Université de Neuchâtel
Faculté des Sciences
Institut d'Informatique

Approaches for Cloudification of Complex High Performance Simulation Systems

par

Andrei Lapin

Thèse

présentée à la Faculté des Sciences
pour l'obtention du grade de Docteur ès Sciences

Acceptée sur proposition du jury:

Prof. Peter Kropf, Directeur de thèse
Université de Neuchâtel, Suisse

Prof. Pascal Felber
Université de Neuchâtel, Suisse

Prof. Jesus Carretero
University Carlos III of Madrid, Spain

Prof. Pierre Kuonen
Haute école d'ingénierie et d'architecture de Fribourg, Suisse

Prof. Marcelo Pasin
Haute école Arc, Neuchâtel, Suisse

Prof. Philip Brunner
Université de Neuchâtel, Suisse

Soutenue le 24 octobre 2017

UNIVERSITÉ DE NEUCHÂTEL

Faculté des Sciences
Secrétariat-décanat de Faculté
Rue Emile-Argand 11
2000 Neuchâtel – Suisse
Tél : + 41 (0)32 718 21 00
E-mail : secretariat.sciences@unine.ch

IMPRIMATUR POUR THESE DE DOCTORAT

La Faculté des sciences de l'Université de Neuchâtel autorise l'impression de la présente thèse soutenue par

Monsieur Andrei LAPIN

Titre:

"Approaches for Cloudification of Complex High Performance Simulation Systems"

sur le rapport des membres du jury composé comme suit:

- Prof. Peter Kropf, directeur de thèse, UniNE
- Prof. Pascal Felber, co-directeur de thèse ad interim, UniNE
- Prof. Philip Brunner, UniNE
- Prof. Pierre Kuonen, HEIA-FR, Fribourg, Suisse
- Prof. Jesus Carretero, U. Carlos III, Madrid, Espagne
- Prof. Marcelo Pasin, HE-Arc, Neuchâtel

Neuchâtel, le 17 novembre 2017 Le Doyen, Prof. R. Bshary

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Peter Kropf, for his precious guidance and continuous support throughout the course of my PhD. I gained many useful skills and discovered multiple interesting research directions thanks to him.

I would like to thank Prof. Pascal Felber for being part of my jury. I would especially like to thank him for his remarkable help and support during the last months of my stay at the university.

I would like to thank Prof. Jesus Carretero for being part of my jury. With Prof. Jesus Carretero and his research group from the Polytechnical School of University Carlos III of Madrid we had a long-lasting research collaboration, which eventually became a very important part of my scientific work.

I would like to thank Prof. Philip Brunner for being part of my jury. Prof. Philip Brunner provided a wonderful hydrological use case, which I had been using as a benchmark for my scientific discoveries. Also, I would like to thank Dr. Oliver Schilling and Dr. Wolfgang Kurtz, with whom I had been working side-by-side on the provided hydrological problem.

I would like to thank Prof. Marcelo Pasin for being part of my jury. I would especially like to thank him for his detailed feedback and his practical suggestions how to improve my work.

I would also like to thank Prof. Peirre Kuonen for being part of my jury.

My sincere thanks also go to the Department of Computer Science of the University of Neuchâtel for the excellent working environment and very friendly atmosphere. A special thought for my colleague and office mate, Dr. Eryk Schiller, for always being helpful and motivated to collaborate. Also, I would like to thank the COST Action IC1305 “Network for Sustainable Ultrascale Computing Platforms” (NESUS) for the support and wonderful research opportunities.

Finally, I would like to thank my dear wife, Nadiia, for her never ending patience, help, and support. She was always there when I needed to find the final bits of motivation to finish this challenging journey. And a special big thank you for our son Daniel!

RÉSUMÉ

Le calcul scientifique est souvent associé à un besoin de ressources toujours croissant pour réaliser des expériences, des simulations et obtenir des résultats dans un temps raisonnable. Même si une infrastructure locale peut offrir de bonnes performances, sa limite est souvent atteinte par les chercheurs. Pour subvenir à ces besoins en constante augmentation, une solution consiste à déléguer une partie de ces tâches à un environnement en nuage. Dans cette thèse, nous nous intéresserons au problème de la migration vers des environnements en nuage d'applications scientifiques basées sur le standard MPI. En particulier, nous nous concentrerons sur les simulateurs scientifiques qui implémentent la méthode itérative Monte Carlo. Pour résoudre le problème identifié, nous (a) donnerons un aperçu des domaines du calcul en nuage et du calcul à haute performance, (b) analyserons les types de problèmes actuels liés à la simulation, (c) présenterons un prototype de simulateur Monte Carlo, (d) présenterons deux méthodes de cloudification, (e) appliquerons ces méthodes au simulateur Monte Carlo, et (f) évaluerons l'application de ces méthodes à un exemple d'utilisation réelle.

Mots-clés: systèmes distribués, calcul à haute performance, calcul en nuage, déroulement scientifique, big data, architecture orienté service, Apache Spark, réseau maillé sans fil.

ABSTRACT

Scientific computing is often associated with ever-increasing need for computer resources to conduct experiments, simulations and gain outcomes in a reasonable time frame. While local infrastructures could hold substantial computing power and capabilities, researchers may still reach the limit of available resources. With continuously increasing need for higher computing power, one of the solutions could be to offload certain resource-intensive applications to a cloud environment with resources available on-demand. In this thesis, we will address the problem of migrating MPI-based scientific applications to clouds. Specifically, we will concentrate on scientific simulators, which implement the iterative Monte Carlo method.
To tackle the identified problem, we will (a) overview high performance and cloud computing domains, (b) analyze existing simulation problem types, (c) introduce an example Monte Carlo simulator, (d) present two cloudification methodologies, (e) apply the methodologies to the example simulator, and (f) evaluate the potential application of methodologies in a real case study.

Keywords: distributed systems, high performance computing (HPC), cloud computing, scientific workflows, big data (BD), service-oriented architecture (SOA), Apache Spark, wireless mesh network (WMN).

Contents

Chapter 1

Introduction

1.1 Motivation

From the 1970s, the high performance computing (HPC) market offered researchers and scientists solutions, which delivered top performance, parallelization, and scalability needed for high quality scientific experiments. Over time building a scientific application for HPC environments became more of a tradition than a necessity. Hence, today a considerable amount of scientific applications is still subject to the traditional HPC infrastructures. According to the TOP500 overview of the IDC's latest forecast 2016-2020 for the HPC Market [63], the demand for HPC continues growing, with a compounded annual growth rate of 8% from 2015 to 2019. However, there is still an ever-growing demand for even bigger computing power. One of the ways to satisfy this demand is to offload applications to cloud computing environments. Despite clear advantages and benefits of cloud computing (i.e., fault-tolerance, scalability, reliability, and more elastic infrastructure), it is not easy to abandon existing traditions. The migration of scientific applications to clouds is substantially impeded by difficulties in incorporating new technologies, significant architectural differences, software and hardware incompatibilities, reliance on external libraries. Moreover, scientific applications are mostly tightly-coupled to a specific platform or programs. Taken together, scientific applications face considerable challenges to evolve in accordance with 8 Laws proposed by Prof. Manny Lehman [102]. Prof. Lehman noted that the evolution of a software system is not an easy task. Software shall continuously adapt to a changing operational environment; otherwise, it becomes less appealing and less satisfactory to its users.

Hence, this thesis is directed at finding a solution that will facilitate the adaptation of scientific applications to emerging new technologies – i.e., cloud computing. Specifically, in this thesis we concentrate on scientific applications, which simulate real-world processes by using a computer model as one of the key mechanisms to study the behavior of processes in question. Such simulations are typically performed by means of specialized software, which shows the operation of the process over time. As simulations become more and more complex, the amount of required computing power notably increases. In particular, that is the case of the Monte Carlo simulation, which quality proportionally depends on the number of repeated samplings. The higher the number of sampling procedure invocations is, the better the simulation quality will be. Because Monte Carlo sampling procedures are commonly compute-intensive, the demand for computing power is continuously increasing. In particular, there may be an issue of limited computing resources with regard to Monte Carlo simulations, which rely on the HPC-based infrastructure, i.e., clusters. The higher the demand for high quality simulations is, the more limitations in terms of computing power and overall resources of the underlying infrastructure a researcher may have.

Therefore, in this thesis we will tackle the problem of computing power limitation of HPC-based infrastructure, i.e. clusters, for the Monte Carlo simulations. To approach this problem, we propose to analyze the differences between computing environments, i.e., HPC and cloud computing, define common types of scientific problems, explain why we focus on a Monte Carlo simulation type, elaborate the solution, i.e., cloudification strategies, apply the solution to a common representative, and evaluate the conclusive results.

1.2 Organization of the Thesis

The crucial issues addressed in the thesis are (a) the state of the art of the HPC and cloud computing; (b) current trends and challenges of modern high performance simulators; (c) potential benefits of the big data cloudification methodology for scientific applications; (d) potential benefits of the service-oriented cloudification methodology for scientific applications; (e) comparison of two methodologies and their practical application. The overall structure of the thesis takes the form of seven chapters, including the introduction and conclusion.

Chapter Two begins by laying out the theoretical dimensions of the research. In this chapter, we cover both high performance and cloud computing, their current trends, and challenges. In the end of the chapter, we analyze the potential application of cloud computing to scientific applications.

Chapter Three presents the current state of modern high performance simulations, describes the number of sophisticated tools directed at the execution of scientific workflows, and concentrates on a standalone scientific application, i.e., a simulator. In this chapter, we cover the problem of adapting and migrating legacy scientific simulators to clouds and study a common representative scientific simulator.

Chapter Four analyzes the existing big data cloudification methodology, investigates its limitations, and offers how to enhance and extend it to a wider range of applications, which cannot be defined as suitable following the initial cloudification procedure. In the end of the chapter, we evaluate the application of the enhanced cloudification methodology to the scientific simulator defined in Chapter 3.

Chapter Five is concerned with the service-oriented cloudification methodology directed at solving performance problem of complex iterative scientific applications. To prove the viability of this methodology, we evaluate its application to the defined scientific simulator.

Chapter Six presents the comparison of two cloudification solutions and illustrates their practical application in a conceptual framework for cloud-based hydrological modeling and data assimilation.

Chapter Seven gives a brief summary of the thesis and critique of the findings. Based on the results obtained, we identify areas for further research.

1.3 Contributions

Contribution 1 - enhancing the "big data inspired cloudification methodology"

In collaboration with the group of Prof. Jesus Carretero from the Polytechnical School of University Carlos III of Madrid partially under the COST Action IC1305 "Network for Sustainable Ultrascale Computing Platforms" (NESUS), we adapted one of the state of the

art simulators from the domain of hydrology to Apache Spark framework and demonstrated the viability and benefits of the chosen approach. As a result, we obtained an accepted conference publication:
Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Cloudification of a Legacy Hydrological Simulator using Apache Spark", XXVII Jornadas de Paralelismo (JP2016)*, Salamanca, Spain, September, 2016.*

We generalized the cloudification procedure and obtained a cloudification methodology for scientific iterative workflows using the MapReduce paradigm. This work resulted in an accepted conference publication:
Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Methodological Approach to Data-Centric Cloudification of Scientific Iterative Workflows", 16th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, Pages 469-482, Granada, Spain, December, 2016.*

We identified potential problems and possible improvements of our implementation. Also, we performed extensive testing of the methodology, which resulted in an accepted workshop publication:
Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Lessons Learned from Applying Big Data Paradigms to a Large Scale Scientific Workflow", 11th Workshop on Workflows in Support of Large-Scale Science (WORKS16)*, Salt Lake City, Utah, USA, November, 2016.*

We studied scalability and efficiency of the cloudified application using large and very large pools of computing resources. As a result we submitted a journal article:
Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Applying Big Data Paradigms to a Large Scale Scientific Workflow: Lessons Learned and Future Directions", Future Generation Computer Systems*, submitted in December 2016.*

Contribution 2 - service-oriented cloudification methodology and computing-oriented cloud services

In order to apply the service-oriented cloudification methodology to an iterative Monte Carlo simulator, there is a number of additional requirements, which should be satisfied, e.g., access to external services, mechanism to control computing resource utilization. To satisfy these requirements, we developed a typical cloud-based computational service at the Software as a Service (SaaS) level. The service allows to execute a non-interactive scientific application in a cloud environment and implies very little changes of the the original application. The work resulted in two accepted publications:
Lapin, A.; Schiller, E.; Kropf, P., "Integrated Cloud-based Computational Services", Proceedings of the 7th GI Workshop in Autonomous Systems (AutoSys)*, Pages 280-292, 2014.*
Lapin, A.; Schiller, E.; Kropf, P., "CLAUDE: Cloud-computing for non-interactive long-running computationally intensive scientific applications", Proceedings of the 8th GI Conference in Autonomous Systems (AutoSys)*, Pages 221-232, 2015.*

Contribution 3 - practical application of the cloudification methodologies and a conceptual framework for hydrological data acquisition and management

In collaboration with the Centre for Hydrogeology and Geothermics (CHYN) of the University of Neuchâtel, we analyzed the requirements of a typical hydrological scientific application and developed a prototype of an environmental monitoring system that allows hydrologists to automatically collect and store real-time sensor data in a cloud-based storage. This work resulted in an accepted conference publication:

Schiller, E.; Kropf, P.; Lapin, A.; Brunner, P.; Schilling, O.; Hunkelet, D., "Wireless Mesh Networks and Cloud Computing for Real Time Environmental Simulations", 10th International Conference on Computing and Information Technology (IC2IT2014)*, Pages 1-11, 2014.*

As a follow-up, we developed a prototype of a system that comprises an environmental monitoring and a cloud-based computational subsystems, which resulted in an accepted conference publication:
Lapin, A.; Schiller, E.; Kropf, P.; Schilling, O.; Brunner, P.; Kapic, A.J.; Braun, T.; Maffioletti, S., "Real-Time Environmental Monitoring for Cloud-based Hydrogeological Modeling with HydroGeoSphere" Proceedings of the High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*, Pages 959-965, August, 2014.*

Then, in collaboration with various universities and institutions:

1. Institute of Bio- and Geosciences (IBG-3): Agrosphere, Forschungszentrum Jülich GmbH;
2. Centre for High Performance Scientific Computing in Terrestrial Systems (HPSC-TerrSys), Geoverbund ABC/J;
3. Centre for Hydrogeology and Geothermics (CHYN) of University of Neuchâtel;
4. Communication and Distributed Systems (CDS) group of University of Bern;
5. Department of Earth and Environmental Sciences of University of Waterloo;
6. Aquanty Inc.

we developed a conceptual framework, which aims to provide the user with (i) an easy-to-use real-time access to measurement data from the field, and (ii) dynamic stochastic simulations, which are continuously improved by using the data assimilation approach.
Kurtz, W.; Lapin, A.; Schilling, O.; Tang, Q.; Schiller, E.; Braun, T.; Hunkeler, D.; Vereecken, H.; Sudicky, E.; Kropf, P.; Hendricks Franssen, H.; Brunner, P.; "Integrating hydrological modelling, data assimilation and cloud computing for real-time management of water resources", Environmental Modelling & Software*, Volume 93, Pages 418-435, 2017.*

Chapter 2

State of the art

This chapter provides a detailed state of the art in both high performance and cloud computing. In this chapter, we cover the current trends, challenges and evaluate the potential of applying cloud computing to scientific applications.

