases, several components of the SWF are designed to be reusable in different contexts (e.g., the cloud Hadoop MapReduce solution of the MCB method). Besides being proof-of-concept implementations, the presented use cases employing these standard components can be considered as blueprint solutions (or design patterns) for problems of similar category.

C3: Interactive research-driven workflow steering
The importance of the researcher in typical ^{13}C-MFA studies cannot be emphasized enough. The presented use cases involve the activity of an expert to direct the paths of a workflow. Several components of the SWF are designed to ease the work of the researcher. The web portal acts as resource and database central, the web services provide a programmable (that is, *scriptable* using the Python language) ad hoc workflow interface, with Hadoop MapReduce a readily usable distributed computing framework is accessible, the provenance collection framework offers online tracking of simulation data, the VCS and its template is utilized to organize researcher data, and numerous (in parts third-party) tools and libraries allow the creation of custom ^{13}C-MFA applications.

C4: Distributed Computing
With the increasing complexity of metabolic reaction network models, there is a clear need for a solution to deploy ^{13}C-MFA simulations on distributed computing resources. In this work, the Apache Hadoop MapReduce framework is chosen as central component in the SWF to realize performance-intensive simulation tasks. On the example of the MCB algorithm, the utilization of cluster and cloud computing resources is demonstrated in multiple use cases. While the development and adaptation of MapReduce workflows

is comparatively easy to realize, the performance measurement results also indicate that there is room for improvement. Several measures, namely, the parameterization of the Hadoop MapReduce framework and tuning of the MCB algorithm, have boosted the overall runtime performance, thus, enabling the efficient work on large-scale ^{13}C-MFA models.

C5: Service orientation
Service orientation is not only a technical aspect, but regarded as a paradigm in the design of the SWF. For instance, a biotechnologist working on a *C. glutamicum* organism most likely wants to focus on the experimental steps rather than the details of the ^{13}C-MFA simulation tasks. Hence, understanding the computational tools of the ^{13}C-MFA procedure as services, purely technical steps (such as data conversion or deployment on HPC resources) can be hidden from the researcher. In this work, 13CFLUX2 and other ^{13}C-MFA tools are identified (cf. fig. 5.1) and embedded into the SWF using web services and foreign programming language integration (e.g., Java and Python).

The use cases have demonstrated that this approach is feasible, however, a performance penalty and the increased memory usage compared to pure 13CFLUX2 simulations is also observed. However, especially in interactive workflows (e.g., modeling with Omix), the availability of 13CFLUX2 tools as web service is an invaluable advantage because the researcher is effectively liberated from several tasks such as file transfer and data conversion.

7.2. Lessons Learned

Several insights are gained in the course of the development of the SWF that go beyond the requirements in the ^{13}C-MFA context.

- With a series of critical IT security issues being disclosed recently[1], the necessity of employing appropriate cryptographical methods (e.g., secure web communication via TLS or RSA public key data encryption) and the utilization of role-based authentication to protect user data is highlighted.

- When this work started in 2008, SOAP-based web services were the de-facto industry standard for providing machine-to-machine communication over the Internet. The WS-BPEL standard is largely based other industry-grade WS-* standards, making a standards-conform implementation of a workflow framework a complex endeavor. Since then, REST has become a quasi-standard in web service and cloud applications, often combined with a scripting language such as Python or Ruby to develop lightweight workflow applications. For instance, the Amazon Web Service SDK

[1] Most prominently, the *Heartbleed* bug rendered applications using the OpenSSL library practically useless against security threats. See `http://heartbleed.com/`. For a detailed overview of important software vulnerabilities the author refers to the *Common Vulnerabilities and Exposures* list (CVE): `https://cve.mitre.org/`.

provides a comprehensive Ruby client library employing REST interfaces. Likewise, upcoming versions of the Java programming language are increasingly emphasizing the importance of REST for web and cloud applications (Evans, 2014). Therefore, the focus of the SWF development has shifted over the years from BPEL and SOAP to REST and scripted Python workflows.

- In the last decades, open source software has gained much interest in professional applications (Raymond, 1999). Especially, the *bazaar-style* decentralized approach of software and knowledge sharing has quickened the development of successful bioinformatics platforms (e.g., NCBI, myExperiment), web databases (e.g., KEGG, BioCyc), and scientific bio tools (e.g., blast, bioperl/biopython) (Hope, 2008). Hence, the professional use of open source development tools and libraries can be a beneficial addition to purely commercial solutions[2].

7.3. Outlook

There are several opportunities for future development:

- Because the tools developed at FZJ are currently being extended to the instationary ^{13}C-MFA method, the established workflows (and their use with the existing tools) will be revised and evaluated and, eventually, adapted and further generalized. The service-oriented design of the SWF allows a quick adoption of novel methods and tools in the ^{13}C-MFA context.
- Today, experimental ^{13}C-MFA data is mostly analyzed outside the biolabs. With the ad hoc availability of distributed ^{13}C-MFA simulation services and cheap mobile devices, such as smart phones and tablets, simulations and data analyses can be conducted while performing biological experiments. This is especially important in labs with high biosafety levels (Bente et al., 2011). To use the SWF on touchless mobile devices, further research to adapt the specific workflows and user interfaces (e.g., the web portal) is required.
- Another area is simulation performance in distributed environments. While the examples impressively demonstrate a performance boost by employing HPC resources, there is still room to improve the parallel efficiency of the simulation workflows.

[2] The author has not only benefited from open source software during the development of the SWF, but also contributed to open source projects. See for example: `http://man7.org/linux/man-pages/changelog.html` and `http://sourceware.org/ml/libc-alpha/2012-02/msg00444.html`.

Appendix A.

Mathematical Background

In the following, the mathematical background regarding MCB in the context of ^{13}C-MFA is briefly described.

A.1. Notation

Scalar variables are written in italic letters, while vectors are denoted with boldface letters. The length or norm of a variable a is denoted with $\|a\|$. The hat descriptor denotes an estimated variable or function, e.g., $\hat{a}$.

A.2. Estimation of Free Flux Parameters

The mathematical foundations of the ^{13}C-MFA parameter estimation procedure is briefly introduced now. The details can be reviewed in the literature (e.g., Weitzel, 2009[§§ 2.6 and 4.1 and 10.1]). The isotope-based ^{13}C-MFA aims at estimating the vector of free fluxes $\hat{\mathbf{v}}_f$ which describes the labeling measurement vector $\mathbf{y}_m$, such that:

$$\hat{\mathbf{v}} = arg \min_{\hat{\mathbf{v}}_f} D(\hat{\mathbf{v}}_f) \tag{A.1}$$

The objective function D is defined as the residual sum of squares between the real measurements $\mathbf{y}_m$ and the weighted simulated measurements $\mathbf{y}_{s,\omega}$. With ω being a measurement group scaling factor and σ_i^2 the standard deviation of the i^{th} measurement $y_{m,i} \in \mathbf{y}_m$.

$$\|\mathbf{y}_m - \mathbf{y}_{s,\omega}\|^2 = \|\mathbf{y}_m - \omega \cdot \mathbf{y}_s\|^2 = \sum_i \frac{(y_{m,i} - \omega \cdot (y_{s,i}(\mathbf{v}_f))^2}{\sigma_i^2} \tag{A.2}$$

A.3. Monte Carlo Bootstrap

In this work, the parametric MCB approach is employed to estimate the accuracy of parameters (e.g., the above mentioned flux values) and statistical errors (Efron and Tibshirani, 1993). Let θ be a parameter of the distribution D. Because D and θ are unknown, an estimation $\hat{\theta} = s(\mathbf{x})$ is derived with $s(\cdot)$ being an estimation function (cf. eq. (A.2)).

$\mathbf{x} = (x_1, ..., x_n)$ is a random sample from the model density function $f_{\hat{\theta}}$. The MCB algorithm for estimating error parameters is then formulated as follows:

1. N bootstrap samples are drawn (i.e., generated from the parametric model):

$$f_{\hat{\theta}} \rightarrow \mathbf{x}^{*i} \text{ for } i = 1, 2, ..., N \tag{A.3}$$

2. For each sample i, perform an estimation of the distribution's parameter:

$$\hat{\theta^*}(i) = s(\mathbf{x}^{*i}) \tag{A.4}$$

3. Compute mean and standard deviation for $\hat{\theta}^*$:

$$\hat{\mu}_{\hat{\theta}^*} = \frac{\sum_{i=1}^{N} \hat{\theta}^*(i)}{N} \tag{A.5}$$

$$\hat{\sigma}^2_{\hat{\theta}^*} = \sum_{i=1}^{N} \frac{[\hat{\theta}^*(i) - \hat{\mu}_{\hat{\theta}^*}]^2}{(N-1)} \tag{A.6}$$

This algorithm yields the ideal bootstrap estimate as N approaches infinity.

Appendix B.

Source Code Listings

B.1. 13CFLUX.net front site JSP source code

```
<%@taglib uri="/cms" prefix="cms" %>

<cms:page site="13cflux-front">
  <cms:pageAttr title="Welcome to the 13CFLUX.NET Portal" />
  <p><div>The Modeling and Simulation group of the IBG-1 has a strong focus on
      the design of specific tools for modeling, simulating and visualizing
      biochemical networks and for improving interdisciplinary communication.</
      div></p>
  <p>Currently, the following software packages are actively supported:</p>
  <table>
    <tr>
      <td><img alt="flux2" src="/images/logo_13cflux2_hires.png" width='250'/>
        <p><div><a href="/13cflux2/">13CFLUX2</a> the new high-performance
            simulator for quantifying intracellular fluxes by analysis of carbon
            labeling experiments</div></p>
      </td>
      <td><p></p></td>
      <td><img alt="omix" src="/images/logo_omix.png" width='200'/>
        <p><a href="/omix/">OMIX</a> a highly customizable visualization tool and
             biochemical network editor</p>
      </td>
    </tr>
    <tr>
      <td><img alt="biopda" src="/images/logo_biopda.png" width='200'/>
        <p><a href="/biopda/">BioPDA3</a> an easy-to-use tool for online process
            data analysis</p>
      </td>
    </tr>
  </table>
</cms:page>
```

Listing B.1: Source code for the 13CFLUX.net front page.

B.2. HDF5ToCSV

```
#include "HDF5ToCSV.h"
#include <charptr_array.h>
#include <HDF5Reader.h>
#include <GLabelMatrix.h>
#include <Error.h>
#include <NLgetopt.h>
#include <iostream>
#include <cstdlib>

struct HDF5ToCSVParams {
  const char*             in;
  const char*            out;
  const char*           data;
  const char*        columns;
  bool             no_header;
  const char*          delim;
  charptr_array logpublishers;

  HDF5ToCSVParams()
  : in(NULL),
    out(NULL),
    data("/flux/data"),
    columns("/flux/names"),
    no_header(false),
    delim(",")
  {
    logpublishers.add("fd:2");
  }
};

using namespace flux;

static void read_HDF5_matrix( la::GLabelMatrix< double >& M, const std::string&
    hdf5file,
  const std::string& data, const char* columns )
{
  la::HDF5Reader h5r(hdf5file);
  h5r.read(M, data, NULL, columns);
}

static void write_CSV_file( const la::GLabelMatrix< double >& M,
  const char* csv, const std::string& delim )
{
  FILE* fd = stdout;
  if (csv) {
    fd = ::fopen(csv, "w");
```

```
    if (!fd) {
      fERROR("Failed␣to␣open␣%s␣for␣reading!", csv);
      exit(EXIT_FAILURE);
    }
  }
  if (M.getColumnLabels()[0]) {
    fprintf(fd, "\"%s\"", M.getColumnLabel(0));
    for (size_t i = 1; i < M.cols(); i++) {
      fprintf(fd, "%s\"%s\"", delim.c_str(), M.getColumnLabel(i));
    }
    fprintf(fd, "\n");
  }
  for (size_t i = 0; i < M.rows(); i++) {
    fprintf(fd, "%f", M.get(i, 0));
    for (size_t j = 1; j < M.cols(); j++) {
      fprintf(fd, "%s%f", delim.c_str(), M.get(i, j));
    }
    fprintf(fd, "\n");
  }

  if (fd != stdout) {
    ::fclose(fd);
  }
}

int main(int argc, char** argv) {
  SETLOGLEVEL(logINFO);
  HDF5ToCSVParams params;
  parse_command_line(argc, argv, params); // function omitted here
  la::GLabelMatrix< double > M;
  read_HDF5_matrix(M, params.in, params.data, params.columns);
  write_CSV_file(M, params.out, params.delim);
  exit(EXIT_SUCCESS);
}
```

Listing B.2: Source code for the HDF5ToCSV tool. Because the HDF5 and 13CFLUX2 libraries are used, the complete implementation requires 80 lines of code. Parameter parsing and help text codes are omitted in the listing for the sake of brevity.

B.3. Iterating many files on Linux

Because standard Linux tools (e.g., `rm`, `find`, or `ls`) employ non-constant time directory iteration algorithms, a simple C++ program is developed fill this gap (Listing B.3).

```
static char buf[65536];

```

```
int main(int argc, char* argv[]) {
  const char* path = argv[1];
  DIR* dir = opendir(path);
  size_t count = 0;
  struct dirent* dent;
  while ((dent = readdir(dir))) {
    /* skip non-regular files */
    if (dent->d_type != DT_REG) {
      continue;
    }
    count++;
    std::string fname(path);
    fname += std::string("/");
    fname += std::string(dent->d_name);
    std::stringstream strm;
    strm << "./process_fwdsim.sh " << fname << " " << count << std::ends;
    int ret = system(strm.str().c_str());
    if (WEXITSTATUS(ret) != EXIT_SUCCESS) {
      continue;
    }
  }
  closedir(dir);
  exit(EXIT_SUCCESS);
}
```

Listing B.3: C++ program to iterate large amounts of FWDSIM files in a directory.

Listing B.4 presents the `process_fwdsim.sh` script which simply invokes the open source `libxml` tool `xmllint`.

```
OUT=simdata.csv
cat "$1" | tr '\n' ' ' > tmp_fwd
echo -e "$(basename $1)_$2\t$(xmllint --noblanks --exc-c14n tmp_fwd)" >> $OUT;
rm -- $1
```

Listing B.4: Shell script which is invoked by `generate_fwdsim.cc` to execute `xmllint`.

B.4. FWDSIM Post-Processing with Hadoop Streaming API

Listing B.5 depicts the implementation of the `map` function.

```
import sys
import xml.etree.ElementTree as ET
NS = 'http://www.13cflux.net/fwdsim'

for line in sys.stdin:
  (key, value) = line.split('\t', 1)
  (mod, seq) = key.rsplit('_', 1)
```

```
8   F = ET.XML(value)
9   M = F.find('{%s}measurements' % NS)
10  res = M.get('residual')
11  print "%s\t%s␣%s" % (mod, seq, res)
```

Listing B.5: Realization of the Hadoop MapReduce `map` program in Python.

For each input line, `map` separates key and value strings (lines 5–6). From the key, the model or experiment name (*mod*) and the Monte Carlo sequence number (*seq*) are extracted (line 7). Using the Python XML parser, the residual value (*res*) is read (lines 8–10). Finally, the `map` program writes the extracted values to standard out.

After combining all outputs from `map` by unique keys, the Hadoop framework invokes a `reduce` processor with a key and a list of values. Thus, `reduce` is realized as shown in Listing B.6.

```
1  import sys
2  residual_threshold = 10000.
3
4  for line in sys.stdin:
5    (mod, value) = line.split('\t', 1)
6    (seq, res) = value.split('␣', 1)
7    res = float(res)
8    if res < residual_threshold:
9      print "%s_%s,%f" % (mod,seq,res)
```

Listing B.6: Python implementation of the Hadoop FWDSIM filter reducer.

Like `map`, `reduce` extracts key and value from the standard input stream (line 4–5). From the value field, the space-delimited values *seq* and *res* are extracted (lines 6–7). Finally, if the *res* is below the threshold, the unique input key from `map` is reassembled from *mod* and *seq* and returned with the residual value (lines 8–9).

B.5. Synchronous and Asynchronous Python SOAP web service clients

This section demonstrates the utilization of *FluxWS* using a Python SOAP web service client in detail. By using the Python web service library SUDS, the 13CFLUX2 web services are accessed conveniently from the researcher's local workstation (Ortel, Noehr, and Gheem, last accessed: Mar 2016). For instance, the following client invokes a `fitfluxes` simulation:

```
1  import sys;
2  import base64;
3  from suds.client import Client;
4
5  fml_f  = open(sys.argv[1], 'rb');
6  fluxml = fml_f.read();
```

```
7  fml_f.close();
8  fml    = base64.b64encode(fluxml);
9
10 F = Client(flux_wsdl_URL);
11 R = F.service.Fitfluxes(fml, None);
12 fwdsim = base64.b64decode(R);
```

Listing B.7: Python web service wrapper client. In this example, the 13CFLUX2 parameter estimation program `fitfluxes` is executed.

The Python web service client script shown in Listing B.7 reads a model file (line 5-7), encodes the contents to base64 in line 8 (fml), and initializes the SUDS library with the web service URL (line 10). The 13CFLUX2 program `fitfluxes` is called in line 11. Because the outcome of the web service call is a base64 encoded document (R), the result data is decoded to an XML string in line 12.

However, the execution of the Python script is non-deterministically blocked until the simulation is finished (line 11). To leverage the handling with long-running processes, this so-called synchronous invocation is substituted by the following sequence of web service calls (cf. Listing B.8):

```
1 C = F.service.FitfluxesStart(fml, None);
2 F.service.waitFor(C, 30000);
3 R = F.service.getResult(C);
4 F.service.delete(C);
```

Listing B.8: Replacement calls for the asynchronous 13CFLUX2 `fitfluxes` web service client.

After starting the simulation asynchronously, a persistent job context (C) is created (line 1). To check the job status, the *waitFor* web service is called (line 2). If the method returns within the given time threshold (here 30,000 ms), the simulation outcome is returned by *getResult* (line 3); otherwise a timeout exception is thrown. Because C is persistent across different program invocations, the *delete* web service method is invoked at the end of the script (line 4).