2.1 High performance computing (HPC)

High performance computing (HPC) refers to computing with high power provided by a single computer with several processors or a cluster of computers directed at solving scientific or engineering problems that require higher performance than regular computers could provide. Broadly speaking, HPC is associated with such concepts as:

Supercomputing represents systems primarily designed for massive parallel processing with low latency, high bandwidth connections between a large number of co-located processors and typically used for a long execution of an algorithm [156]. The term "supercomputer" originated in the 1960s and it could be described as a computer with an elevated level of computing performance compared to a general purpose computer. The performance of a supercomputer is measured in floating-point operations per second (FLOPS). It is important to emphasize that the performance of a computer is a complicated issue to measure because of its distinctive characteristics, e.g., the computer architecture, hardware, operating system, compiler's ability to optimize the code, application, implementation, algorithm, high level language. Since 1986, TOP500 evaluates the performance of supercomputers by using the LINPACK Benchmark and publishes its ranking every year [3]. The LINPACK Benchmark is used not to evaluate the overall performance of a given machine but to reflect the theoretical peak performance of a dedicated system for solving a dense system of linear equations. The theoretical peak performance is determined by counting the number of floating-point additions and multiplications that can be completed in a period of time, usually the cycle time of the CPU [53]. According to the latest TOP500 list from June 2017, the aggregate performance of all supercomputers in the list is 749 petaflops.

Cluster computing refers to a type of parallel or distributed processing system consisting of a set of interconnected standalone computers operating as a single integrated computing unit [23]. In its basic form, a cluster consists of two or more computers or systems (also known as nodes), which work together to execute applications or perform tasks. High bandwith and low latency connection of nodes is critical to cluster computing performance. Cluster computing is popular among researchers due to the

possible increase of scalability, resource management, availability or processing to a supercomputing level for an affordable price.

Grid computing as a concept was established by Carl Kesselman, Ian Foster, and Steve Tuecke in the 1990s. According to Foster et al. [65], grids could be defined as "a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high end computational capabilities." The key concept behind grids is to establish resource-sharing arrangements among providers and consumers, then to use the resulting resource pool for large-scale computations. Ian Foster [66] suggested a checklist to help define what are, and what are not grids: (i) grids coordinate resources that are not subject to centralized control; (ii) grids use standard, open, general purpose protocols and interfaces; (iii) grids deliver nontrivial qualities of service. One of the key grid computing characteristics is that its size can increase and decrease dynamically depending on the resource costs and availability. Another important feature of grid computing is that it can use heterogeneous and dynamic compute, storage and network resources, which could come from multiple geographically distributed domains having different resource usage policies, hardware and software configurations and platforms. To integrate these geographically distributed resources into a single entity, grids apply a set of protocols and services [67] standartized by the Global Grid Forum. Grids can be assembled over a wide area, by using heterogeneous servers and CPU types or by borrowing spare computing cycles from idle machines in an in-house environment or across the Internet [131]. Essentially, grids are directed at addressing large-scale computation problems by using generally heterogeneous and dynamic resources from multiple geographically distributed domains.

High Throughput Computing (HTC) was firstly developed to address scientific problems, which require extended periods of time (e.g., weeks, months) to perform a computational task. To meet this demand, the Center for High Throughput Computing at UW-Madison developed the first high throughout computing system HTCondor to enable scientists and engineers to increase their computing throughput [13]. In 1996, the HTCondor team presented the HTC system and underlined its distinction from the HPC one, i.e., the ability to deliver enormous amounts of computing resources and processing capacities over extended periods of time [140]. The purpose of HTC is to efficiently harness the use of all available computing power. It is specifically related to an environment with distributed ownership (thousands of personal computers instead of one supercomputer) when throughout an organization users own computing resources. In this case, the utilization of all organizational computing power to its full extent may be low as many desktop machines will be idle when their users are occupied. The HTCondor, a software system that creates an HTC environment, effectively utilizes the computing power of workstations communicating over a network. When the job is submitted to the HTCondor, it finds an idle machine, runs the job on it, and, if the machine is no longer available, migrates the job to another machine [162].

Many Task Computing (MTC) refers to a system directed at completing many independent and dependent computational tasks by using enormous number of computing resources over short periods of time. In contrast to HTC computations (completed in days, weeks), MTC primary metrics are in seconds, e.g., tasks/sec, FLOPS. Primarily, Raicu et al. [138] coined the term MTC to draw attention of the scientific community to heterogeneous applications that are not well parallelized. MTC denotes the high performance computation tasks, which could be small or large, static or dynamic,

homogeneous or heterogeneous, compute-intensive or data-intensive, loosely-coupled or tightly-coupled. MTC aims to bridge HPC paradigm, which covers tightly-coupled communication-intensive tasks, with HTC, which operates with long-running parallel tasks [99, 138].

In this section, HPC and related concepts were overviewed. In summary, HPC aims at delivering higher computing performance, than a typical desktop computer can provide, in order to solve large and complex problems. In the next subsections, we will review the HPC anatomy, i.e., architecture, programming models, current trends and challenges.

2.1.1 HPC architectures

Since many years, the taxonomy developed by Flynn is still considered to be useful for the classification of high performance computer architectures [64]. This classification consists of four major HPC architectural classes:

1. Single instruction stream, single data stream (SISD):

   These systems contain one single processor (one CPU core) that accommodates one instruction stream directed at operating on data stored in one single memory. Some examples of SISD systems could be older generation mainframes, minicomputers, workstations and single processor/core PCs.

2. Single instruction stream, multiple data streams (SIMD):

   These systems contain a large number of processing units (from 1'024 to 16'384 PUs) and a single control unit (CU) that could execute the same instruction (an identical action) to operate on multiple data pieces in parallel. There are two types of SIMD systems: array and vector processors. Array processors operate on multiple data pieces at the same time, while vector processors act on data arrays in consecutive time slots. SIMD systems like GPUs are well suited for problems with a high degree of regularity like graphics or image processing. Some other examples of the SIMD-based systems are IBM's AltiVec and SPE for PowerPC, Intel's MMX and iwMMXt.

3. Multiple instruction streams, single data stream (MISD):

   In theory, MISD systems have multiple processors and perform multiple instructions using a single input data stream. However, no practical solution of such an architecture has been constructed yet. MISD might be well used for multiple frequency filters operating on a single data stream or multiple cryptography algorithms directed at cracking a single encrypted code.

4. Multiple instruction streams, multiple data streams (MIMD):

   The MIMD architecture is typical for computers with multiple processors. These systems execute multiple instructions in parallel over multiple data streams. MIMD systems can run many sub-tasks in parallel to shorten the execution time of the main task. MIMD execution can be synchronous or asynchronous, deterministic or non-deterministic. Currently, most common examples of MIMD are desktop PCs.

Considering a vast variety of SIMD and MIMD systems, it is important to elaborate more on their classification [160]:

Distributed memory systems are systems with multiple CPUs having their own associated private memory. The CPUs are interconnected and may exchange data among their memories if it is required. Both SIMD and MIMD can be distributed memory systems.

Shared memory systems have multiple CPUs sharing the same address space and accessing one memory resource on an equal basis. Both SIMD and MIMD can be shared memory systems.

Overall, the Flynn's taxonomy and its enhanced version proposed by Van der Steen [160] aims to break computer architecture down by the number of instructions and data streams that can be handled in parallel.

2.1.2 HPC programming models

Regarding the HPC software development, the most frequently used programming models are the following:

Message Passing Interface (MPI) is a programming interface also known as a thread-safe application programming interface (API), which defines the syntax and semantics of a software library and provides standardized basic routines necessary for building parallel applications, as noted by Frank Nielsen [120]. MPI is directed at developing parallel programs that encapsulate data and exchange it by sending/receiving messages [120]. To put it simple, MPI aims to provide a standard for developers writing message passing programs. Its interface stands for being practical, efficient, portable, and flexible. It is important to emphasize that MPI does not depend either on a memory architecture (i.e., distributed or shared memory) or a programming language (MPI commands can be used with different languages, i.e., Java, Python, C, C++, Fortran). This programming interface is portable, widely available, standartized, and highly functional (with more than 430 functional routines).

High Performance Fortran (HPF) is defined by Harvey Richardson [143] as "a programming language designed to support the data parallel programming style". HPF was developed to secure high performance of parallel computing, which does not sacrifice portability. HPF is based upon the Fortran 90 programming language and extends it by providing support for controlling the alignment and distribution of data on a parallel computer; adding new data constructs; extending the functionality at a higher level of abstraction, and addressing some sequence- and storage-related concerns. According to Kennedy et al. [97], HPF pioneered a high level approach to parallel programming, initially gained high community interest in the language and its implementation but failed to build a strong user base.

Pthreads or POSIX threads is a standardized C language threads programming interface defined in the POSIX standard [5]. According to Narlikar et al. [117], Pthreads can be implemented either at the kernel level or as a user-level threads library. The first implementation approach is relatively expensive in terms of making thread operations. Though, it provides a single, rigid thread model with access to system-wide resources. The second approach provides an implementation in user space without kernel intervention. Thus, it is significantly cheaper in making thread operations. Overall, both implementation approaches are well suited for writing parallel programs with many lightweight threads. In the POSIX model, threads share dynamically allocated heap

memory and global variables. This may cause some programming difficulties. For instance, when multiple threads access the same data, programmers must be aware of race conditions and deadlocks to protect a critical section of the code. In general, the POSIX model is well suited for the fork/join parallel execution pattern. Although, Diaz et al. [51] do not recommend Pthreads as a general purpose parallel program development technology due to its unstructured nature.

OpenMP is a shared memory API directed at easing the development of shared memory parallel programs. The characteristics of OpenMP allow for a high abstraction level, making it well suited for developing HPC applications in shared memory systems. Overall, at the lowest level, OpenMP is a set of compiler directives and library routines, which are accessible from Fortran, C and C++ to express shared memory parallelism [42]. At the higher level, the OpenMP supports the execution of both parallel and sequential programs [126]. The switch between parallel and sequential sections follows the fork/join model of parallel execution [51, 94]. Overall, in OpenMP the use of threads is highly structured. Altogether that makes OpenMP a well suited alternative to MPI.

In general, every HPC programming model presented above has its place in specific situations. Its choice depends more on software preferences, programmer's expertise, and available development time than on a hardware.

2.1.3 Current trends and challenges

Strohmaier et al. [149] analyzed the HPC market, its evolution, and trends from the advent up to 2005. The HPC market was born in the 1970s with the introduction of vector computing (typically represented by aforementioned SIMD-based systems), which offered higher performance than a typical system. In the end of the 1980s, parallel computing with scalable systems and distributed memory gained popularity. At that time, massively parallel processing (MPP) systems, still SIMD-based, were developed providing the highest performance. In the early 1990s, symmetric multiprocessing (SMP) systems became popular due to their performance/price ratio suitability for both low and medium HPC market segments. Typically, SMP systems were MIMD-based and well adopted both in the industry and academia. In the 2000s, computer clusters gained popularity; by 2005 most of the supercomputers presented in the TOP500 list were clusters. By now HPC clusters are massively used in the large, government-funded organizations, academic institutions, and commercial sectors, e.g., aerospace, automotive, manufacturing, energy, life sciences, financial institutions. In all these fields, HPC is used for large computations with ensured high levels of accuracy and predictability [131].

According to the TOP500 overview of the IDC's Forecast 2016-2020 for the HPC Market [10], the demand for HPC continues growing, with a compounded annual growth rate of 8% from 2015 to 2019. IDC forecasts that the HPC growth is fueled by performance-critical areas, e.g., deep learning, artificial intelligence, and big data. IDC calls these areas high performance data analytics (HPDA), which is growing at high compounded annual growth rates both in the academia (16.8%) and industry (26.3%). However, it is also important to emphasize that in recent years TOP500 supercomputers have shown a general slowdown in the performance advancements. It is difficult to identify the key reason behind this slowdown, whether it is related to budgetary constraints imposed on businesses and government organizations or it is related more to time and cost necessary to deploy and maintain HPC infrastructures. When there is a need to maintain the advancement rate of certain domains,

an alternative could be to offload applications to another environment, e.g., cloud computing.

In general, this subsection illustrated that the demand for HPC continued growing. There is also a need for an alternative that could solve efficiently and cost-effectively data- and compute-intensive problems in case the possibility to deploy and maintain an HPC infrastructure is impeded.