B.6. RepoManager Web Service

```
1 import sys
2 import pysvn
3 import os
4 import shutil
5 from bottle import route, run, template
6 import subprocess
7
8 # Bind address and port of the REST service
9 host = 'localhost'
```

```
port = 8080

proto = 'file://'
repo_root = '/var/subversion/SWF_repositories'

# create a new repository on this server
# client call:
#   curl http://<host>:<port>/init/repo
@route('/init/<repo>')
def initialize(repo):
  subprocess.check_output("svnadmin create %s/%s" % (repo_root, repo),
                          shell=True )

# list repositories on server
# client call:
#   curl http://<host>:<port>/list
@route('/list')
def list():
  return os.listdir(repo_root)

# list repositories on server
# client call:
#   curl http://<host>:<port>/list
@route('/list/<repo>')
def list(repo):
  client = pysvn.Client()
  return client.list(proto + repo_root + '/' + repo)

# add and initialize repository project paths
# client call:
#   curl http://<host>:<port>/add_project/repo/myproject
@route('/add_project/<repo>/<name>')
def add_project(repo, name):
  client = pysvn.Client()
  try:
    client.checkout(proto + repo_root + '/' + repo, repo)
    project_name = repo + '/' + name
    client.mkdir(project_name, 'Initial create')
    client.mkdir(project_name + '/raw_data', 'Initial create')
    client.mkdir(project_name + '/results', 'Initial create')
    client.mkdir(project_name + '/protocols', 'Initial create')
    client.mkdir(project_name + '/provenance_data', 'Initial create')
    client.mkdir(project_name + '/tools', 'Initial create')
    client.mkdir(project_name + '/notes', 'Initial create')
    client.checkin([repo], 'Initial repository checkin')
  except Exception as e:
    shutil.rmtree('./' + repo)
    raise e;
```

```
  shutil.rmtree('./' + repo)

# Start REST web service
run(host=host, port=port)
```

Listing B.9: Source Code of the *RepoManager* RESTful web service. The service provides access to the SWF project repositories.

B.7. Boxplot Script for the BCG Comparison Use Case

```
#!/usr/bin/env python

import matplotlib
matplotlib.use('Qt5Agg')

import matplotlib.pyplot as mp
import csv
import numpy
import sys
import gzip

fluxnames = ['emp1', 'ana1', 'tca2']

for flux in fluxnames:

  # slow
  fp = open('BCG_slow.csv')

  csvfile = csv.reader(fp, delimiter=',')
  data_net_slow = []
  count = 0
  header = []

  for line in csvfile:
    if count == 0:
      header = line
      index_n = header.index(flux + '_S.net');
      print(header)
    else:
      val = float(line[index_n])
      data_net_slow.append(val)
    count += 1

  # fast
  fp = open('BCG_fast.csv')

  csvfile = csv.reader(fp, delimiter=',')
```

```
  data_net_fast = []
  count = 0
  header = []

  for line in csvfile:
    if count == 0:
      header = line
      index_n = header.index(flux + '_F.net');
      print(header)
    else:
      val = float(line[index_n])
      data_net_fast.append(val)
    count += 1

  mp.rcParams.update({'font.size': 16})
  mp.boxplot([data_net_slow, data_net_fast])
  mp.xticks([1, 2], [flux + '.net␣slow', flux + '.net␣fast'])

  mp.ylabel('flux␣value')
  mp.savefig(flux + '.svg')
```

Listing B.10: Source Code of the boxplot script which is used in the BCG comparison use case.

B.8. Sampling and Multicore Simulation Script

```
import sys
import subprocess
import numpy as np
import h5py

if len(sys.argv) < 4:
  print("Usage:␣%s␣<fml>␣<N>␣<BESTM>" % sys.argv[0])
  exit(-1);

fml      = sys.argv[1]
N        = int(sys.argv[2])
bestM    = int(sys.argv[3]) # m best residuals

if bestM > N: bestM = N

samples  = 'samples.hdf5'
fwdout   = 'fwdout.hdf5' # filename for simulation output

def getResidualsFromHDF5(f):
  data = f['/optimizer/residual'][0];
  return list(zip(range(len(data)), data))
```

```
22
23
24 def getFluxesFromHDF5(f):
25   names = f['/flux/names'][0].astype(np.str_)
26   data  = fwdhdf5['/flux/data']
27   return (names, data)
28
29
30 def printFluxes(residuals, fluxes):
31   N = bestM
32   D = len(fluxes[1])
33
34   # print header
35   print("residual", end="")
36   for j in range(D):
37     print(",␣%s" % fluxes[0][j], end="")
38   print()
39
40   for i in range(N):
41     print("%f" % residuals[i][1], end="")
42     for j in range(D):
43       print(",␣%f" % fluxes[1][j][i], end="")
44     print()
45
46
47 subprocess.check_output("ssampler␣-i␣%s␣-o␣%s␣-n␣%d␣-l␣/dev/null" % (fml,
       samples, N), shell=True)
48 subprocess.check_output("multifwdsim␣-i␣%s␣-H␣%s␣-f␣%s␣>/dev/null␣2>&1" % (fml,
       samples, fwdout), shell=True)
49
50 fwdhdf5 = h5py.File(fwdout, "r")
51
52 residuals = getResidualsFromHDF5(fwdhdf5)
53 residuals.sort(key = lambda tup : tup[1])
54 fluxes = getFluxesFromHDF5(fwdhdf5);
55 printFluxes(residuals, fluxes)
```

Listing B.11: Source Code of the sampling and simulation script. The script calls the 13CFLUX2 programs *ssampler* and *multifwdsim* to generate random samples and forward simulation results, respectively. Ignoring whitespaces and sanity checks, the script only has 37 lines or code.

B.9. Histogram Plotting

```
1 import matplotlib
2 matplotlib.use('Qt5Agg')
```

```
3
4  import matplotlib.pyplot as mp
5  import csv
6  import numpy
7  import sys
8  import gzip
9
10 fp = None
11 if sys.argv[1].endswith('.gz'):
12   fp = gzip.open(sys.argv[1])
13
14 if fp == None:
15   fp = open(sys.argv[1])
16
17
18 csvfile = csv.reader(fp, delimiter=',')
19 res = []
20 minres = None
21 maxres = None
22 count = 0
23
24 idx = (int(sys.argv[2]) if len(sys.argv) > 2 else 0)
25 label = (sys.argv[3] if len(sys.argv) > 3 else 'residual value')
26
27 header = next(csvfile)
28
29 for line in csvfile:
30   val = float(line[idx])
31   res.append(val)
32   if minres == None or minres > val:
33     minres = val;
34   if maxres == None or maxres < val:
35     maxres = val;
36   count += 1
37
38 print("range [%f %f]" % (minres, maxres))
39 print("median: %f" % numpy.median(res))
40 print("mean: %f +/- %f" % (numpy.mean(res), numpy.std(res)))
41 print("count: %d" % count)
42
43 mp.hist(res, 100)
44 mp.xlabel(label)
45 mp.ylabel('count')
46
47 mp.show()
```

Listing B.12: Histogram visualization script using Python and matplotlib.

B.10. Scatter Plot Visualization

```
import matplotlib
matplotlib.use('QT5Agg')

import matplotlib.pyplot as mp
import csv
import numpy
import sys
import gzip
import fluxes

max_samples = 2000.

csvfname = sys.argv[1]
x_idx = int(sys.argv[2])
y_idx = int(sys.argv[3])

fp = None
if csvfname.endswith('.gz'):
  fp = gzip.open(csvfname)

if fp == None:
  fp = open(csvfname)

csvfile = csv.reader(fp, delimiter=',')
res = []
x = []
y = []
cnt = 0

header = next(csvfile);

for line in csvfile:
  val = float(line[1])
  if cnt < max_samples:
    res.append(val)
    x.append(float(line[x_idx]))
    y.append(float(line[y_idx]))
    cnt += 1

print('Count: %d' % (cnt))
print('%d -> %s [%f, %f]' % (x_idx, header[x_idx], min(x), max(x)))
print('%d -> %s [%f, %f]' % (y_idx, header[y_idx], min(y), max(y)))

mp.scatter(x, y, c=res)
mp.xlabel(header[x_idx])
mp.ylabel(header[y_idx])
```

```
mp.jet();

mp.show()
```

Listing B.13: Scatter plot script using Python and matplotlib.

B.11. Scripts for the Data Exploration Use Case

```
FML=$1
N=$2
OUT=simdata.csv

mkdir -p fwdsimout

ssampler -i $FML -n $N -o samples.hdf5
multifwdsim -i $FML -H samples.hdf5 -o fwdsimout

i=0
for m in $(find fwdsimout -type f); do
  echo "Processing␣$m␣($i)␣...";
  cat $m | tr '\n' ' ' > tmp_fwd
  echo -e "$(basename␣${m})_${i}\t$(xmllint␣--noblanks␣--exc-c14n␣tmp_fwd)" >>
      $OUT;
  ((i++));
done

rm -Rf tmp_fwd samples.hdf5
```

Listing B.14: Shell script to prepare the deployment on a Hadoop MapReduce cluster. The script is tested with GNU Bash version 4.3.

```
import sys
import xml.etree.ElementTree as ET
NS = 'http://www.13cflux.net/fwdsim'

for line in sys.stdin:
  (key, value) = line.split('\t', 1)
  (mod, seq) = key.rsplit('_', 1)
  F = ET.XML(value)
  M = F.find('{%s}measurements' % NS)
  res = M.get('residual')
  print("%s\t%s␣%s" % (mod, seq, res))
```

Listing B.15: *Map* script used in the data exploration workflow.

```
import sys
residual_threshold = 10000.
```

```
3
4 for line in sys.stdin:
5   (mod, value) = line.split('\t', 1)
6   (seq, res) = value.split('␣', 1)
7   res = float(res)
8   if res < residual_threshold:
9     print("%s_%s,%f" % (mod,seq,res))
```

Listing B.16: *Reduce* script used in the data exploration workflow.

Appendix C.

13CFLUX2 Tools

C.1. Simulation Tools

- *fwdsim*: the forward simulation tool solves the MFA equations, i.e., given a flux vector and the FluxML model, isotopomers or cumomers are computed. The output data of this tool is represented in the FWDSIM format.

- *fitfluxes*: by utilizing a non-linear optimizer, *fitfluxes* solves the inverse parameter estimation problem of the MFA equations. The output of this tool is a FWDSIM file.

- *edopt*: with a given definition of input substrates, *edopt* computes an optimal design for subsequent ILEs (Möllney et al., 1999). The objective function is subject to a statistical optimality criterion which is applied to the covariance matrix of each sample (Atkinson and Donev, 1992). For each sample the flux covariance matrix is computed.

- *multifwdsim*: combined with the output from *ssampler*, this tool operates on a set of inputs rather than a single flux vector. Because the input model file is only parsed and validated once per sample, *multifwdsim* processes large amounts of flux vectors by magnitudes faster than simulating each flux with *fwdsim*. In addition, the flux vector processing can be parallelized on all processors (resp. cores) on a compute node. The output of this tool is either a directory containing FWDSIM files, or a comprehensive HDF5 data file.

- *multifit*: This Perl script performs multi-start parameter estimations by invoking *ssampler*, *fitfluxes*, and *setfluxes* in a multi-tasked fashion.

- *multifitfluxes*: like *multifwdsim*, this tool parallelizes multiple parameter estimations on a single compute node. Accordingly, the output of *multifitfluxes* is either a directory with FWDSIM files, or an HDF5 data file. Because the processing of FluxML data is minimized with *multifitfluxes*, it offers performance benefits compared to *multifit*.

C.2. Sampling and Analysis Programs

- *ssampler*: random fluxes within the constraints of a FluxML model are generated with this tool. The output is a HDF5 file containing samples.
- *sscanner*: this tool serves three main purposes. Firstly, by scanning the model stoichiometry and the model inequality constraints, the analytical center of the flux space is assigned to the model flux values. Secondly, unbound free fluxes are set with *sscanner*. Finally, user-defined and inferred flux inequality constraints are presented.
- *edscanner*: by sampling the given input substrate space, this tool generates a mixing simplex. For each sample the flux covariance matrix is computed. Unlike *edopt*, this tool performs a grid sampling method instead of applying an optimizer search in the mixing simplex space.
- *perturb*: measurement values of a FluxML model are randomly perturbed according to the measurement model and stored alongside the original measurement values in the model file. Thus, this tool is useful for performing statistical applications, like the computation of Monte Carlo bootstrap statistics (Weitzel et al., 2013).
- *multiperturb*: like *multifwdsim* and *multifitfluxes*, this tool extends *perturb* by generating multiple random measurements from a single input. Likewise, the output is either a directory containing multiple FluxML files, or a single HDF5 containing measurement vectors.

C.3. Conversion Utilities and Reporting Tools

- *collectfitdata*: this helper utility collects specific information from multiple FWDSIM input files into a single HDF5 output file. Currently, measurements, free fluxes, netto fluxes, and exchange fluxes can be selected for collection.
- *ftbl2fml*: model files from the 13CFLUX format FTBL are converted to 13CFLUX2 FluxML using this tool Wiechert et al., 2001.
- *fml2sbml*: this tool converts FluxML model files to the SBML format (Hucka et al., 2003). Unlike SBML, FluxML supports a notation for specifying labeling positions and atom transitions. Hence, this information is lost when converting FluxML models to SBML.
- *fmllint*: FluxML files are checked with *fmllint*. The XML models are syntactically checked and validated against the FluxML schema.
- *fmlstat*: FluxML model statistics (e.g., number of reactions and pools) are displayed with this tool.

- *fwdsim2csv*: this tool converts flux values, measurement norms, and optimization residuals from FWDSIM files to a CSV formatted output. Thus, FluxML measurements are easily exported to table spreadsheet programs.
- *fwdsimflt*: like *fwdsim2csv*, this program converts data from FWDSIM to a CSV file. However, the tools differ in two aspects: (a) *fwdsimflt* is a conversion tool from measurements, flux values, stoichiometry, standard deviations, unknown parameters, optimization flags, and isotopomer fractions in a CSV formatted output; (b) while *fwdsim2csv* is designed to process multiple files into a single CSV output, *fwdsimflt* generates the CSV data row-wise, i.e., a FWDSIM input file is converted to a single CSV output file.
- *sbml2fml*: this tool converts SBML model files to the FluxML format. Thus, it is the reverse operation of *fml2sbml*. However, the conversion data is restricted to the stoichiometry.
- *simreport*: Generate a report of a simulation by analyzing FWDSIM files. The generated output are simulated measurement values, norm values and differences between model measurement values and simulated data.

Appendix D.

Apache HTTPD Configuration

The Apache HTTPD web server is used as dispatcher for all incoming HTTP messages. Thereby, the Apache HTTPD rewrite engine[1] is employed to either handle incoming request or forward to the JBoss application server. The decision is made on the accessed resource address (Uniform Resource Locator; URL). The following URL substitution rules are configured in the HTTPD web server:

- To validate FluxML and FWDSIM XML files, so called XML Schema Definitions (XSD) are employed. In the SWF, the following configuration entry is used to access the FluxML and FWDSIM XSD files:

```
RewriteRule ^/fluxml$   /srv/www/htdocs/schema/fluxml.xsd [L]
RewriteRule ^/fwdsim$   /srv/www/htdocs/schema/fwdsim.xsd [L]
```

- Information about Omix and the Omix library documentation is redirected to the Omix web page `http://omix-visualization.com`.
- The access to the URL `http://www.13cflux.net/x3cfluxlicense` is passed to a Perl CGI script[2] which authenticates the usage of 13CFLUX2.
- To improve security, the access to the portal login sites is always redirected to the corresponding HTTPS URL.
- Web services provided by other server nodes are passed through, e.g.,

```
RewriteRule ^/FluxWS/Flux(.*)$ \
  http://ibt-v708:8080/FluxWS/Flux$1 [P,L]
```

  In this example, *ibt-v708* is an internal host (i.e., the host name is inaccessible by an Internet client) that provides the FluxWS web service via the URL prefix `www.13cflux.net/FluxWS/Flux`.

[1] Details on the Apache HTTPD rewrite engine *mod_rewrite* and its configuration can be found here: `https://httpd.apache.org/docs/2.2/mod/mod_rewrite.html`; last accessed: May 22, 2017

[2] The CGI license script is developed by Michael Weitzel

- Standard HTTP error pages and static sites are rewritten to static pages to provide a consistent error reporting to the user (e.g., when the JBoss server fails).
- Accesses to the URL `www.13cflux.net/` is passed to the JBoss server using the AJP protocol with the following rule:

```
RewriteRule ^/*(.*)$    ajp://localhost:8009/$1 [P,L]
```

  If none of the above substitution patterns match, the application server will handle the web request. Hence, this pattern is placed at the end of the rewrite engine rules.

Appendix E.

SWF Services and Components

Since the SWF follows the SOA paradigm, functional components of the ^{13}C-MFA procedure (i.e., tasks or sub-workflows) are wrapped and exposed as web services. In addition, several tools and libraries are part of the framework that support the realization of ^{13}C-MFA workflow applications. Table E.1 lists the most important ^{13}C-MFA tools and services categorized by type. From these components, as well as external tools and services, analysis workflows are composed.

Name	Type	Comment	Artifacts
13CFLUX2	Tool Library (C++)	Collection of Linux command-line tools and libraries to create custom ^{13}C-MFA applications	20 programs 168 headers 14 libraries
FluxCore	Library (Java)	13CFLUX2 wrapper library; extensible to other applications	3 classes
FluxWS	Service (SOAP)	Collection of SOAP web services for 13CFLUX2 (one service per 13CFLUX2 program); built on top of FluxCore; supports asynchronous operation	3 classes
FluxHadoop	Tool	Hadoop MapReduce implementation of the MCB algorithm; built on top of FluxCore	4 classes
FluxHadoop2	Tool	Specialized version of FluxHadoop for clusters with multi-core nodes (Dalman, Weitzel, Freisleben, et al., 2011)	4 classes
Hadoop Utilities	Tool	Tools and scripts to prepare, deploy, and use Hadoop MapReduce in the context of the SWF	3 programs
CAAS	Library (Java) Service (SOAP)	Library and SOAP service for web portal authentication and authorization	59 classes
MeasurementDB	Library (Java) Service (SOAP)	Library and SOAP service for accessing the measurement database	11 classes
ModelDB	Library (Java) Service (SOAP)	Library and SOAP service for accessing the model database	27 classes
ReactionDB	Library (Java) Service (SOAP)	Library and SOAP service for accessing the measurement database	23 classes
FluxProvenance	Tool Library (C++)	Provenance framework consisting of server, client, and library	2 programs 1 library 3 headers
LogCollector	Library (Java) Service (SOAP)	Java and web service interfaces for the provenance collection module; built on top of FluxProvenance	5 classes
RepoManager	Service (REST)	High-level services to manage project repositories	2 classes

Table E.1.: Collection of services to compose ^{13}C-MFA applications.