2.2 Cloud computing

The idea behind cloud computing is not new. In the 1950s, a gradual evolution towards cloud computing started with mainframe computing. Considering the high cost of purchasing and operating multiple computers at every organization, a shared access to a single computing resource, a central computer, provided an economic benefit. In the 1970s, the concept of virtual machines was established. By using virtualization, it became possible to execute one or many operating systems in parallel on one physical hardware. The virtual machine took the mainframe computing to a new level as it enabled multiple distinct computing environments to operate inside one physical machine. In the 1990s, the telecommunication companies started offering virtualized private network connections at a reduced cost. Instead of getting a personal connection, an end user got shared access to the same physical infrastructure. Altogether, this served as catalysts for the development of cloud computing [8].

There are many definitions of the cloud computing. Hereafter, the most relevant definitions will be presented. According to the National Institute of Standards and Technology (NIST) [80], cloud computing is "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models".

According to a joint technical committee of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) [86], cloud computing is "a paradigm for enabling network access to a scalable and elastic pool of shareable physical or virtual resources with self-service provisioning and administration on-demand. The cloud computing paradigm is composed of key characteristics, cloud computing roles and activities, cloud capabilities types and cloud service categories, cloud deployment models and cloud computing cross cutting aspects".

Whereas Cisco [24] defined cloud computing through three key attributes characterizing a cloud computing system:

1. **On-demand** provisioning means that resources can be delivered immediately when needed, released when no longer required, and billed only when used.
2. **At-scale** service means that the service provides the illusion of infinite resource availability in order to meet the requirements.
3. **Multitenant environment** means that the resources are provided to many consumers from a single pool at the same time, saving the provider from significant costs.

In Cisco's definition, all the three attributes are required to define a cloud service. Interestingly, the physical location of resources (on-premise or off-premise) is not a part of the definition.

Alternatively, IBM [19] defined cloud computing to be "an all-inclusive solution in which all computing resources (hardware, software, networking, storage, and so on) are

provided rapidly to users as demand dictates. The resources, or services, that are delivered, are governable to ensure things like high availability, security, and quality. The key factor to these solutions is that they possess the ability to be scaled up and down, so that users get the resources they need: no more and no less. In short, cloud computing solutions enable IT to be delivered as a service."

Considering all these relevant definitions, we adopt the following definition in the context of this thesis:

Cloud computing is a popular paradigm, which relies on resource sharing and virtualization. Cloud computing provides a consumer with a transparent, scalable and elastic computing system having computing, network, and storage resources and being able to expand and shrink on-the-fly.

The reason for the existence of distinct perceptions and, consequently, definitions of cloud computing is that it is not a new technology but rather a set of various services (2.2.1.1) and deployment models (2.2.1.2) bringing together the enabling technologies (2.2.2) to meet the technological and economic requirements of today's service-oriented demand (2.2.3).

2.2.1 Cloud computing models

Cloud computing [161] is continuously evolving, delivering various computing resources as services to a consumer. Taking into account the NIST Special Publication [80], the cloud computing essential characteristics could be defined as:

1. **On-demand self-service**, when a consumer can acquire computer services such as applications, networks, or servers without any interaction with the service provider.
2. **Broad network access**, when sufficient cloud capabilities are available over the network and accessed through standard network infrastructures and protocols.
3. **Resource pooling**, when computing resources are pooled to serve multiple consumers by using a multi-tenant model with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Even though a consumer has no control or knowledge over the location of provided resources, but it is possible to specify the location at a prominent level, e.g., country, state, datacenter.
4. **Rapid elasticity** and **scalability**, when the consumer has access to unlimited capabilities rapidly and elastically provisioned on a self-service basis almost in real time.
5. **Measured Service**, when the cloud computing resource usage appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts) can be transparently measured, controlled, and reported both by the provider and the consumer.

The aforementioned characteristics bring a considerable value to the consumer by lowering operating cost due to the pay-as-you-go model, minimizing management effort due to maintenance and administration of the infrastructure by a third party, and providing virtually unlimited resources on demand. Overall, that makes cloud computing attractive to the industry and facilitates its adoption.

2.2.1.1 Service models

According to the NIST Special Publication [80], there are only three service models: Cloud Software as a Service (SaaS), Cloud Platform as a Service (PaaS), and Cloud Infrastructure as a Service (IaaS). Interestingly, different authors extended this list and enlarged it with additional service models, e.g., Database as a Service, Security as a Service, Network as a Service, Simulation Software as a Service. Consequently, it is possible to emphasize the current trend described well in Subsection 2.2.3 that cloud computing is emerging to become Everything/Anything as a service (XaaS) model. Many scholars hold the view that three models (SaaS, PaaS, and IaaS) are the major ones and they could be presented as a pyramid (Figure 2.1), in which the Software as a Service will be placed at the top of the pyramid, while the Infrastructure as a Service will be placed at the bottom [19, 112, 146]. The following brief overview presents there service models.

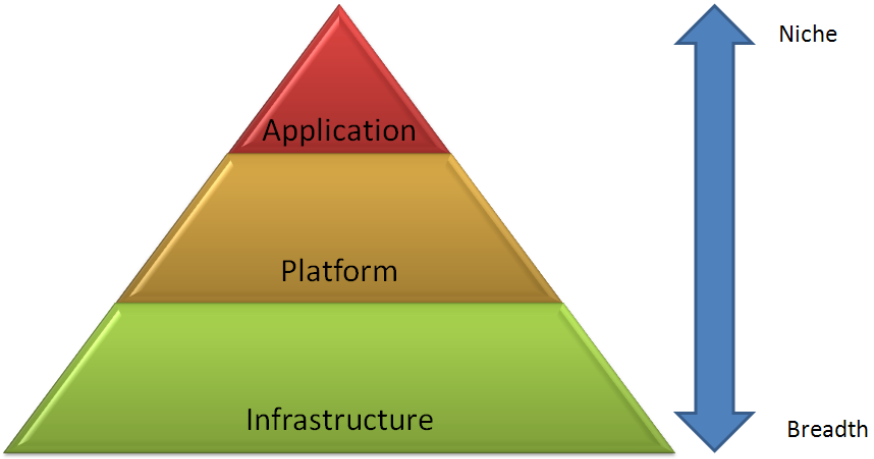


Figure 2.1: Cloud Computing Pyramid [146]

Service model 1: the Software as a Service (SaaS)

SaaS model delivers a consumer the access to software applications over the Internet. This refers to prebuilt and vertically integrated applications (e.g., an email system, an ERP, a CRM, a human resource management system, a payroll processing system) that are delivered to and purchased by users as services. Besides some user-specific application configuration settings, the consumer has no control over the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities. The applications are centrally managed on the cloud, possibly by a third-party vendor, and can be accessed with a web browser without any preliminary installation. This approach simplifies the application usage experience for the end users and excludes necessity to update the software or deal with licenses. SaaS may be regarded as a user layer, which can be further classified into services (a particular application for a specific use, e.g., Gmail) and applications (a unified software that incorporates many business functions, e.g., SAP Business ByDesign). The market of SaaS products is very broad as services can be anything from Web-based email to enterprise resource planning (ERP). Interestingly, Cisco [83] forecasts that by 2020, 74 percent of the total cloud workload will be Software as a Service (SaaS), up from 65 percent in 2015.

Service model 2: the Platform as a Service (PaaS)

PaaS model (e.g., OpenShift) delivers an application development platform allowing developers to concentrate on software design, development, and deployment rather than maintenance of the underlying infrastructure. Developers do not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but have control over the deployed applications and possibly application hosting environment configurations. PaaS could be considered as a developer layer intended to support the SaaS layer of the pyramid. Typical PaaS offering includes runtime environment for application code, cloud services, computing power, storage, and networking infrastructure. Developers are usually charged per each billing period for storage (per GB), data transfer in or out (per GB), and I/O requests (per n thousand requests), etc. Good examples of PaaS services are AWS Elastic Beanstalk, Windows Azure, Heroku, Force.com, Google App Engine, Apache Stratos. According to Cisco [83] forecast, by 2020, 8 percent of the total cloud workload will be Platform as a Service (PaaS), down from 9 percent in 2015.

Service model 3: the Infrastructure as a Service (Iaas)

IaaS model (e.g., Amazon, OpenStack, OpenNebula) delivers any kind of system resources on demand (e.g., compute power, storage, networking) and allows users to pay as they go for the actual consumption. The consumer does not manage or control the underlying cloud infrastructure but has control over CPU operating systems, storage, deployed applications, and possibly limited control over certain networking components (e.g., host firewalls). IaaS is the lowest layer of the pyramid that supports PaaS and SaaS layers. The added value of IaaS is the use of a complex infrastructure on a pay-for-what-you-use basis rather than investing upfront in building and maintaining an infrastructure (e.g., hardware, software, servers, datacenter facilities). Regarding Cisco forecast [83], by 2020, 17 percent of the total cloud workloads will be Infrastructure as a Service (IaaS), down from 26 percent in 2015.

2.2.1.2 Deployment models

According to National Institute of Standards and Technology (NIST) [80], there are four cloud deployment models:

Deployment model 1: private cloud

The private clouds, also known as internal clouds, are proprietary networks; often, its data centers reside within the organization [112]. It can be managed by the organization or a third party and can be on premise or off premise. In a case of being on premise, the organization oversees setting up and maintaining the cloud. Thus, it is the organization who is responsible for cloud resources provisioning and functioning. The added value of private clouds for the organization is a possibility to control security and quality of service aspects. It is especially relevant in case of managing critical processes and storing sensitive data [80].

Deployment model 2: community cloud

The community cloud infrastructure is semi-private as it is shared by several organizations and supports a specific group of tenants with similar backgrounds that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). Then the cloud

infrastructure becomes a private cloud for this specific community. It may be managed by the organizations or a third party and exist on premises or off premises.

Deployment model 3: public cloud

The public clouds, also known as external clouds, are cloud services owned by cloud service providers that are made available to a general public or a large industry group. The cloud providers are responsible for installation, management, provisioning and maintenance of the cloud infrastructure. Considering the variety of cloud services provided to a generic public, the pooling of resources becomes more efficient. In a public cloud, the infrastructure is organized and managed entirely by the cloud provider. Hence, the security mechanisms (for example, encryption, replication) may not be fully transparent to the user.

Deployment model 4: hybrid cloud

The hybrid clouds are a composition of two or more clouds that remain unique entities but are bound together by a standard or a proprietary technology. This enables data and application load balancing in order to avoid overload of one single cloud. Hence, hybrid clouds can be valuable for the user who manages dynamic and highly changeable environments. However, they can also present some management challenges because responsibilities are split among an organization, a community and/or a public cloud provider.

This subsection presented an overview of the cloud computing anatomy from different perspectives: essential characteristics, service and deployment models. The cloud computing distinguishes itself from other computing paradigms by its attractive features, i.e., on-demand, flexible, easy-to-measure, virtually unlimited resource obtainable services. Clouds can be classified according to their offered service models, i.e., SaaS, PaaS, IaaS. Clouds can also be classified according to their deployment models, i.e., private, community, public, hybrid. Thereby, this subsection laid the foundations for the further discussion of the IT complexity and technologies behind the cloud computing paradigm.

2.2.2 Enabling technologies behind cloud computing

While a consumer experiences cloud computing as an easy-access and easy-to-use technology, cloud computing is a complex and powerful paradigm with a multitude of important technologies behind. In 2008, Wang et al. [164] published a paper, in which they identified a number of enabling technologies behind cloud computing presented below.

Virtualization technology

In cloud computing, virtualization holds a crucial position. Virtualization is a very old concept that dates back to 1974 when Popek and Goldberg derived the first set of formal requirements for efficient virtualization [135]. The term virtualization refers to provisioning a virtual resource (e.g., a computer network resource, a storage device, a server) that behaves identically to its physical counterpart. The idea is to run several instances of the operating system (OS) called guests or virtual machines (VMs) on a single physical machine (bare metal), hereafter called host. Virtualization is provided through a special piece of software called a hypervisor or a virtual machine manager (VMM) running on a physical machine. Hypervisors can run directly on the system hardware or on a host operating system. The

latter allows to delegate tasks like I/O device support and memory management to the host OS.

Overall, virtualization allows to split the physical infrastructure and create dedicated virtual resources. Moreover, it is a key component of the Infrastructure as a Service (IaaS) cloud, which enables tenants to access VM-based computing resources. The tenant can scale resources in and out to meet its demand, for example, by releasing idle VMs, or requesting new VMs upon a workload burst.

Web services and SOA

Considering the fact that normally cloud computing services are exposed as Web services, they also follow such industry standards as Universal Description, Discovery and Integration (UDDI), Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP), an XML-based RPC, and a messaging protocol [55]. In brief, UDDI is a registry mechanism that can be used to look up Web service descriptions. WSDL is a descriptive interface and protocol binding language. While SOAP is a service-oriented architecture protocol, which allows distributed applications to communicate by using HTTP and XML. The cloud computing services could be designed and managed by a Service Oriented Architecture (SOA). SOA is a set of software design principles, patterns, and criteria, which address modularity, encapsulation, reusability, and loose coupling of distributed application components, i.e., services. By applying the industry standards and SOA, cloud computing services become interoperable, dynamic, and composable.

Worldwide distributed storage system

According to Cisco [83], the need for distributed data storage centers and cloud resources accelerates drastically due to the forecasted almost quadruple (3.7-fold) growth of cloud traffic, which will increase from 3.9 ZB in 2015 to 14.1 ZB in 2020. The cloud storage model should foresee the need for data storage migration, merger and transparent management by the end user, for example, as it is done by Google File System and Amazon S3.

Programming model

For easy access to cloud services, cloud programming models intend to hide its underlying complexity from non-specialized individuals. For instance, MapReduce is a programming model and a corresponding implementation for simplified processing and generation of large volumes of data [44]. Programs written in its functional style are automatically parallelized and executed. This allows programmers not experienced with parallel and distributed systems to easily utilize the resources of a large distributed system [90]. For example, Hadoop is a MapReduce implementation, which allows the user to process large data sets across clusters of computers. Hadoop was primarily designed to detect and handle failures at the application layer. Thus, it allows programmers to store and read a large amount of data in reliable, fault-tolerant, and cost-effective way.