List of Symbols and Abbreviations

List of Symbols

Performance and Parallel Efficiency

T_p . . . . . Program runtime on p processors or cores with $p \in \mathbb{N}^+$
$S(p)$ . . . . . Speedup: T_1/T_p
$PE(p)$ . . . . . Parallel efficiency: $S/p = T_1/pT_p$

List of Abbreviations

API . . . . . Application Programming Interface
AWS . . . . . Amazon Web Services
BPEL . . . . . Business Process Execution Language
CPU . . . . . Central Processing Unit
DBMS . . . . . Database Management System
DTW . . . . . Dynamic Time Warp
EMR . . . . . (Amazon) Elastic MapReduce
FZJ . . . . . Forschungszentrum Jülich
GUI . . . . . Graphical User Interface
HDF5 . . . . . Hierarchical Data Format (ver. 5)
HDFS . . . . . Hadoop Distributed Filesystem
HTTP . . . . . HyperText Transport Protocol
HTTPS . . . . . HyperText Transport Protocol Secure
IBG . . . . . Institute of Bio- and Geosciences
I/O . . . . . Input/Output
IT . . . . . Information Technology
JPA . . . . . Java Persistence API
JSP . . . . . JavaServer Pages
KEGG . . . . . Kyoto Encyclopedia of Genes and Genomes
MCB . . . . . Monte Carlo Bootstrap
MFA . . . . . Metabolic Flux Analysis
MS . . . . . Mass Spectrometry
MPI . . . . . Message Passing Interface
POSIX . . . . . Portable Operating System Interface
REST . . . . . REpresentational State Transfer

SCTP . . . . . Stream Control Transmission Protocol
SOA . . . . . Service-Oriented Architecture
SWF . . . . . Scientific Workflow Framework
UDP . . . . . User Datagram Protocol
UML . . . . . Unified Modeling Language
URL . . . . . Uniform Resource Locator
TCP . . . . . Transmission Control Protocol
VCS . . . . . Version Control System
VM . . . . . Virtual Machine
WSDL . . . . . Web Service Description Language
WSRF . . . . . Web Services Resource Framework
XML . . . . . Extensible Markup Language
XSD . . . . . XML Schema Definition

List of Figures

1.1. Classical metabolic engineering experimental cycle 3
1.2. Overview of the complete ^{13}C-MFA workflow 4

2.1. Screenshot of the Bio-jETI client 16

3.1. Schematic overview of the ^{13}C-MFA method 20
3.2. Screenshot of the Omix user interface 26

4.1. Design of the SWF for ^{13}C-MFA 33
4.2. UML diagram of the JuMeDaS measurement database schema 35
4.3. UML diagram of the JuMetReD model database schema 36
4.4. Layers of the 13CFLUX2 legacy wrapper architecture 41
4.5. Provenance life-cycle model 45
4.6. Design overview of the log collector 46
4.7. Screenshot of the `www.13cflux.net` front web page 50
4.8. UML diagram of the CAAS database schema 52
4.9. Diagram of repository integration classes using SVNKit 53
4.10. Screenshot of the SVNKit web site 54
4.11. Overview of the cluster infrastructure at FZJ/IBG-1 57

5.1. The ^{13}C-MFA workflow – revisited 60
5.2. UML class diagram of the *FluxCore* library 62
5.3. *FluxHadoop* architecture overview 68
5.4. Embedding the Hadoop MCB algorithm into a BPEL workflow 71
5.5. UML diagram of the fluxlog class library 73
5.6. Simplified UML diagram of the provenance store fluxprov 74
5.7. UML class diagram of *multifwdsim*, *multifitfluxes*, and *multiperturb* 82

6.1. A typical ^{13}C-MFA modeling workflow 87
6.2. Boxplot of the BCG result data comparing flux distributions 89
6.3. Explorative ^{13}C-MFA simulation data (*C. glutamicum*) 91
6.4. Overview of the sampling-oriented flux space exploration workflow 92
6.5. Benchmark results on Amazon instances 99
6.6. Results of a microbenchmark comparing *FluxHadoop* and *FluxHadoop2* . 102
6.7. Runtime comparison of *FluxHadoop* and *FluxHadoop2* 103
6.8. Residual values collected from a *fitfluxes* run 105
6.9. Overview of the interactive ^{13}C-MFA workflow 106

6.10. Visualizations generated in the course of a ^{13}-C MFA workflow 107
6.11. Identified clusters of the *C. glutamicum* model 109

List of Tables

2.1. Comparison between various existing scientific workflow approaches . . . 13

3.1. Overview of network model characteristics 23
3.2. Benchmarks with *fwdsim* and *fitfluxes* 24

4.1. Comparison table of popular VCSs 37
4.2. Comparison of four Subversion VCS clients 38
4.3. Comparison of ActiveBPEL and Python as workflow engine 40
4.4. Comparison table of employed network transport protocols 47

5.1. Comparison of the MCB realizations with Hadoop and 13CFLUX2 72
5.2. Comparison between local file and networking protocols 77

6.1. Computation times on local machine (single core usage) 95
6.2. Single core benchmark results on a local node 96
6.3. Computation times on different Amazon instances 97
6.4. Custom Hadoop parameter settings 101

E.1. Collection of ^{13}C-MFA services 138

Bibliography

Aalst, W. M. P. van der and A. H. M. ter Hofstede (2005). "YAWL: yet another workflow language". In: *Inf. Syst.* 30 (4), pp. 245–275.

Aalst, W. van der and C. Stahl (2011). *Modeling business processes: a Petri net-oriented approach*. Cooperative information systems. MIT Press.

Aalst, W. M. P. van der and K. M. van Hee (2002). *Workflow management: models, methods, and systems*. MIT Press.

Akram, A., D. Meredith, and R. Allan (2006). "Evaluation of BPEL to scientific workflows". In: *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 06)*. Vol. 1. IEEE, pp. 269–274.

Altintas, I., O. Barney, and E. Jaeger-Frank (2006). "Provenance collection support in the Kepler scientific workflow system". In: *Provenance and Annotation of Data: International Provenance and Annotation Workshop (IPAW 2006), Revised Selected Papers*. Vol. 4145. Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin, Heidelberg, pp. 118–132.

Altschul, S., W. Gish, W. Miller, E. Myers, and D. Lipman (1990). "Basic local alignment search tool". In: *Journal of Molecular Biology* 215, pp. 403–410.

Anand, M. K., S. Bowers, and B. Ludäscher (2010). "Techniques for efficiently querying scientific workflow provenance graphs". In: *Proceedings of the 13th International Conference on Extending Database Technology (EDBT '10)*. ACM, pp. 287–298.

Andrews, T., F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana (2003). *Business process execution language for web services version 1.1*. URL: `https://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf`.

Antoniewicz, M. R., J. K. Kelleher, and G. N. Stephanopoulos (2006). "Determination of confidence intervals of metabolic fluxes estimated from stable isotope measurements". In: *Metabolic Engineering*, pp. 324–337.

– (2007). "Elementary metabolite units (EMU): a novel framework for modeling isotopic distributions". In: *Metabolic Engineering* 9 (1), pp. 68–86.

Atkinson, A. C. and A. N. Donev (1992). *Optimum experimental designs*. Vol. 8. Oxford Statistical Science Series. Clarendon Press, Oxford.

Banks, T. (2006). *Web services resource framework (WSRF) – primer v1.2*. URL: `http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf`.

Barker, A. and J. V. Hemert (2008). "Scientific workflow: a survey and research directions". In: *Proceedings of the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM 2007)*. Vol. 4967. Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin, Heidelberg, pp. 746–753.

Barseghian, D., I. Altintas, M. B. Jones, D. Crawl, N. Potter, J. Gallagher, P. Cornillon, M. Schildhauer, E. T. Borer, E. W. Seabloom, and P. R. Hosseini (2010). "Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis". In: *Ecological Informatics* 5 (1), pp. 42–50.

Bellman, R. (1957). *Dynamic programming*. Princeton University Press.

Bente, D. A., J. Friesen, K. White, J. Koll, and G. P. Kobinger (2011). "A computerized data-capture system for animal biosafety level 4 laboratories". In: *Journal of the American Association for Laboratory Animal Science (JAALAS)* 50 (5), pp. 660–664.

Berthold, M. R., N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel (2009). "KNIME - the Konstanz information miner: version 2.0 and beyond". In: *SIGKDD Explor. Newsl.* 11 (1), pp. 26–31.

Beste, D., K. Nöh, S. Niedenführ, T. Mendum, N. Hawkins, J. Ward, M. Beale, W. Wiechert, and J. McFadden (2013). "^{13}C-flux spectral analysis of host-pathogen metabolism reveals a mixed diet for intracellular Mycobacterium tuberculosis". In: *Chemistry & biology* 20, pp. 1012–1021.

Beste, D., B. Bonde, N. Hawkins, J. Ward, M. Beale, S. Noack, K. Nöh, N. Kruger, R. Ratcliffe, and J. McFadden (2011). "^{13}C metabolic flux analysis identifies an unusual route for pyruvate dissimilation in mycobacteria which requires isocitrate lyase and carbon dioxide fixation". In: *PLoS PATHOGENS* 7, e1002091.

Bissyandé, T. F., F. Thung, D. Lo, L. Jiang, and L. Réveillère (2013). "Popularity, interoperability, and impact of programming languages in 100,000 open source projects". In: *37th Annual International Computer Software & Applications Conference (COMPSAC 2013), Kyoto*. IEEE, pp. 303–312.

Bowen, R. and K. Coar (2008). *Apache cookbook*. 2nd. O'Reilly & Associates, Inc.

Bowers, S. and B. Ludäscher (2005). "Actor-oriented design of scientific workflows". In: *Conceptual Modeling ER 2005*. Vol. 3716. Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin, Heidelberg, pp. 369–384.

Buyya, R., J. Broberg, and A. Gościński (2011). *Cloud computing: principles and paradigms*. Wiley Series on Parallel and Distributed Computing. John Wiley & Sons.

Cao, B., B. Plale, G. Subramanian, E. Robertson, and Y. Simmhan (2009). "Provenance Information Model of Karma Version 3". In: *Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA: IEEE Computer Society, pp. 348–351. DOI: 10.1109/SERVICES-I.2009.54.

Chernick, M. R. (2007). *Bootstrap methods: a guide for practitioners and researchers*. 2nd ed. Wiley-Interscience.

Chopra, V., S. Li, and J. Genender (2007). *Professional Apache Tomcat 6*. Wrox Professional Guides. Wiley.

Churches, D., G. Gombás, A. Harrison, J. Maassen, C. R. M. Shields, I. Taylor, and I. Wang (2006). "Programming scientific and distributed workflow with Triana services". In: *Concurrency and Computation: Practice and Experience* 18 (10), pp. 1021–1037.

Cieslik, M. and C. Mura (2011). "A lightweight, flow-based toolkit for parallel and distributed bioinformatics piplines". In: *BMC Bioinformatics* 12 (1), pp. 61+.

Cock, P. J., T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. de Hoon (2009). "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics (Oxford, England)* 25 (11), pp. 1422–1423.

Collette, A. (2013). *Python and HDF5*. O'Reilly Media.

Collins-Sussman, B., B. W. Fitzpatrick, and C. M. Pilato (2008). *Version control with Subversion*. 2nd. O'Reilly Media.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms*. 3rd. The MIT Press.

Cortassa, S., M. A. Aon, A. A. Iglesias, J. C. Aon, and D. Lloyd (2012). *An introduction to metabolic and cellular engineering*. 2nd ed. World Scientific Publishing Company, Incorporated.

Crown, S. B. and M. R. Antoniewicz (2013). "Publishing 13C metabolic flux analysis studies: a review and future perspectives". In: *Metab Eng.* 20, pp. 42–48.

Curbera, F., Y. N. Doganata, A. Martens, N. Mukhi, and A. Slominski (2008). "Business Provenance - A Technology to Increase Traceability of End-to-End Operations." In: *OTM Conferences (1)'08*, pp. 100–119.

Curcin, V. and M. Ghanem (2008). "Scientific workflow systems - can one size fit all?" In: *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International*, pp. 1–9.

Dalman, T., T. Dörnemann, E. Juhnke, M. Weitzel, K. Nöh, W. Wiechert, and B. Freisleben (2010). "Metabolic flux analysis in the cloud". In: *Proceedings of IEEE 6th International Conference on e-Science*, pp. 57–64.

– (2013). "Cloud MapReduce for Monte Carlo bootstrap applied to metabolic flux analysis". In: *Future Generation Computer Science* 29 (2), pp. 582–590.

Dalman, T., P. Droste, M. Weitzel, W. Wiechert, and K. Nöh (2010). "Workflows for metabolic flux analysis: data integration and human interaction". In: *Proceedings of the 4th International Conference on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA)*. Vol. 6415. Lecture Notes in Computer Science (LNCS). Springer-Verlag Berlin, Heidelberg, pp. 261–275.

Dalman, T., E. Juhnke, T. Dörnemann, M. Weitzel, K. Nöh, W. Wiechert, and B. Freisleben (2010). "Service workflows and distributed computing methods for ^{13}C metabolic flux analysis". In: *Proceedings of 7th EUROSIM Congress on Modelling and Simulation*, pp. 1–7.

Dalman, T., M. Weitzel, B. Freisleben, W. Wiechert, and K. Nöh (2011). "A hybrid parallelization approach for cloud-enabled metabolic flux analysis simulation workflows". In: *Proceedings of 4th GRID4TS Workshop*, pp. 30–31.

Dalman, T., M. Weitzel, W. Wiechert, B. Freisleben, and K. Nöh (2011). "An online provenance service for workflows for distributed metabolic flux analysis". In: *Proceedings of IEEE 9th European Conference on Web Services (ECOWS)*. IEEE Press, pp. 91–98.

Davidson, S. B. and J. Freire (2008). "Provenance and scientific workflows: challenges and opportunities". In: *Proceedings of ACM SIGMOD*, pp. 1345–1350.

Dean, J. and S. Ghemawat (2004). "MapReduce: simplified data processing on large clusters". In: *OSDI'04: Proceedings of the 6th conference on Symposium on Operating Sys-*

tems Design & Implementation. San Francisco, CA: USENIX Association, pp. 107–113.

Deelman, E., D. Gannon, M. Shields, and I. Taylor (2009). "Workflows and e-science: an overview of workflow system features and capabilities". In: *Future Generation Computer Systems (FCGS)* 25 (5), pp. 528–540.

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

Dörnemann, T. (2013). "Supporting quality of service in scientific workflows". PhD thesis. Marburg, University.

Dörnemann, T., E. Juhnke, and B. Freisleben (2009). "On-demand resource provisioning for BPEL workflows using Amazon's elastic compute cloud". In: *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '09)*. IEEE Press, pp. 140–147.

Dörnemann, T., M. Smith, and B. Freisleben (2008). "Composition and execution of secure workflows in WSRF-grids". In: *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '08)*. IEEE Press, pp. 122–129.

Droste, P. (2011). "Customizable visualization in the context of metabolic networks". PhD thesis. Siegen, University.

Droste, P., E. von Lieres, W. Wiechert, and K. Nöh (2010). "Customizable visualization on demand for hierarchically organized information in biochemical networks". In: *Computational Modeling of Objects Represented in Images*. Ed. by R. P. Barneva, V. E. Brimkov, H. A. Hauptman, R. M. N. Jorge, and J. M. R. S. Tavares. Vol. 6026. Lecture Notes in Computer Science. Springer-Verlag Berlin, Heidelberg, pp. 163–174.

Ebert, B. E., A.-L. Lamprecht, B. Steffen, and L. M. Blank (2012). "Flux-P: automating metabolic flux analysis". In: *Metabolites* 2 (4), pp. 872–890.

Edelhofer, T. (2014). *Selecting a MATLAB application deployment strategy*. URL: `https://www.mathworks.com/company/newsletters/articles/selecting-a-matlab-application-deployment-strategy.html`.

Edlich, S., A. Friedland, J. Hampe, and B. Brauer (2010). *NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken (in german)*. Hanser.

Efron, B. and R. J. Tibshirani (1993). *An introduction to the bootstrap*. Chapman & Hall/CRC.

Erl, T. (2004). *Service-Oriented Architecture: a field guide to integrating XML and web services*. Prentice Hall PTR.

– (2014). *Next generation SOA: a concise introduction to service technology & service-orientation*. Prentice Hall.

Eronen, L. and H. Toivonen (2012). "Biomine: predicting links between biological entities using network models of heterogeneous databases". In: *BMC Bioinformatics* 13 (1), pp. 1–21.

Evans, B. (2014). *What every Java developer needs to know about Java 9*. O'Reilly Radar (online); http://radar.oreilly.com/2014/09/what-every-java-developer-needs-to-know-about-java-9.html.

Ferguson, N., B. Schneier, and T. Kohno (2010). *Cryptography engineering - design principles and practical applications*. Wiley.

Folk, M., A. Cheng, and K. Yates (1999). "HDF5: A file format and I/O library for high performance computing applications". In: *Proceedings of Supercomputing*. Vol. 99.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing, Inc.

– (2003). *UML distilled: a brief guide to the standard object modeling language*. 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Fuhrmann, S. (2010). *Metabolische Datenbanken in der Stoffflussanalyse: Entwurf und Implementierung (in german)*. VDM Publishing.

Gannon, D., E. Deelman, M. Shields, and I. Taylor (2006). "Introduction". In: *Workflows for e-Science: scientific workflows for grids*. Springer-Verlag New York, Inc. Chap. 1.

Giorgino, T. (2009). "Computing and visualizing dynamic time warping alignments in R: the dtw package". In: *Journal of Statistical Software* 31 (7), pp. 1–24.

Goecks, J., A. Nekrutenko, J. Taylor, and T. G. Team (2010). "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences". In: *Genome Biology* 11 (8), R86.

Goyvaerts, J. and S. Levithan (2012). *Regular expressions cookbook*. O'Reilly Media.

Gross, J. H. (2011). *Mass spectrometry – a textbook*. 2nd ed. Springer-Verlag Berlin Heidelberg.

Hart, W. E. (2011). *Managing scientific workflows in Python with pyutilib.workflow*. URL: `https://software.sandia.gov/trac/pyutilib/export/2215/pyutilib.workflow/trunk/doc/workflow/workflow.pdf`.

Haugwitz, M. von (2016). "Mass spectrometric data processing for metabolomics and fluxomics – a flexible evaluation framework with quality awareness". RWTH Aachen, Diss., 2015. Dr. RWTH Aachen, 291 p.

Hick, J. and J. Shalf (2009). "Storage Technology". In: *Scientific data management*. Ed. by A. Shoshani and D. Rotem. Computational Science Series. Chapman & Hall. Chap. 1.

Hintjens, P. (2013). *ZeroMQ: messaging for many applications*. O'Reilly Media.