This subsection presented a short overview of the technologies behind the cloud computing paradigm and paved the way for the further discussion of a current service-oriented demand and the evolution of cloud computing service models to a broader form of Everything/Anything as a Service (XaaS).

2.2.3 Service oriented approach to resources

While three major service models are SaaS, PaaS, and IaaS, different authors extended the list with additional resources presented as services, for example, Storage as a Service, Database as a Service, Security as a Service, Network as a Service, Communication as a Service, Integration as a Service, Testing as a Service, Business Process as a Service, Simulation Software as a Service. Altogether, these broad categories of services related to computing and remote access form a larger set called Everything/Anything as a Service (XaaS). Virtualized physical resources, virtualized infrastructure, as well as virtualized middleware platforms and business applications are provided and consumed as cloud services. A rich ecosystem of cloud computing services and providers has emerged and it forms a complex environment, in which Web applications and services are developed, tested, deployed, and operated. While this complex environment provides many choices, at the same time it poses a great challenge to engineers in charge of building resilient application architectures. The growing diversity of services, frameworks, platforms and tools within the cloud computing community tends to obfuscate the reasonable use and combination of the advertised offerings. In this subsection, the most distinct service models established in recent years will be categorized and overviewed.

Service model 1: DataBase as a Service (DBaaS)

It is difficult to argue with the importance of relational database management systems (DBMSs) or diminish their value, which became an integral and indispensable component in most computing environments today. With the advent of hosted cloud computing and storage, the opportunity to offer a DBMS as an outsourced service is gaining a momentum, as well illustrated by Amazon's RDS and Microsoft's SQL Azure [41]. The Database as a Service (DBaaS) gains attention for two major reasons. First, due to economies of scale, the users will likely pay less for the database service than when they buy the hardware and run it themselves. Second, when the DBaaS is well designed, any costs incurred are proportional to how much it was used (a pay-per-use payment model), the same applies to both software licensing and administrative costs. The latter costs often accelerate significantly when a specialized expertise is required to obtain a satisfactory performance from commodity DBMSs. When database management tasks are well centralized and automated, DBaaS operational costs can be substantially reduced, while maintaining the performance [41].

There are three challenges that drive the design of relational database management systems: efficient multi-tenancy to minimize the hardware footprint required for a given (or predicted) workload, elastic scale-out to handle growing workloads, and database privacy. Considering the existing offerings like Amazon RDS and Microsoft SQL Azure, they validate the market need for DBaaS. However, it is important to emphasize that they are constrained as both DBaaS offerings support only a limited consolidation (often based on VMs), lack scalability beyond a single node (i.e., a server), and have considerable issues with data privacy or processing of queries over encrypted data [41]. Considering the DBaaS multi-tenancy, the key challenge is related to dealing with big data and overcoming the database limitations on extremely large amount of tables and/or columns. The scalability of database systems is a popular issue both in academic and business areas as both approaches (NoSQL and SQL-based systems) have some limitations, e.g., the former sacrifices a fair amount of expressive power and/or consistency in favor of extreme scalability, while the later limits the type of transactions allowed. At this stage, all the attempts to preserve consistency and achieve scalability via workload-aware partitioning are too expensive to be put in practice [41].

Service model 2: SECurity as a Service (SECaaS)

Clouds offer attractive options to migrate corporate applications without any implication for management or security of physical resources. However, the migration of critical business applications with sensitive data can be limited due to the lack of transparent security mechanisms from cloud service providers (CSPs). Typically, the cloud service consumers have security requirements to fulfill due to regulatory compliance but to verify how well the CSP could satisfy these requirements is still difficult. Recently, the cloud security community, represented by workgroups of the European Network and Information Security Agency (ENISA) and the Cloud Security Alliance (CSA), has identified that specifying security parameters in Service Level Agreements (SLAs) could serve well to establish the common ground for the CSPs and standartize the security issues from both the user and provider sides. SLAs lay the foundation for the transparency of the security mechanisms of the provided cloud services.

The growth of the SaaS market has also given a rise to a new generation of tools, platforms, and hardware, which can provide protection for cloud-based applications, data, business processes. This generation of tools is known as Security as a Service (SECaaS). According to MarketsandMarkets^{TM}, the SECaaS market is expected to grow from USD 3.12 billion to USD 8.52 billion by 2020. The types of SECaaS solutions are Data Loss Prevention (DLP), Security Information and Event Management (SIEM), email encryption, and endpoint protection. Cloud security could also be ensured from the hardware perspective. For example, if cloud providers integrate Intel SGX into their infrastructure, then cloud users will be able to protect critical code or/and data from disclosure or modification.

Service model 3: Network as a Service (NaaS)

Most cloud applications are distributed by nature and often involve significant network activity to perform their operations. Today, in current cloud computing offerings, tenants have little control over the network. Tenants can accurately select the number and types of computational and storage resources needed for their applications, but they cannot directly access and manage the network infrastructure (i.e., routers or switches) unless it is a private or community cloud. This means that all packet processing must occur at the end hosts. Even some relatively common communication operations, such as multicast, are not supported, requiring tenants to implement them using inefficient application-level overlays. Triggered by lowering costs and increasing performance, a new approach towards software- and FPGA-based programmable routers arose and aimed at replacing the traditional switches' and routers' operations (e.g., IPv4 forwarding) with a custom implementation [40]. This approach evolves to a new stage of a Network as a Service model when this flexibility could be provided to tenants in order to implement part of the application logic in the cloud network. NaaS could be described as a new cloud computing model, when tenants have access to additional computing resources collocated with switches and routers. Tenants can use NaaS to implement custom forwarding rules based on application needs, i.e., a load-balancing anycast or a custom multicast service. Tenants can process packets on-path, even modify the payload or create new packets on the fly. This enables the design of efficient in-network services, such as data aggregation, stream processing, caching and redundancy elimination protocols, that are application-specific as opposed to traditional application-agnostic network services. NaaS model is expected to be widely adopted, however, there are considerable limitations to overcome – i.e., scalability, multi-tenant isolation of different network resources, and support of distinct solutions developed by various organizations and executed concurrently.

Service model 4: SiMulation as a Service (SMaaS)

In the academia, there is a need for services that could provide researchers and scientists with tremendous computing resources and substantial amount of storage necessary for conducting scientific simulations and experiments. Simulation software, which is based on the mathematical process of modeling a real operation or phenomenon, is widely used in simulating weather conditions, heat pumps, chemical reactions, complex biological processes, etc. It may manifest the following characteristics [77]: long-running time, high resource consumption, complicated I/O, interconnection with other simulation software(s). Typically, researchers and field experts need to have either the simulation software installed on their own machines or the access to machines that can host a simulation. Sometimes, experiments may be disrupted when there is a dependency between different simulations, while one of them may not be immediately available. With the advent of cloud computing and SOA, researchers and scientists are exposed to software, platform, and infrastructure on the Internet. In the cloud environment, a researcher could conduct an experiment by connecting multiple simulation software services to form a workflow, which represents how the experiment proceeds. Even though simulation could be wrapped into web services, there is still one intrinsic difference between regular software and simulation – it's fundamental for the simulation to be dynamic in its nature and have its elements constantly changing over time. Thus, the time factor is crucial and critical to the correctness of the simulation in order to guarantee and maintain the correct temporal order of the events occurring in the simulation. Comparing to regular services, simulation services are tightly-coupled and may incur extra work [77].

The aim of Subsection 2.2.3 was to identify the service-oriented offering models that altogether form a broad cloud computing concept of Everything/Anything as a Service (XaaS). While there exists a rich ecosystem of cloud service offerings, both the academia and industry face considerable challenges of engineering resilient and reliable services, standardizing them, and meeting all the security demands. Hence, in the next subsection we will overview the major challenges that may hinder the rapid development and adoption of cloud computing services.

2.2.4 Current challenges

Although the cloud computing became popular both in the industry and academia, the cloud computing market is considered to be in its infancy. According to the cloud computing market maturity study results [85], cloud computing is considered as a relatively immature service offering, especially IaaS and PaaS that are on the stage of a small adoption with a potential for growth, though SaaS has started demonstrating a significant adoption with a notable innovation in terms of product offerings. There are still many challenges and issues to be considered in order to accelerate the cloud computing adoption. Some of these challenges are briefly described below:

Challenge 1: security

It is obvious that the security issue plays a crucial role in the adoption of cloud computing due to the fact that the cloud service consumer puts the data and runs the software at the third party's side. Security issues such as data loss, phishing, password weaknesses, and compromised hosts running botnets pose serious threats to the organization's data and software and, consequently, delay its cloudification [52]. Furthermore, with multi-tenancy and pooled computing resources, new security challenges arise. First, in a shared resources

environment, unexpected side channels, which passively monitor information, and covert ones, which actively send data, can emerge. Lastly, there is an issue with a reputation fate-sharing, which can arise when a single subverter disrupts many users, who rely on cloud providers, ensuring the use of the cloud security best practices. Because the users can share the same network address, the disruption will be allocated to every tenant without distinguishing the real subverter [38].

Challenge 2: standardization

According to Sixto Ortiz Jr. [127], the rapid growth and adoption of cloud computing is limited and threatened by the failure of comprehensive cloud-computing standards to gain traction, regardless how many groups work on them, i.e. IEEE Working Groups P2301 and P2302, Distributed Management Task Force (DMTF), Open Grid Forum (OGF), Organization for the Advancement of Structured Information Standards (OASIS), Storage Networking Industry Association (SNIA). The lack of standards could make cloud computing trickier to use and restrict implementation by limiting interoperability among cloud platforms and causing inconsistency in the area of security. Despite having no control over the underlying computing resources, the cloud service consumer shall be ensured with the quality, availability, reliability, and performance of the computing resources. But because there is a a lack of effective cloud services standardization, it is difficult for consumers to compare, evaluate, and choose the best cloud offering. However, there is a list [127] of in-progress standards that are based upon the essential characteristics defined by NIST and described in Subsection 2.2.1:

1. Open Virtualization Format (OVF) by DMTF;

   OVF establishes a transport mechanism for moving virtual machines from one hosted platform to another.

2. Guide for Cloud Portability and Interoperability Profiles (CPIP) by IEEE Working Groups P2301;

   CPIP serves as a metastandard for existing or in-progress standards in critical areas such as cloud-based applications, portability, management, interoperability interfaces, file formats, and operation conventions.

3. Standard for Intercloud Interoperability and Federation (SIIF) by IEEE Working Groups P2302;

   SIIF creates and defines the topology, protocols, functionality, and governance necessary for cloud-to-cloud interoperability and data exchange.

4. Open Cloud Computing Interface (OCCI) by OGF;

   OCCI provides specifications for numerous protocols and APIs for different cloud-computing management tasks, i.e., automatic scaling, deployment, and network monitoring.

5. Symptoms Automation Framework (SAF) by OASIS;

   SAF is directed at providing CSPs with the guide to understand the consumer requirements, e.g., quality of service and capacity, to design better services.

6. Cloud Data Management Interface (CDMI) by SNIA;

   CDMI standartizes client interactions with cloud-based storage, cloud data management, and cloud-to-cloud storage interactions.

Challenge 3: cloud interoperability

Currently, each cloud offering is proprietary and defines its own way of client/application/user interactions with the cloud. That considerably hinders the development of cloud ecosystem as clients are locked-in and could not choose an alternative cloud offering and easily switch to it due to the high costs. The user lock-in could be attractive to the vendor but in this situation clients are particularly vulnerable to price changes, reliability problems, or even the vendor's bankruptcy. Moreover, it is very important to emphasize that proprietary APIs cause difficulties in cloud services' integration with an organization's existing local systems. The scope of this challenge refers both to the interoperability across clouds and between cloud and private systems. Fox et al. offer one solution, which could deal with the whole scope of this challenge – a standardization [68]. Considering the argument of Dillon et al. [52] that Microsoft and Amazon, the two major cloud service providers, do not support the Unified Cloud Interface (UCI) Project proposed by the Cloud Computing Interoperability Forum (CCIF), it is clear that the standardization progress is considerably inhibited as the big market players could not reach consensus.

Challenge 4: energy efficiency

As stated in McKinsey report [95], data centers consume the enormous amount of energy and, consequently, emit carbon dioxide at the same level or even more than both Argentina and the Netherlands. According to Cisco [83], by 2020 the number of large-scale public cloud data centers will almost double. That means that the energy consumption will accelerate too. Moreover, cloud data centers consume electricity, especially when resources are always available. An idle server consumes about 70% of its peak power. This waste of idle power is considered as a major cause of energy inefficiency [71]. As the energy costs increase while the availability diminishes, there is a clear need for energy efficiency optimization while maintaining high service level performance and availability.

This subsection provided an overview of cloud computing challenges and illustrated that the development and adoption of cloud services may be substantially impeded by security and privacy issues, not widely adopted standards, lack of interoperability and portability, and energy inefficiency. Considering all the advantages and drawbacks presented in Section 2.2, we will evaluate the application of the cloud computing potential to compute-intensive scientific applications in the next Section 2.3.

2.3 Cloud computing for scientific applications

The distinct characteristic of scientific computing is associated with ever-increasing need for computer resources to conduct experiments, simulations and gain outcomes in a reasonable time frame. Recently, there were only two feasible options how to satisfy this need. It was to use either expensive supercomputers or cheap commodity resources, e.g., clusters, grids. With cloud computing, this need could be alternatively satisfied by leasing necessary computing resources from large-scale data centers. Even though several cloud offerings, e.g., Amazon, GoGrid, exist on the market, the cloud potential for scientific high performance computing is not yet considerably investigated. Considering this issue, the current section presents a brief analysis of cloud computing and high performance computing (HPC) or high throughput computing (HTC) capabilities for scientific applications.