Hohmann, L. (2003). *Beyond software architecture: creating and sustaining winnig solutions*. 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Hohpe, G. and B. Woolf (2003). *Enterprise integration patterns: designing, building, and deploying messaging solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Hope, J. (2008). *Biobazaar: the open source revolution and biotechnology*. Harvard University Press.

Hucka, M., A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. L. Novère,

L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang (2003). "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models". In: *Bioinformatics* 9 (4), pp. 524–531.

Hudson, G. (2002). *Notes on keeping version histories of files*. `http://web.mit.edu/ghudson/thoughts/file-versioning`; last accessed: 2016-02-07.

Hull, D., K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn (2006). "Taverna: a tool for building and running workflows of services". In: *Nucleic Acids Research* 34, pp. 729–732.

Ibrahim, S., H. Jin, L. Lu, S. Wu, B. He, and L. Qi (2010). "LEEN: locality/fairness-aware key partitioning for MapReduce in the cloud". In: *CloudCom*. IEEE, pp. 17–24.

Jamae, J. and P. Johnson (2009). *JBoss in action: configuring the JBoss application server*. Manning Publications Co.

Jiang, D., B. C. Ooi, L. Shi, and S. Wu (2010). "The performance of MapReduce: an in-depth study". In: *Proc. VLDB Endow.* 3 (1-2), pp. 472–483.

Jin, H., S. Ibrahim, L. Qu, H. Cao, S. Wu, and X. Shi (2011). "The MapReduce programming model and implementations". In: *Cloud computing: principles and paradigms*. John Wiley & Sons. Chap. 14.

Johnson, R. E. and B. Foote (1988). "Designing reusable classes". In: *Object-Oriented Programming* 1 (2).

Joshi, M., A. Seidel-Morgenstern, and A. Kremling (2006). "Exploiting the bootstrap method for quantifying parameter confidence intervals in dynamical systems". In: *Metabolic Engineering* 8 (5), pp. 447–455.

Joshi, S. B. (2012). "Apache Hadoop performance-tuning methodologies and best practices". In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ICPE '12. New York, NY, USA: ACM, pp. 241–242.

Josuttis, N. M. (2007). *SOA in practice: the art of distributed system design*. O'Reilly Media.

Juhnke, E., D. Seiler, T. Stadelmann, T. Dörnemann, and B. Freisleben (2009). "LCDL: an extensible framework for wrapping legacy code". In: *Proceedings of ERPAS'2009*, pp. 646–650.

Kanehisa, M. and S. Goto (2000). "KEGG: Kyoto encyclopedia of genes and genomes". In: *Nucleic acids research* 28 (1), pp. 27–30.

Kitano, H. (2002). "Systems Biology: brief overview". In: *Science* 295 (5560), pp. 1662–1664.

Krishnan, S., L. Clementi, J. Ren, P. Papadopoulos, and W. Li (2009). "Design and evaluation of Opal2: a toolkit for scientific software as a service". In: *IEEE Congress on Services*.

Kruchten, P. (1995). "Architectural blueprints: the 4+1 view model of architecture". In: *IEEE Software* 12 (6), pp. 42–50.

Kumar, V., A. Grama, A. Gupta, and G. Karypis (2003). *Introduction to parallel computing*. 2nd ed. Addison-Wesley.

Lamprecht, A.-L. (2013). *User-level workflow design - a bioinformatics perspective*. Vol. 8311. Lecture Notes in Computer Science (LNCS). Springer.

Lamprecht, A.-L., T. Margaria, and B. Steffen (2009). "Bio-jETI: a framework for semantics-based service composition". In: *BMC Bioinformatics* 10 Suppl 10, S8.

Larkin, M., G. Blackshields, N. Brown, R. Chenna, P. McGettigan, H. McWilliam, F. Valentin, I. Wallace, A. Wilm, R. Lopez, J. Thompson, T. Gibson, and D. Higgins (2007). "Clustal W and Clustal X version 2.0". In: *Bioinformatics* 23 (21), pp. 2947–2948.

Love, R. (2013). *Linux system programming: talking directly to the kernel and C library*. 2nd ed. O'Reilly Media.

Ludäscher, B., I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao (2006). "Scientific workflow management and the Kepler system". In: *Concurrency and Computation: Practice and Experience* 18 (10), pp. 1039–1065.

Martín-Requena, V., J. Ríos, M. García, S. Ramírez, and O. Trelles (2010). "jORCA: easily integrating bioinformatics web services". In: *Bioinformatics* 26 (4), pp. 553–559.

Matsunaga, A., M. Tsugawa, and J. Fortes (2008). "CloudBLAST: combining MapReduce and virtualization on distributed resources for bioinformatics applications". In: *ESCIENCE '08: Proceedings of the 2008 4th IEEE International Conference on eScience*. Washington, DC, USA: IEEE Computer Society, pp. 222–229.

Matsuoka, Y., S. Ghosh, and H. Kitano (2009). "Consistent design schematics for biological systems: standardization of representation in biological engineering". In: *J. R. Soc. Interface* 6 Suppl. 4, pp. 393–404.

Mell, P. and T. Grance (2011). *The NIST definition of cloud computing (NIST special publication 800-145)*. National Institue of Standards and Technology, Computer Security Division.

Miebach, S. (2012). "Charakterisierung und Validierung der ^{13}C-Stoffflussanalyse im Parallelansatz (in german)". PhD thesis. Bielefeld, University.

Milstein, S., J. Biersdorfer, and M. MacDonald (2006). *Google: the missing manual*. 2nd. Missing manual. Sebastopol, CA, USA: O'Reilly & Associates, Inc.

Miner, D. and A. Shook (2012). *MapReduce design patterns*. O'Reilly Media.

Missier, P., N. Paton, and K. Belhajjame (2010). "Fine-grained and efficient lineage querying of collection-based workflow provenance". In: *Procs. EDBT*. Lausanne, Switzerland.

Möllney, M., W. Wiechert, D. Kownatzki, and A. A. de Graaf (1999). "Bidirectional reaction steps in metabolic networks: IV. optimal design of isotopomer labeling experiments". In: *Biotechnology and Bioengineering* 66 (2), pp. 86–103.

Moreau, L., B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. V. den Bussche (2011). "The Open Provenance Model core specification (v1.1)". In: *Future Generation Computer Systems* 27 (6), pp. 743–756.

Moreau, L., P. Groth, S. Miles, J. Vazquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, A. Schreiber, V. Tan, and L. Varga (2008). "The provenance of electronic data". In: *Commun. ACM* 51 (4), pp. 52–58.

Mulder, S. and Z. Yaar (2006). *The user is always right: a practical guide to creating and using personas for the web*. First. Thousand Oaks, CA, USA: New Riders Publishing.

Neuweger, H., S. P. Albaum, M. Dondrup, M. Persicke, T. Watt, K. Niehaus, J. Stoye, and A. Goesmann (2008). "MeltDB: a software platform for the analysis and integration of metabolomics experiment data". In: *Bioinformatics* 24 (23), pp. 2726–2732.

Newman, S. (2015). *Building microservices – designing fine-grained systems*. O'Reilly Media.

Niedenführ, S. (2014). "Analyzing the fluxome of P. chrysogenum in an industrial environment - workflows for 13C metabolic flux analysis in complex systems". PhD thesis. RWTH Aachen, 293 p.

Niedenführ, S., W. Wiechert, and K. Nöh (2015). "How to measure metabolic flux: a taxonomic guide for ^{13}C fluxomics". In: *Current opinion in biotechnology* 34, pp. 82–90.

Nöh, K., P. Droste, and W. Wiechert (2015). "Visual workflows for ^{13}C-metabolic flux analysis". In: *Bioinformatics* 31 (3).

Novère, N. L., A. Finney, M. Hucka, U. S. Bhalla, F. Campagne, J. Collado-Vides, E. J. Crampin, M. Halstead, E. Klipp, P. Mendes, P. Nielsen, H. Sauro, B. Shapiro, J. L. Snoep, H. D. Spence, and B. L. Wanner (2005). "Minimum information requested in the annotation of biochemical models (MIRIAM)". In: *Nature Biotechnology* 23 (12), pp. 1509–1515.

Obe, R. O. and L. S. Hsu (2014). *Postgresql: up and running: a practical introduction to the advanced open source database*. O'Reilly Media.

Oinn, T., P. Li, D. B. Kell, C. Goble, A. Goderis, M. Greenwood, D. Hull, R. Stevens, D. Turi, and J. Zhao (2006). "Taverna/myGrid: aligning a workflow system with the life sciences community". In: *Workflows for e-Science: scientific workflows for grids*. Springer-Verlag New York, Inc. Chap. 19.

Ortel, J., J. Noehr, and N. V. Gheem (last accessed: Mar 2016). *SUDS library web page*. https://fedorahosted.org/suds/.

Pacheco, P. S. (2011). *An introduction to parallel programming*. Morgan Kaufman.

Palankar, M. R., A. Iamnitchi, M. Ripeanu, and S. Garfinkel (2008). "Amazon S3 for science grids: a viable solution?" In: *DADC '08: Proceedings of the 2008 International Workshop on Data-aware Distributed Computing*. Boston, MA, USA: ACM, pp. 55–64.

Palsson, B. (2011). *Systems Biology: simulation of dynamic network states*. Cambridge University Press.

Papoulis, A. and S. U. Pillai (2002). *Probability, Random Variables, and Stochastic Processes*. 4th ed. McGraw-Hill Higher Education.