Cloud computing holds an immense potential for scientific computing. Clouds can be cheaper than supercomputers, more reliable than grid computing, and much easier to scale than clusters. Clouds charging model "pay-as-you-go" brings additional advantages, i.e., limitless scale up with only one constraint – financial budget, while clusters and supercomputers are physically limited to adding additional nodes. Besides, clouds also incur significant challenges, especially in performance. The root for challenges is within main differences between cloud and scientific computing: the job execution model, system size, and performance demand. Scientific job execution model mostly requires space-shared use of resources, while clouds mostly rely on time-shared resources and use virtualization to abstract away from physical hardware resources, which distinct advantage is the increase of users' concurrency, while its disadvantage is the potential decrease of achievable performance. Considering system size, scientific facilities include very large systems, while cloud computing services are mostly directed at substituting small and medium-sized data centers.

The scientific community has shown a clear interest in shifting HPC- or HTC-based applications to clouds, though it has also raised an important question: *"Could the clouds' performance be sufficient for scientific computing and its applications?"* Putting it differently, *"Could clouds execute the high performance scientific tasks at the same or similar performance level as HPC/HTC?"*

Considering this question, some scientific works were directed at exploring data-intensive workflows, since they are tightly related to conventional scientific applications in terms of data volumes [106, 181]. Some experiments with well known workflows, e.g., Montage, clearly presented findings that running costs could significantly decrease with cloud computing infrastructure, while performance would still suffer from virtualization and latency overheads [27, 48, 79]. While Juve et al. [92] measured the usefulness and value of cloud computing for scientific workflows by conducting experiments on three workflow applications like Montage (an astronomy application), Broadband (a seismology application), and Epigenomics (a bioinformatics application). Montage is an application with a high I/O usage but low memory and CPU one. Broadband requires a high memory usage but medium I/O and CPU ones. While Epigenomics is considered as a CPU-bound application with low requirement for I/O and medium memory usage. Juve et al. [92] ran experiments on a popular, widely-used and stable Amazon EC2 and a typical HPC system NCSA's Abe cluster. The findings presented that Amazon EC2 performance was not similar to the Abe one, however, it was reasonable enough with a small (1-10%) virtualization overhead. These findings indicated that clouds could serve as a viable alternative to HPC systems if cloud service providers offer high speed networks and parallel file systems.

Other studies revealed the feasibility of running cloud-based frameworks for multi-scale data analysis while preserving the performance and storage capabilities of grids [110, 172]. For example, hydrology domain could serve as an illustration of the cloud-based framework feasibility and similarity to HPC performance level [39]. A wide range of hydrological problems has been proved suitable for the execution in hybrid computing infrastructures integrating grids with external cloud providers. This covers both computationally intensive HPC simulators and MTC-based applications with multiple scenarios. There is also a strong need for scientific applications, which can execute a non-trivial task in parallel on distributed data sets [124]. Cloud computing provides a scalable and cost-efficient solution to the big data challenge (for example, MapReduce). Jackson et al. [87] examined the performance of a typical HPC workload executed on Amazon EC2 and proved in their experiments a strong correlation between time and amount of communication and the overall application perfor-

mance on Amazon EC2 – "the more communication there is, the worse performance becomes".

Overall, Section 2.3 presented the evaluation of cloud computing potential for HPC/HTC-based scientific applications and the overview of related works in this field.

2.4 Summary

In this chapter, the state of the art in HPC and cloud computing was described. An overview of the current state, trends, and challenges for both paradigms was provided in Subsections 2.1 and 2.2 respectively. While in Subsection 2.3, the overview of related works was provided to evaluate the feasibility and suitability of cloud computing paradigm to scientific applications. Considering how demanding in terms of computing power and efficiency scientific applications are, the application of cloud computing to scientific applications must be studied in more detail. In the next chapter, we will overview scientific high performance simulators, analyze the key features of HPC and cloud execution environments, and evaluate the cloudification potential of the simulators.

Chapter 3

Modern high performance simulators

3.1 Introduction

Nelson Goodman [74] famously observed that only a few other terms are used in scientific discourse more promiscuously than "model". The same might be said about the simulation, which became used as a positive term in science after the World War II and gained its current definition: "the technique of imitating the behavior of some situation or process ... by means of a suitably analogous situation or apparatus, especially for the purpose of study or personnel training" [96]. The rise of simulation in post WWII science is not exclusively associated with the military history or the advent of the computer. Though, it is difficult to imagine the construction of the first hydrogen bomb and then its successful test in 1952 without the use of computer simulation [133]. Specifically, the introduction of the computer provided the impetus for the simulation adoption in different scientific disciplines, e.g., meteorology, nuclear physics, biology, geology, hydrology, economics, sociology.

The term "simulation" is often used interchangeably with the term "computer simulation". A modern simulation could be represented by a computer program built to explore and analyze a mathematical model of either a real-world system or a hypothetical one. Primarily, a simulation relies upon a pre-defined algorithm that takes as an input a specific state of the system at a concrete time slot t and calculates its state at the next time slot $t + 1$ under certain conditions. Prior to the computer, the study of complex, non-linear systems has been limited in computations and could only be achieved by perturbation methods, simplifying models/approximations, or paper schemes for numerical approximation [96]. Even though computers are not capable of providing researchers and scientists with the exact solution to the studied complex problem, they could definitely approximate the solution with a high degree of accuracy in a reasonably short period of time. Specifically, the accuracy and speed of scientific computations is tightly interconnected with the exponential growth of computing power. According to Denning et al. [49], this growth could be considered as a unique phenomenon continuously stimulating economic, social, and political disruptions. The performance of computers is doubling every 18 months (as claimed by David House) or 24 months (as claimed by Gordon Earle Moore) [49]. The exponential growth of computing power occurs not only at the chip level, when the number of chip components – transistors, resistors, and capacitors – double every two years according to the Moore's law, but also at the system and market levels. For the last five decades, Moore's Law proved to be sustainable. Regarding the analyses conducted by Denning et al. [49], the exponential growth at all three levels of the computing ecosystem is likely to continue for decades to come too. During the continuance of the available computing power growth, scientists and researchers have gained the opportunity to investigate more complex problems, simulate compound systems, and

describe more sophisticated interactions between these systems and processes by building and deploying advanced interconnected computer models. Although to realize this opportunity to its full extent, researchers require a big variety of sophisticated tools.

Scientific workflows (SWFs) emerged as one of the tools directed at integrating, structuring, and orchestrating various heterogeneous systems, services, and components of scientific computations to enable and accelerate the pace of scientific progress [72, 105, 155]. Lin et al. [105] defined SWFs as a scientific process formalization that represents, structures, and automates the workflow steps from dataset selection to computation, from computation to data analysis and final data representation. The scientific process formalization demands a new system, which can ease the process of developing scientific workflows, and provide adequate capabilities to modify, run, monitor, and control the execution of the workflows. A scientific workflow management system (SWfMS) emerged in response to this demand. Recent years have been marked with a considerable development of SWfMS, for example, Taverna, Kepler, Pegasus [137]. Pegasus is a framework that maps complex scientific workflows to the underlying distributed resources, specifically grid computing resources. Pegasus can accommodate different scheduling and replica selection algorithms and provide partition-level failure recovery [47]. Taverna is directed at creating and running workflows of bioinformatics experiments. Taverna builds a workflow by using Scufl language and represents it in a form of a graph consisting of processes, which transforms input data into output data [125]. Kepler claims to be unique because it provides the design of scientific workflows on the highest abstract level, execution, runtime interaction, access to data, and service invocation. Basically, Kepler has such features as an intuitive graphical user interface (GUI) to design a workflow and an actor-oriented paradigm to prototype, execute, and reuse workflows. Kepler workflows could be presented in XML, while Kepler worker can be executed as Java threads [18].

Considering three presented examples, SWfMSs can be used as a convenient tool for researchers to build and test scientific workflows without spending too much time on the technical aspects of the execution (e.g., implementation, optimization, and resource control). However, before being conveniently used, each SWfMS requires an installation and a proper configuration in a specific execution environment. In itself, that might be a nontrivial task for a non-computer scientist. Considering the distributed nature of compute environments, scientists can run programs on different hardware platforms. Because the tasks shall be properly assigned to these various resources, scientists may look for the SWfMS flexibility to adapt scheduling techniques and find a suitable match between workflow tasks and computing capabilities. However, neither automatic nor manual workflow model modification during runtime is an easy task for a non-computer scientist, because it should be carefully designed to handle exceptions for expected errors. Moreover, some scientific workflows, e.g., Montage, consist of a large number of short-running tasks [26]. In distributed infrastructures, these tasks can introduce a runtime overhead due to network latency. To cope with this issue, the user might need to manually implement certain workflow optimizations, which might be also a challenging task. Normally, SWfMSs aim to comply with various requirements like data-driven, advanced data handling, flexibility, monitoring, reproducibility, robustness, and scalability. However, there are specific requirements for every scientific domain, e.g., life science, chemistry, computer science, physics. Altogether, it becomes hard to cover all the key and domain-specific requirements in one SWfMS, consequently, there will always be a need to modify or extend a workflow model.

Summarizing advantages and drawbacks of a typical scientific workflow management system, it is important to highlight that a SWfMS normally offers a user-friendly and intuitive interface to quickly build a complex multistep workflow. Moreover, a SWfMS does not require

from researchers and scientists a profound technical background or even a knowledge of a specific programming language. Overall, a SWfMS is well suited for prototyping, testing ideas, and sharing workflows with others. However, a SWfMS has a number of drawbacks. It can perform relatively well when dealing with simple and small workflows but it faces considerable challenges to perform big and complex workflows as they might require low-level optimizations, custom scheduling techniques, and specific execution environments. Another considerable disadvantage is related to migration of the execution environment to a different infrastructure as both a scientific workflow and a SWfMS environment shall be migrated. Moreover, any SWfMS comes with additional overhead as it also consumes resources.

Considering the disadvantages presented above, an alternative to using a SWfMS could be to build a standalone application, which is capable of executing both simple and complex scientific workflows. An example of such an application could be a simulator as it normally represents a self-contained application, which is installed in a computing environment and dedicated to executing a specific scientific workflow. Regardless some possible library dependencies, a simulator does not require any additional layer for its execution. In comparison to SWfMS, a simulator is often optimized to efficiently execute a workflow of any complexity, though it normally contains an implementation of only one single workflow at a time. Also, in order to develop a simulator, a researcher shall have an extensive programming knowledge in order to apply all necessary programming techniques, different optimization strategies or algorithms.

In general, both a scientific workflow management system and simulator are sophisticated tools directed at execution of scientific workflows and production of simulation results. While a SWfMS is more workflow-oriented as it allows a researcher to build as many workflows as needed to test ideas, a simulator is more result-oriented as it allows a researcher to simply execute the workflow and get the execution result.

3.2 Problem definition

Historically, high performance computing (HPC) environments are considered as the main execution environment for scientific applications. Universities and research centers often maintain local HPC infrastructures, i.e., clusters. While local infrastructures could hold substantial computing power and capabilities, researchers may still reach the limit of available resources. Consequently, they can confront a challenge of accommodating large volumes of data or scheduling the required number of jobs.

The first concern we identified here is the limitation of computing power and resources for scientific applications.

One strategy how to deal with this concern is to use a data center big enough for any computing burst. However, this strategy is of a high cost. Another strategy is to off-load local HPC infrastructure to remote cloud-based resources on demand. Although, the migration is a complicated task due to significant architectural differences, software and hardware incompatibilities. Moreover, scientific applications are dependency-rich and may rely on external libraries.

Hence, we identified here the second concern to be a migration of scientific applications to clouds regardless environments' differences and all existing incompatibilities.

Considering "Laws of Software Evolution" postulated by Prof. Manny Lehman [102] in 1996, the adaptation of scientific applications to emerging technologies can be subject to Lehman's laws, specifically to:

1. **Law of continuing change** indicates that a software becomes less appealing to its users over time, when it does not adapt recurrently to arising needs.
2. **Law of declining quality** indicates that a software can be perceived as declining in quality or as a legacy application over some time if it is not adapted to a changing execution environment.
3. **Law of feedback system** states that to evolve successfully, a software shall be recognized as a multi-loop, multi-agent, multi-level feedback system. This law clearly indicates the importance of a user feedback for a software evolution, considering how difficult it becomes to change or improve software in some period of time due to the complexity of its artifacts, processes, and agents' interaction involved in a software change.

So far, there is a straightforward evidence of the law of continuing change as new sophisticated technologies – e.g., clouds, big data, service-oriented technologies – evolve every day, rapidly enter the market, and gain momentum because they could satisfy the current users' needs. However, many of the current scientific applications face the problem of adapting to new operational constraints (described well in the law of declining quality) as well as a difficulty of considering and adopting the user's feedback (described well in the law of feedback system). Specifically, such systems could be recognized as legacy applications.

Al Belushi et al. [15] defined legacy applications as the ones that were built without incorporating new technologies and resisting adaptations to that technologies. In general, legacy applications can be reliable, secure, and widely used because they are well tested and reach current objectives. It is important to emphasize that legacy applications are mostly tightly-coupled to a specific platform or programs. Hence, their integration with other heterogeneous software can be challenging, in its turn that can substantially limit the number of users and further scalability. This is certainly true in case of numerous scientific applications, e.g., simulators.

Therefore, we consider the third concern to be the resistance of scientific applications to adaptation to new technologies.

Overall, the academia could substantially benefit from the adaptation of legacy scientific applications to clouds. Because a cloud could provide researchers and scientists with fault-tolerance, scalability, reliability, and more elastic infrastructure. Generally, cloud applications are intended to operate on a large scale. Moreover, the cost model is straightforward as scientists can access massive computing resources and pay only for what they use. Considering the aforementioned concerns, we identify the key problem to be solved in this thesis as follows:

How can we successfully migrate cluster-based scientific applications to clouds, when these applications are built without incorporating new technologies and resist adaptations to that technologies?

Hence, the major goal of this thesis is to provide a cloudification solution for adapting scientific applications to continuous changes of an execution environment. To achieve this goal, I propose to analyze the main differences between computing environments, specifically, HPC and clouds, define common types of scientific problems, elaborate a solution, i.e., cloudification strategies, apply the solution to a common representative, and evaluate the final results.