Pelleg, D. and A. W. Moore (2000). "X-means: extending K-means with efficient estimation of the number of clusters". In: *ICML*. Ed. by P. Langley. Morgan Kaufmann, pp. 727–734.

Petersen, S. (2001). "Investigating the in vivo activity of anaplerotic pathways in Corynebacterium glutamicum with ^{13}C-tracer technology (in german)". Institut für Biotechnologie, Berichte des Forschungszentrums Jülich, 3875. PhD thesis. University of Düsseldorf.

Peterson, J. L. and A. Silberschatz (1985). *Operating system concepts*. 2nd. Addison-Wesley.

Pitkänen, E., A. Åkerlund, A. Rantanen, P. Jouhten, and E. Ukkonen (2008). "ReMatch: a web-based tool to construct, store and share stoichiometric metabolic models with carbon maps for metabolic flux analysis". In: *J. Integrative Bioinformatics* 5 (2).

Pratx, G. and L. Xing (2011). "Monte Carlo simulation of photon migration in a cloud computing environment with MapReduce". In: *J. Biomed. Opt.* 16 (12).

Quek, L. E., C. Wittmann, L. Nielsen, and J. Kromer (2009). "OpenFLUX: efficient modelling software for 13C-based metabolic flux analysis". In: *Microbial Cell Factories* 8 (1), pp. 25+.

Raue, A., C. Kreutz, T. Maiwald, U. Klingmüller, and J. Timmer (2011). "Addressing parameter identifiability by model-based experimentation". In: *IET Systems Biology* 5 (2), pp. 120–130.

Ravikirthi, P., P. F. Suthers, and C. D. Maranas (2011). "Construction of an E. coli genome-scale atom mapping model for MFA calculations". In: *Biotechnology and Bioengineering* 108 (6), pp. 1372–1382.

Raymond, E. S. (1999). *The cathedral & the bazaar*. 1st ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc.

Romano, P. (2008). "Automation of in-silico data analysis processes through workflow management systems". In: *Briefings in Bioinformatics* 9 (1), pp. 57–68.

Rubinger, A. L. and B. Burke (2010). *Enterprise JavaBeans 3.1*. 6th. O'Reilly Media.

Runkel, J. (2009). "Entwurf und Implementierung einer Datenbank- und Sicherheits-Middleware für ein Scientific Workflow System (in german)". Diploma Thesis. Univerität Siegen, Fachbereich Elektrotechnik & Informatik.

Sammer, E. (2012). *Hadoop operations*. O'Reilly Media.

Schuster, S., T. Dandekar, and D. Fell (1999). "Detection of elementary flux modes in biochemical networks: a promising tool for pathway analysis and metabolic engineering". In: *Trends Biotechnology* 17 (2), pp. 53–60.

Schwender, J. (2011). "Experimental flux measurements on a network scale". In: *Front in Plant Science* 2 (63).

Scott, S. L., A. W. Blocker, and F. V. Bonassi (2016). "Bayes and big data: the consensus Monte Carlo algorithm". In: *International Journal of Management Science and Engineering Management (to appear)*.

Sears, R., C. V. Ingen, and J. Gray (2006). *To BLOB or not to BLOB: large object storage in a database or a filesystem*. Tech. rep. MSR-TR-2006-45. Microsoft Research, p. 10.

Sehgal, S., M. Erdelyi, A. Merzky, and S. Jha (2011). "Understanding application-level interoperability: scaling-out MapReduce over high-performance grids and clouds". In: *Future Generation Computer Systems* 27 (5), pp. 590–599.

Senger, M., P. Rice, A. Bleasby, T. Oinn, and M. Uludag (2008). "Soaplab2: more reliable Sesame door to bioinformatics programs". In: *9th annual Bioinformatics Open Source Conference*.

Shoshani, A. and D. Rotem (2009). *Scientific data management: challenges, technology, and deployment*. Chapman & Hall/CRC.

Sneed, H. M. (2006). "Integrating legacy Software into a service oriented architecture". In: *10th European Conference on Software Maintenance and Reengineering (CSMR 2006), 22-24 March 2006, Bari, Italy*. IEEE Computer Society, pp. 3–14.

Spector, D. H. M. (2000). *Building Linux clusters*. O'Reilly & Associates, Inc.

Stephanopoulos, G. N. (1999). "Metabolic Fluxes and Metabolic Engineering". In: *Metabolic Engineering* 1 (1), pp. 1–11.

Stephanopoulos, G. N., A. A. Aristidou, and J. Nielsen (1998). *Metabolic engineering: principles and methodologies*. Academic Press.

Stevens, W. R., B. Fenner, and A. M. Rudoff (2003). *UNIX network programming, vol. 1*. 3rd ed. Pearson Education.

Stewart, J. (2014). *Python for Scientists*. Cambridge University Press.

Stewart, R. (2007). *Stream Control Transmission Protocol*. RFC 4960 (Proposed Standard). Internet Engineering Task Force.

Synesis Software Pty Ltd (2010). *Pantheios performance*. http://www.pantheios.org/performance.html.

Tan, W. and M. Zhou (2013). *Business and scientific workflows: a web service-oriented approach*. IEEE Press Series on Systems Science and Engineering. Wiley.

Tan, W., P. Missier, I. Foster, R. Madduri, D. De Roure, and C. Goble (2010). "A comparison of using Taverna and BPEL in building scientific workflows: the case of caGrid". In: *Concurrency and Computation: Practice and Experience* 22 (9), pp. 1098–1117.

Tanenbaum, A. (2002). *Computer networks*. 4th. Prentice Hall Professional Technical Reference.

Taylor, C. F., N. W. Paton, K. S. Lilley, P.-A. Binz, J. Randall K. Julian, A. R. Jones, W. Zhu, R. Apweiler, R. Aebersold, E. W. Deutsch, M. J. Dunn, A. J. R. Heck, A. Leitner, M. Macht, M. Mann, L. Martens, T. A. Neubert, S. D. Patterson, P. Ping, S. L. Seymour, P. Souda, A. Tsugita, J. Vandekerckhove, T. M. Vondriska, J. P. Whitelegge, M. R. Wilkins, I. Xenarios, I. John R. Yates, and H. Hermjakob (2007). "The minimum information about a proteomics experiment (MIAPE)". In: *Nature Biotechnology* 25, pp. 887–893.

Taylor, I. J., E. Deelman, D. B. Gannon, and M. Shields (2006). *Workflows for e-Science: scientific workflows for grids*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Tichy, W. F. (1982). "Design, implementation, and evaluation of a revision control system". In: *Proceedings of the 6th International Conference on Software Engineering*. ICSE '82. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 58–67.

Todorov, D. (2007). *Mechanics of user identification and authentication: fundamentals of identity management*. Auerbach Publications. Taylor & Francis Group.

Wall, D. P., P. Kudtarkar, V. A. Fusaro, R. Pivovarov, P. Patil, and P. J. Tonellato (2010). "Cloud computing for comparative genomics". In: *BMC Bioinformatics* 11 (1), pp. 259–270.

Waltemath, D., R. Adams, D. A. Beard, F. T. Bergmann, U. S. Bhalla, R. Britten, V. Chelliah, M. T. Cooling, J. Cooper, E. J. Crampin, A. Garny, S. Hoops, M. Hucka, P. Hunter, E. Klipp, C. Laibe, A. K. Miller, I. Moraru, D. Nickerson, P. Nielsen, M. Nikolski, S. Sahle, H. M. Sauro, H. Schmidt, J. L. Snoep, D. Tolle, O. Wolkenhauer, and

N. L. Novère (2011). "Minimum information about a simulation experiment (MIASE)". In: *PLoS Comput Biol* 7 (4), e1001122.

Weitzel, M. (2009). "High performance algorithms for metabolic flux analysis". PhD thesis. University of Siegen, Germany.

Weitzel, M., K. Nöh, T. Dalman, S. Niedenführ, B. Stute, and W. Wiechert (2013). "13CFLUX2 – high-performance software suite for ^{13}C-metabolic flux analysis". In: *Bioinformatics* 29 (1), pp. 143–145.

White, T. (2009). *Hadoop: the definitive guide*. 1st ed. O'Reilly Media.

Wiechert, W. (1996). *Metabolische Kohlenstoff-Markierungssysteme: Modellierung, Simulation, Analyse, Datenauswertung (in german)*. Vol. 3301. Zentralbibliothek Forschungszentrum Jülich GmbH.

– (2001). "^{13}C metabolic flux analysis". In: *Metababolic Engineering* 3 (3), pp. 195–206.

Wiechert, W., M. Möllney, S. Petersen, and A. A. de Graaf (2001). "A universal framework for ^{13}C metabolic flux analysis". In: *Metabolic Engineering* 3 (3), pp. 265–283.

Yang, T. H. (2013). "^{13}C-based metabolic flux analysis: fundamentals and practice". In: *Systems Metabolic Engineering: Methods in Molecular Biology* 985, pp. 297–334.

Zaharia, M., A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica (2008). "Improving MapReduce performance in heterogeneous environments". In: *OSDI*. Ed. by R. Draves and R. van Renesse. USENIX Association, pp. 29–42.

Zamboni, N., S.-M. Fendt, M. Rühl, and U. Sauer (2009). "^{13}C-based metabolic flux analysis". In: *Nature Protocols* 4 (6), pp. 878–92.

Zamboni, N., E. Fischer, and U. Sauer (2005). "FiatFlux – a software for metabolic flux analysis from 13C-glucose experiments". In: *BMC bioinformatics* 6.