3.3 Towards the paradigm shift

Historically, simulators are HPC-based. From the 1970s, HPC market offered researchers and scientists high performance, parallelization, and scalability. Over time, building a simulator for HPC environments became more of a tradition than a necessity. Even though the HPC environment is well developed and well studied, there is still always a demand for even bigger computing power. In this regard, some simulation problems could easily benefit from cloud computing and boost the research in certain domains due to the increased scale of simulations.

Considering this potential, in the following subsection, two computing environments, i.e., HPC and cloud, will be compared and analyzed based on their key features. Then the simulation problem types will be analyzed and classified in Subsection 3.3.2. Based on these analyses, common aspects of modern simulators will be presented and aligned with the most beneficial computing environment in Subsection 3.3.3.

3.3.1 Analysis of the key features of execution environments

In Table 3.1, the most distinct and relevant features that are generally found in two computing environments, i.e., cluster and cloud, are presented.

	Cluster	**Cloud**
Network Connection	Low Latency	Medium Latency
Type of Resources	Homogeneous	Heterogeneous
Scalability	Fixed	High
Storage	Parallel FS	Object Storage, Local Storage, Network FS
Provisioning	Physical Deployment	Cloud Provider
Virtualization Overhead	No	Yes
Dominant Paradigm	MPI	MapReduce, Service Oriented

Table 3.1: Comparison of the key features generally found in two computing environments.

Before going into the classification of simulation problem types, it is important to analyze what types of computational problems fit best each computing infrastructure. The ability to parallelize a computational problem is tightly interconnected with the decomposition of the original problem, specifically the degree of sub-problems coupling. The degree of coupling is defined by the amount of dependencies between sub-problems, and varies between loose-coupling and tight-coupling. In general, the more interconnections and interdependencies there are between sub-problems, the stronger/tighter coupling is likely to be. A good example of a loosely-coupled process could be an asynchronous communication between two systems, when a system A sends a message to a system B. The system B is offline, it gets the message when it comes online, processes it, and sends a response. Meanwhile, the system A does not wait for an immediate response and continues performing other tasks. As both systems do not depend on each other, they continue their operations without being blocked. According to Yourdon et al. [176], there are four key factors that could increase or decrease coupling, i.e., the type of connection (minimally connected systems are loosely-coupled), the complexity of the interface (the more complex a connection is, the higher the coupling), the type of information flows (data, control, or hybrid), and the binding time (time of fixing values of system identifiers).

Clusters rely on low-latency networks, which connect the distributed computing resources, homogeneous hardware and a high performance parallel file system. Hence, clusters are perfectly capable of solving any kinds of problems from loosely- to tightly-coupled. However, the fixed size of the infrastructure might significantly limit the scale of the experiments to be executed. In other words, even if a problem is loosely-coupled and able to effectively use more computing resources, the user is always bound to the amount of physically available resources. On the other hand, clouds provide the consumer with potentially unlimited amount of computing resources on-demand, which should theoretically remove the barrier for the scale of experiments to be executed. However, the communication may become expensive due to the physical distribution of data centers, regular-speed network connections between them, and little to no control over the actual location of leased resources, especially in public clouds. Moreover, the virtualization overhead and low performance network storage may significantly influence the performance of a distributed application.

This subsection provided a comparison of key features generally found in the two computing environments, i.e., cluster and cloud, classifies computational problems based on the degree of the original problem decomposition, and matched problem types with the appropriate computing infrastructure. In the next subsection, the simulation problem types will be analyzed and classified.

3.3.2 Classification of the simulation problem types

There are four main types of simulation problems defined in the literature, while only two of them (i.e., equation-based and agent-based simulations) are widely adopted and well studied as they are the most directly related to the laws of physics and nature. The other two (i.e., multiscale and Monte Carlo simulations) require the usage of more advanced methods and techniques, hence, there is still room for further studies [28, 130, 151].

Equation-based simulations

The equation-based simulation adopts a model consisting of a set of equations, which define and govern the entire system [130]. Equation-based simulations could be of two types, either particle-based or field-based. The former type adopts a model with n-number of particles and a set of differential equations governing the interactions among particles, e.g., a simulation of a galaxy formation [178]. While, the latter type is based on a model with a set of equations governing the evolution time of a field, e.g., a simulation of a meteorological system, a tsunami [178].

Figure 3.1 depicts an example of the inter-process communication within an abstract equation-based simulation. Normally, this type of simulations is not very suited for parallelization as the system must be treated as a whole. Even if the main problem domain can be decomposed and the governing equations are applied to its sub-domains in parallel, the reverse process is not trivial. The process of merging sub-domains into the original problem domain might involve a significant computational and communicational effort.

Agent-based simulations

An agent-based simulation adopts a model consisting of a number of agents. In general, agents are autonomous to interact with each other and their environment regarding a set of decision-making rules, so that the higher-level system behavior can be observed. A simple

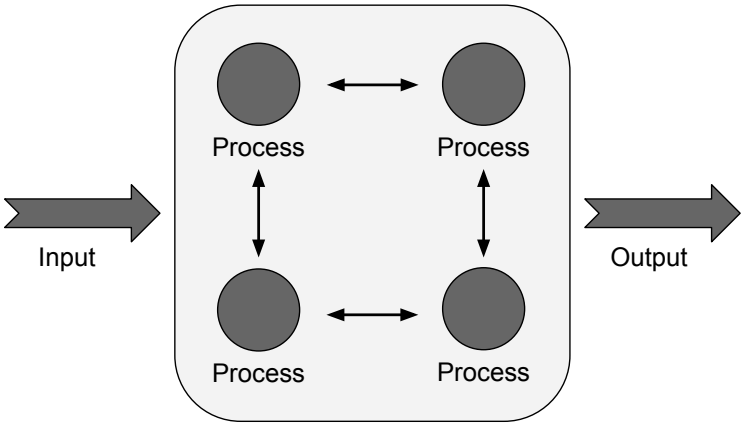


Figure 3.1: An example inter-process communication within an abstract equation-based simulation.

example of agent-based modelling is cellular automata, which comprises a grid of cells that can be in one of discrete states governed by transition rules. Each cell can have an influence on neighboring cells, thus emerging behavioral patterns can be observed. Moreover, there are some similarities between the agent- and particle-based simulations considering the behavior of n-number of individuals. However, the agent-based simulation is considerably different, because it is not governed by global system-level differential equations [130, 151, 178].

Figure 3.2 depicts an example of the inter-process communication within an abstract agent-based simulation. This type of simulations might be well suited for parallelization as each agent represents a distinct element of a system with its own governing rules. However, such simulations often imply a large amount of communication between agents. Therefore, there is a considerable demand for high performance communication channels, otherwise it might easily become a bottleneck for the entire system.

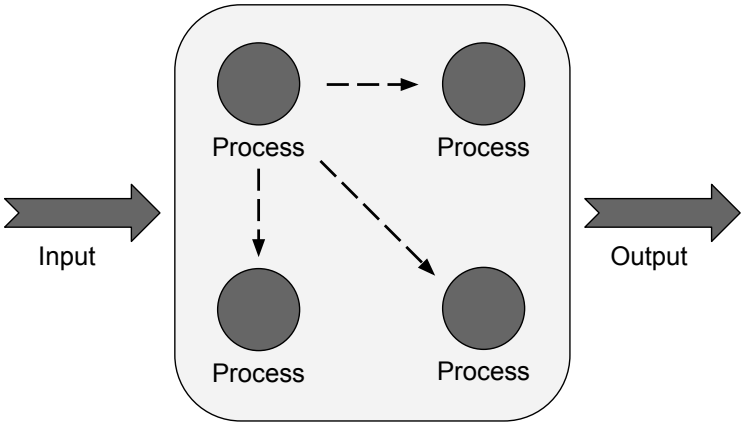


Figure 3.2: An example inter-process communication within an abstract agent-based simulation.

Multiscale simulations

While the two previous types (equation-based and agent-based) are typical and based on one certain method, a multiscale simulation model is a hybrid of multiple modeling methods at different scales. A good example of a multi-scale simulation is a model that treats a

certain material (e.g., fluids, solids, polymers) like a field, simulates this field under some conditions (e.g., stress, strain), decompose it into distinguishable micro-elements, and models the elements, where important small-scale effects take place. A multiscale simulation is well suited to treat two types of problems. The first problem relates to notable events happening locally, e.g., chemical reactions, singularities. To approach this problem, there is a need to model the local behavior as well as to model the entire system dynamics under this event. The second problem correlates with the lack of constitutive information in the entire system model. To approach this problem, the missing information shall be gained through a microscale (i.e., a certain system element) modeling [81, 167].

Figure 3.3 depicts an example of the inter-process communication within an abstract multiscale simulation. The ability to properly parallelize this simulation problems fully depends on the actual sub-simulations at different scales and their interconnections. In some cases, such problems might be perfectly parallelized by initially running a fast and low-precision model of the entire problem domain, and then executing long-lasting high precision models only of the specific sub-domains in parallel. In other cases, the simulation might involve a set of sequential workflow steps to properly treat the transitions between the low-precision and high precision models.

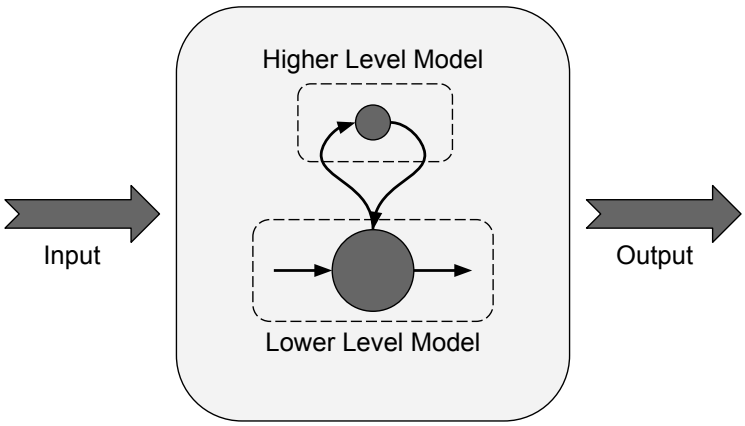


Figure 3.3: An example inter-process communication within an abstract multiscale simulation.

Monte Carlo simulations

Another type of problems is a Monte Carlo simulation. It covers dynamic systems with random processes that could not be modeled but only predicted by the values of statistics and probability. In Monte Carlo simulations, randomness is a mechanism to obtain numerical results via repeated execution of a single sampling procedure provided with random parameters and/or random time-function inputs. A simple example could be a calculation of the number π by simulating random objects' dropping in a square with the circle inscribed in it. The number of objects landed inside the circle in proportion to square will be $\pi/4$ [178].

Figure 3.1 depicts an example of the inter-process communication within an abstract Monte Carlo simulation. A distinct feature of all Monte Carlo simulations is that they consist of a large number of identical but completely independent processes, which can be trivially parallelized and distributed over any kind of computing infrastructure.

In this section, four major types of simulation problems were presented. In summary, both equation- and agent-based simulations are two widely-adopted approaches, which simulate

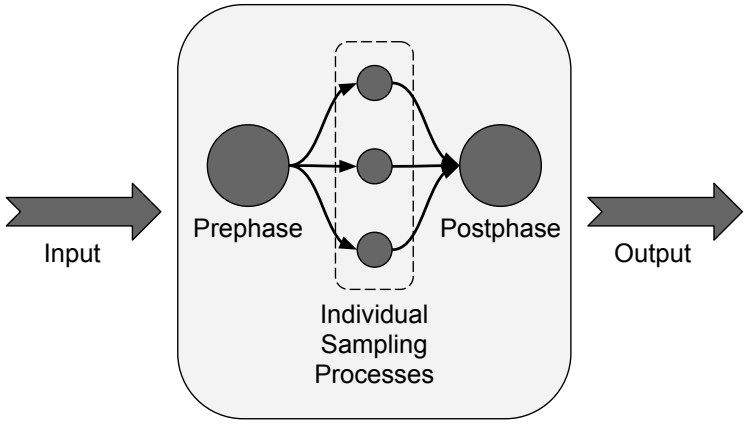


Figure 3.4: An example inter-process communication within an abstract Monte Carlo simulation.

the system by constructing a model consisting of, respectively, either equations or agents and executing it on a computer. A multiscale simulation represents a combination of multiple models at different scales. Whereas, a Monte Carlo simulation is a unique approach that covers distinct dynamic systems with random processes that could be only predicted by the values of statistics and probability.

3.3.3 Common aspects of modern simulators

When the simulation problem types are defined, now we can develop a generic cloudification methodology, which will allow us to adapt and migrate legacy scientific applications to clouds and gain maximum results from that migration. However, it is important to mention that the internal organization, technological choices, and implementation of existing workflows might considerably differ. Therefore, the cost and effort of migrating scientific applications to clouds might be equal to reimplementing the workflow from scratch. Hence, to avoid this situation, we propose to look at the internals (i.e., complexity and parallelization layers, internal communication patterns, and programming models) of existing simulators in order to understand better potential challenges of the migration strategy.

3.3.3.1 Complexity and parallelization layers

As mentioned in Section 3.1, simulators have become a key tool for scientists working with complex systems and multiphysics models. Nowadays, the computational complexity of these models has increased notably, threatening the scalability of the addressable simulation size, number of runnable scenarios, and time required to obtain the results. The complexity is even more aggravated by a continuously increasing amount of input data originating from geographically distributed sources like sensors, radars or cameras.

Every workflow has diverse resource requirements and constraints. Workflows can vary in their characteristics, e.g., size, constraints, resource usage, structural patterns, data patterns, and usage scenarios. Altogether, these characteristics signify the level of complexity the workflow may have. Ramakrishnan et al. [139] classified workflows regarding their characteristics and overall complexity and presented examples from different scientific domains: bioinformatics and biomedicine, astronomy and neutron, weather and ocean modeling. It is

important to emphasize that the workflow complexity reached a level when there is a need to run several additional sub-workflows in order to execute one key workflow.

For example, the Linked Environments for Atmospheric Discovery (LEAD) workflow is directed at supporting dynamic and adaptive response to severe weather conditions. The workflow's major inputs are real-time observational data and static terrain data. While the workflow outputs are the results of data mining post-processing. Overall, LEAD's workflow requires the execution of two sub-workflows: weather forecasting and data post-processing by a data mining component, which can also trigger the execution of two supplementary forecast workflows or even steer remote radars for extra localized data [139].

Another example can be taken from molecular sciences. In the drug-design process, it is important to understand the three-dimensional structure of protein and ligand atoms. The Gemstone is an interface consisting of a set of tools that allow users to analyze and visualize these atomic structures. The Gemstone workflow consists of a set of sub-workflows launched manually in a sequential way. The first stage of Gemstone workflow is to download data from the Protein Data Bank database and convert it into the required format. Then data is preprocessed concurrently by Babel and LigPrep services. Finally, the analysis of the preprocessed data is conducted by GAMESS and APBS. In the end, the results are visualized by using QMView service [139].

We can see the dependencies of the workflows described above as a layered structure, where the lowest layer covers simple mathematical operations like matrix operations, and the highest layer comprises a complex multi-physics simulation problem. Actually, the computational infrastructure can be perceived as a layered structure as well, where the lowest layer consists of local resources like CPU, and the highest layer includes a distributed environment like clouds (Figure 3.5). Of course each layer has its own characteristics, which should be taken into account in the application development process. Ideally, a problem should have multiple parallelization layers. If each layer is properly parallelized, then the highest efficiency and resource utilization can be achieved.

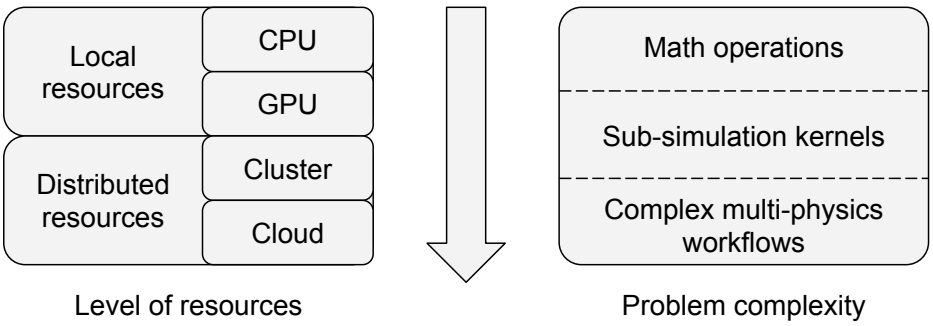


Figure 3.5: Infrastructure Parallelization Layers and Problem Parallelization Layers

3.3.3.2 Internal communication patterns

Every computational problem consists of interconnected algorithms, which work at different layers. Altogether, these algorithms form the final workflow. Considering algorithms' interdependency and the layered structure of a workflow, it is possible to abstract communication between workflow elements into a communication pattern. To define a pattern, it is worth to cite Christopher Alexander [17] who coined a "language pattern" term: "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a

million times over, without ever doing it the same way twice". The workflow patterns have been a widely investigated topic both in academia and industry interested in formalizing the behavior of business processeses [45, 69, 139, 159, 169]. Gamma et al. [69] proposed a set of 23 design patterns in the field of object-oriented technologies that could be categorized into 3 groups: creational, structural, and behavioral patterns. For the Business Process Management community, van der Aalst et al. [159] elaborated a collection of control-flow patterns that could be categorized into 6 groups: basic control flow patterns, advanced branching and synchronization patterns, structural patterns, multiple instance patterns, state-based patterns and cancellation patterns. Certain authors offered only basic patterns [45, 139], while other authors also examined advanced patterns not supported by today's generation of workflow management systems [158].

Here we present 7 basic workflow patterns (Figure 3.6):

1. Sequential pattern

   The sequential pattern can be described as a sequence of workflow tasks following one another, when each new task is enabled only after completion of the previous task. The sequential pattern is implemented to design consecutive steps in a workflow process.

2. Iterative pattern

   In the iterative pattern, a workflow task is repeated multiple times but the parameters of each new iteration are based on the results of the previous iteration.

3. Parallel-split pattern

   The parallel-split pattern consists of one task that splits itself into multiple tasks executed in parallel. The results of the first single task serves as an input for the next parallel tasks.

4. Parallel pattern

   The parallel pattern consists of multiple tasks that could be executed independently. Such pattern appears due to a specific logic of a workflow (e.g., parallel-split) or an input data that arrived to all the tasks at the same time.

5. Parallel-merge pattern

   The parallel-merge pattern consists of multiple tasks executed in parallel and then merged into one task. The parallel tasks might differ in parameters or/and operations performed but when the tasks are completed, the results are fed into the next single task.

6. Dynamic pattern

   In the dynamic pattern, a task might produce additional tasks at a runtime under certain conditions. A priori it is unknown whether and how many new additional tasks might be produced.

7. Multi-stage pattern

   The multi-stage pattern consists of multiple tasks running in parallel, however there are certain synchronization points when these tasks stop their execution, exchange data, and then resume.

This basic classification is the most relevant to scientific workflows as it provides only a high level view on the workflow sub-processes and clearly shows their interdependencies without over-complicating them [45, 139].

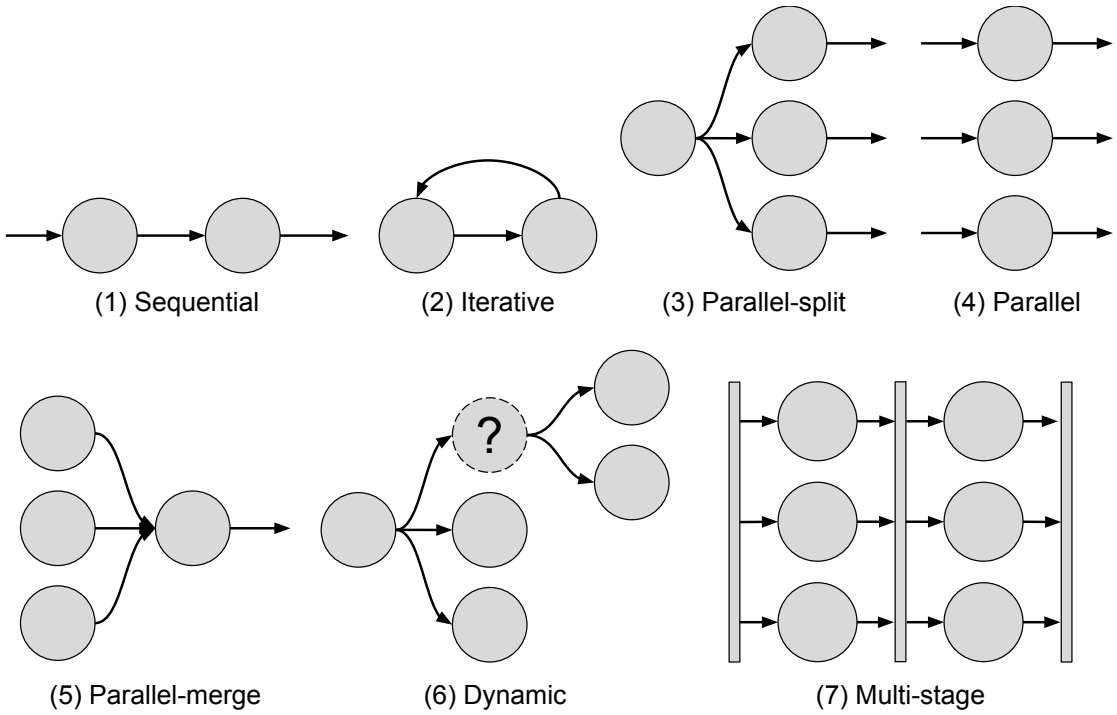


Figure 3.6: Basic Workflow Patterns

3.3.3.3 Programming model

According to Subsection 2.1.2, there is a number of key HPC programming models, i.e., MPI, HPF, OpenMP. Among all of them, MPI is mostly recognized as a de facto standard for HPC applications. It is important to underline that the MPI-based application performance might be significantly affected by different factors like network latency and bandwidth characteristics, file system performance, and homogeneity of computing resources. To be most effective, MPI requires a low-latency broadband network (e.g., InfiniBand, Myrinet 10G) to exchange data by sending and receiving messages uninterruptedly, a high performance parallel file system (e.g., Lustre) to share large files across multiple nodes, and homogeneous computing resources to achieve higher resource utilization.

Considering Table 3.1 on Page 27, these requirements are most relevant to clusters. Furthermore, it is worth to mention that MPI lacks integrated fault-tolerance and elasticity. Moreover, it cannot adapt dynamically to infrastructure changes. Altogether this makes MPI not the best technological choice for clouds, hence, running MPI-based applications directly in cloud environment might significantly hurt the performance. To prove this statement, two practical evaluations of the performance drop will be presented in Subsections 4.4.3 and 5.4.3. Hence, the cloud execution of MPI-based applications might require certain changes and possible redesign of applications or some of their components.

Overall, this subsection overviewed the computational complexity, parallelization layers, internal communication patterns, and programming models of modern simulators. Considering the classification of the simulation problems described in Subsection 3.3.2, we propose to concentrate on the Monte Carlo simulators. These simulators may considerably vary, because they cover a wide range of scientific problems [36, 103, 132, 163] from distinct domains, i.e., physics, chemistry, biology, computer science, electronic engineering. Although, all these simulators have one particular similarity – they are based on the Monte Carlo method. Monte

Carlo simulators use the same computational algorithm. It requires a large random sampling to determine the properties of some phenomenon or system behavior. An example of such a simulator can be the EnKF-HGS simulator [101], which will be presented in the following Section 3.4.

3.4 Case study: hydrological ensemble Kalman filter simulator (EnKF-HGS)

In the domain of hydrology, recent technological and mathematical advances allow researchers to significantly improve the precision of simulations by integrating measurement data in the modeling process [25]. One of the state of the art simulators for this purpose is the compute- and data-intensive EnKF-HGS simulator [101], which was originally developed for execution on clusters. EnKF-HGS performs real-time stochastic simulations, which are continuously improved by assimilating the most recent field measurements, with the possibility for additional real-time control of water resource systems. EnKF-HGS introduces a high computational demand due to the high nonlinearity of the simulated processes. Moreover, the adaptation of the EnKF-HGS simulator to new technologies is impeded by its two proprietary simulation kernels. Therefore, we consider EnKF-HGS to be a common representative of scientific applications, which has a continuously increasing demand for computing power but faces challenges in migration to clouds.

In this section, we will present the EnKF-HGS simulator in more detail and use it as a test scientific application throughout the rest of the thesis.

3.4.1 Simulator description

Data assimilation is the process of incorporating observations of an actual system into the model state of a numerical model of that system. In the EnKF-HGS simulator, it is done via the ensemble Kalman filter [30, 58]. The ensemble Kalman filter was first adopted by Evensen [58] in 1994, since then it was widely examined, applied in different studies, and gained popularity due to its simple conceptual formulation and relative ease of implementation [59]. Moreover, its computational requirements are easily accessible and comparable with other popular sophisticated assimilation methods, e.g., the 4DVAR method [109], the representer method by Bennett [118]. The ensemble Kalman filter is used to effectively merge uncertain model predictions with uncertain observation data in a Bayesian sense. The uncertainty of model predictions is approximated through the forward simulation of an ensemble of model realisations, where each realisation can have a different combination of initial conditions, model forcings and model parameters.

The model state vector $\mathbf{x}$ for each model realisation i at time step t (where observations are available) is derived by forward propagation of the dynamical model M using as input the state vector from the previous time step $(t-1)$ and parameters $\mathbf{p}$ (e.g., hydraulic conductivity, porosity) and hydraulic forcings $\mathbf{q}$ (e.g., water level in a river). Parameters and forcings are different for each model realisation:

$$\mathbf{x}_i^t = M(\mathbf{x}_i^{t-1}, \mathbf{p}_i, \mathbf{q}_i) \tag{3.1}$$

First, we would like to overview the integrated hydrological modelling software HydroGeoSphere (HGS) and its numerical model (see 3.4.1.1). Then we will present advantages of

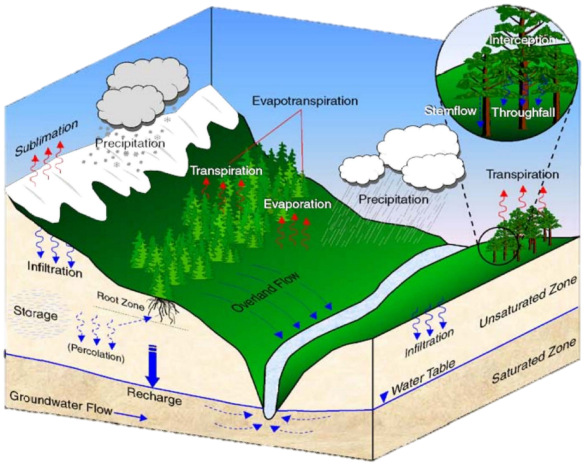


Figure 3.7: Typical surface water and groundwater processes in a pre-alpine type of valleys [93].

the ensemble Kalman filter (see 3.4.1.2). In the end of this subsection, the challenges of this method will be described (see 3.4.1.3).

3.4.1.1 Numerical model

The latest generation of numerical models is able to simulate the interactions between surface water and groundwater in a fully coupled [29]. One of the most advanced codes in this respect is HydroGeoSphere (HGS) [153]. HGS is a numerically demanding code that implements a 3D control-volume finite element hydrology model with a fully integrated surface-subsurface waterflow and thermal energy transport. It provides functionality for dynamic stochastic simulations between water profiles composed of numerous elements, as depicted in Figure 3.7. In addition to simulating surface water and groundwater interactions, the code can also simulate vegetation dynamics as well as the recharge[1] and discharge processes in response to precipitation, evapotranspiration or groundwater abstraction [129, 145].

HGS uses the control volume finite element method to solve flow equations for all domains considered in a simulation. Non-linear equations are solved by means of the Newton-Raphson linearisation method, and the matrix equation arising from the discretization is solved by a preconditioned iterative solver. As shown in Eq. 3.2, the execution control is conducted with a variable time-stepping procedure. This is defined in HGS according to the rate of change of a solution variable (e.g., flow, temperature), so that the simulation uses increasingly larger time steps to reduce the number of steps towards convergence, and thus the computing time, if the dependent variable, V, does not experience drastic changes in several iterations.

$$\Delta_t^{k+1} = \frac{V_{max}}{max|V_i^{k+1} - V_i^k|}\Delta_t^k \tag{3.2}$$

3.4.1.2 Data assimilation vs. "classical" model calibration

Data assimilation is the process of adjusting the simulated numerical model by continuously incorporating newly available environmental field measurements. Normally, the geometric setup is based on a high resolution digital terrain model. The numerical coupling between the

[1] water infiltrating the soil reaching the underground water table

surface and subsurface domains in HGS is conceptualized through a dual node approach, as described by Therrien et al. [153]. The model requires a very large amount of parameters, such as hydraulic properties of the streambed, the soil or the aquifer. These parameters cannot be measured in the field at required spatial resolution. Therefore, they must be estimated. Numerous approaches are available in this regard. A "classical" way is to adjust parameters in order to minimize the mismatch between the available historical measurement data and the corresponding model simulations. Once a model reproduces historic measurement data satisfactorily, it is used to predict future system states under changing forcing functions. However, all numerical models are a simplification of reality, both in terms of the considered processes as well as in their parameterization. Therefore, any calibrated model will sooner or later deviate from the real, physical system state. Clearly, the model state (i.e., the simulated water levels or the actual discharge in the river) has to be as close as possible to the real system in order to provide reliable predictions on how a planned pumping scheme will affect the system in the near future. Therefore, the "classical" calibration approach is not well suited for this application. By using a data assimilation approach, the model is continuously updated in terms of its state and parameters that allows EnKF-HGS to minimize the deviation of the model from the simulated system.

3.4.1.3 Monte Carlo simulation

The ensemble Kalman filter is an implementation of the Monte Carlo method that consist of two distinct steps: (i) the forward propagation of the ensemble of model realisations (i.e., forward propagation phase), and (ii) an update of the simulated model state with the measurements (i.e., filtering phase). The filtering phase is a relatively short-lasting but tightly-coupled process that performs a set of matrix operations and requires multiple data synchronization points. On the other hand, the forward propagation phase comprises a large pool of independent model realizations, which introduces a tremendous demand for computing power, especially if combined with a complex numerical model such as HydroGeoSphere. On top of that, the iterative nature of the data assimilation process imposes that the two phases have to be repeated continuously, thus shifting the demand even further. Even though this method results in higher quality model predictions than the "classical" simulation methods, the high resource demand of the method remains an unsolved problem for many environmental scientists.

This makes EnKF-HGS simulator a perfect representative of a complex and compute-intensive scientific application. EnKF-HGS performs the two aforementioned steps of the ensemble Kalman filter with a couple of implementation specific auxiliary steps shown in Figure 3.8. Without a carefully designed parallelization strategy, obtaining any simulation results within a reasonable execution time would be absolutely infeasible.

In summary, this subsection described the internal functionality of the EnKF-HGS simulator, i.e., a numerical model, the process of adjusting the model to environmental changes, and the use of the Monte Carlo method in the context of water resource management. Overall, the EnKF-HGS simulator can provide higher quality predictions but at the same time it demands a large amount of computational resources. Altogether this makes the EnKF-HGS simulator a typical representative of complex scientific applications. The aim of the next subsection is to present in more detail the internals of the EnKF-HGS simulator, i.e., its implementation and execution model.

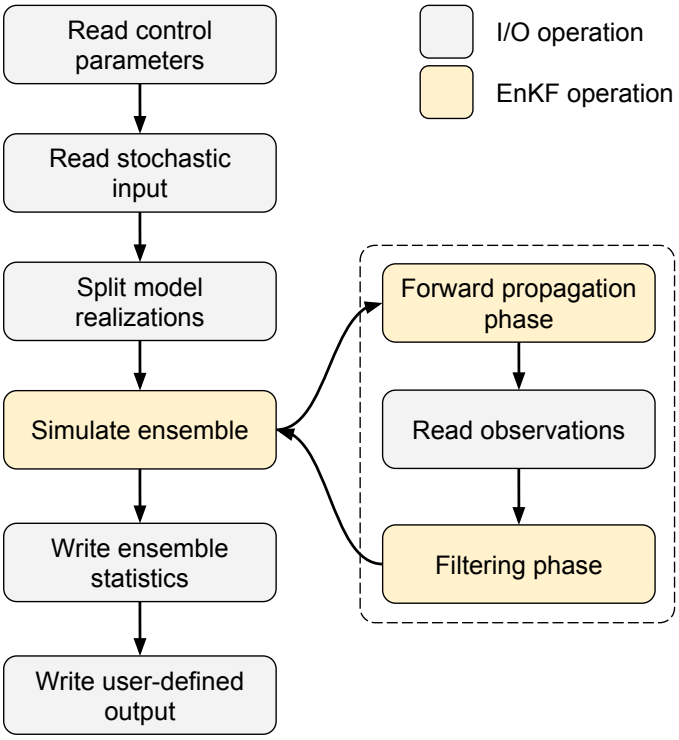


Figure 3.8: Main operations of the EnKF-HGS workflow.

3.4.2 Implementation and execution model

EnKF-HGS is a C program that uses the Message Passing Interface (MPI) framework for parallelizing the ensemble forward propagation and the filtering phases in order to speed up calculations. The parallelization is done by distributing different realisations among available CPUs, so that each CPU handles the forward propagation and filtering phases of a specific subset of the whole data assimilation problem. It is worth mentioning that one simulation of the HydroGeoSphere model comprises the sequential execution of two proprietary simulation kernels: GROK and HGS [29, 152]. Where GROK is a preprocessor that prepares the input files for HGS, which makes GROK an I/O intensive application. HGS, on the other hand, is an integrated hydrological modelling simulator, which mainly relies on the CPU to solve the aforementioned differential equations. The interfacing between EnKF and HGS is done via the input and output files of HGS.

Figure 3.9 shows the execution model of the MPI implementation of the EnKF-HGS, depicting the following stages:

Ensemble preparation stage

A. Initialization

At the beginning of the workflow, the root MPI process initializes global data structures and reads the provided model parameters from the input files.

B. Data distribution

According to the initial model parameters, the root MPI process generates input files for each model realization and stores the files in separate directories on the network storage.

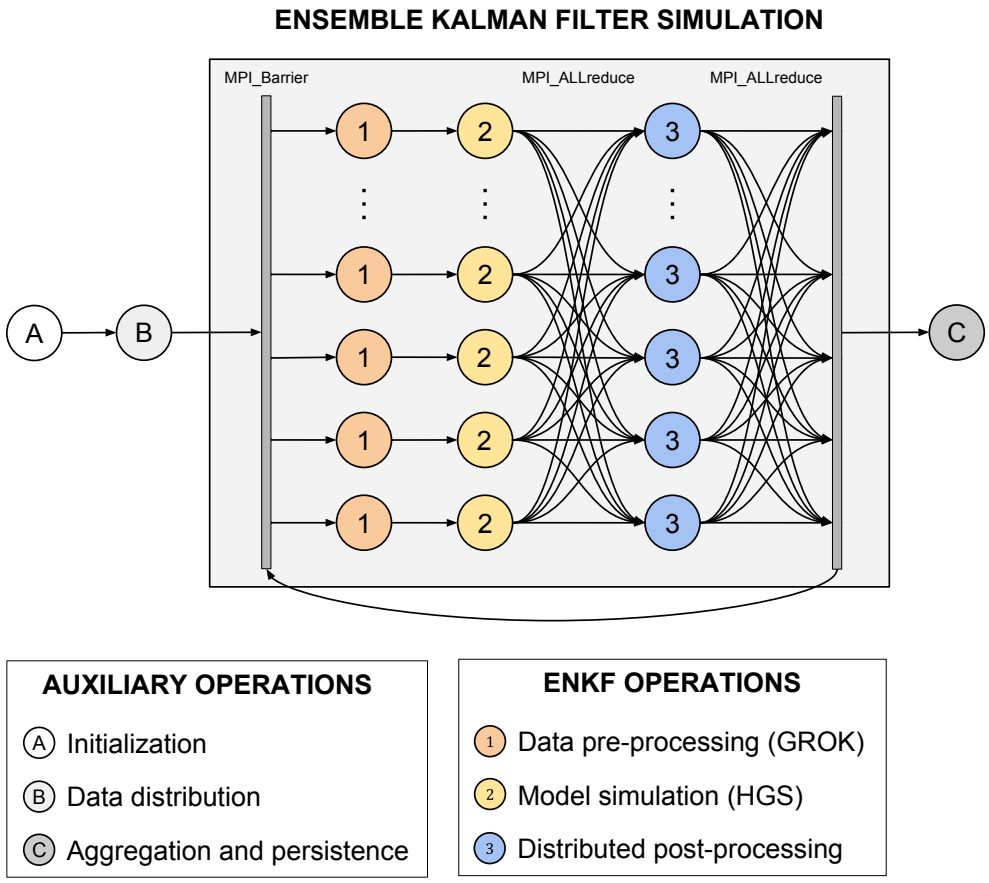


Figure 3.9: Execution model of the MPI-based implementation of EnKF-HGS.

Iterative ensemble simulation

1. **Data pre-processing (GROK)**

   After all running MPI processes reach the first synchronization point (an MPI barrier), each process execute the preprocessor GROK on the corresponding input directory in order to generate HGS-specific input files that will be written to the network file system.

2. **Model simulation (HGS)**

   HGS reads the output of GROK, runs the model realization and writes the output to the network file system. At this point, all MPI processes synchronize for the second time using an MPI all-reduce directive, since further data updates require simulation results of all model realizations.

3. **Distributed post-processing**

   During the data update process, each model realization is optimally weighted and updated with the most recent field measurements in order to reduce the simulation error. Each process is in charge of updating its block, and results are afterwards merged through another MPI all-reduce call.

Output management

C. Aggregation and persistence

After all iterations are completed and all MPI processes reach the barrier, the root MPI process aggregates the model simulation data from the realization directories and updates the global data structures. Before the program terminates, the root MPI process writes the model simulation results to the output files.

It is important to emphasize that the EnKF-HGS simulator could also be divided into two major parallelization layers with reference to Subsection 3.3.3, where the computational complexity of modern simulators and their layered structure is described. The first layer is the EnKF-HGS workflow itself. Using the classification of basic workflow patterns, this layer could be represented by a set of the following patterns: (3) parallel-split, and (4) parallel patterns are used in the forward propagation phase; (7) multi-stage pattern is used in the filtering phase; (2) iterative, (5) parallel-merge, and (1) sequential patterns connect the two phases together. The second layer is the GROK and HGS simulation kernels. Because both kernels are proprietary binaries, their parallelization can be neither analyzed nor modified, thus it is out of the scope of this thesis.

Overall, the EnKF-HGS simulator as well as any other application with such an execution model performs reasonably well in regular HPC environments due to certain important characteristics of the latter (e.g., availability of a high performance network storage, a low-latency broadband network connection). However, when moved to the cloud environment, the application performance might drop drastically (as it will be illustrated in Subsections 4.4.3 and 5.4.3). In order to be able to benefit from the advantages of clouds while maintaining the performance at an acceptable level, certain modifications of the execution model are necessary. Respectively, each cloudification methodology, which are proposed in Chapters 4 and 5, will modify the EnKF-HGS workflow accordingly (see Subsections 4.4.2 and 5.4.2).

3.5 Summary

The aim of this chapter was to overview scientific simulations, define the problem associated with the HPC-based scientific simulators, which require high computing power but face challenges in adaptation to new technologies, and present a common representative scientific simulator. In the following Chapters 4 and 5, we will present cloudification solutions to the defined problem and evaluate them by using the EnKF-HGS simulator.

Chapter 4

Big data inspired cloudification methodology

4.1 Introduction

Considering the complexity of modern scientific applications presented in Subsection 3.3.3, it is important to emphasize how convoluted and compute-intensive current simulators are. At the same time, these simulators become data-intensive too. For example, various numerical simulations in physics, chemistry, bioinformatics, and other disciplines collect, process, and store vast volumes of data necessary for experiments. All these simulations – i.e., Monte Carlo simulations, N-body solvers, molecular dynamics – require an abundance of compute hours and generate extremely large volumes of data. For example, at the Large Hadron Collider (LHC) every particle collision experiment generates petabytes of data but the Worldwide LHC Computing Grid will only be able to store, distribute, and analyze around 50 petabytes of data per year in 2017 [2]. These scientific workflows are composed of heterogeneous and coupled components that simulate different aspects of the domain they model. Their modules interact and exchange significant volumes of data at runtime, consequently, having efficient data transfers can considerably influence the overall performance of the resulting application [180]. Hence, both the storage infrastructure and logical file system abstractions affect performance and scalability, thus making data management a key aspect of workflow design and implementation [157].

Given the data-intensive nature of scientific simulations, recent studies have suggested to combine the traditional high performance computing (HPC) and grid-based approaches with big data (BD) analytics paradigm [108]. For example, typical BD analytics tools – such as MapReduce – have been considered to substitute MPI parallelism induction mechanisms, following a data-centric approach. This may substantially affect the underlying computing infrastructures. Indeed, cloud computing – a key element in current data analytics systems – could inspire hybrid platforms for exascale scientific workflows, in which storage is not completely isolated from computing nodes [142].

Following this trend, BD infrastructures and paradigms are increasingly seen as alternatives to traditional HPC approaches for some major types of scientific applications, especially those with many loosely-coupled tasks [43], or heterogeneous tasks with few interdependences [138]. The application of these mechanisms could improve scalability in parameter-based scientific simulations, as it was proved by Caíno-Lores et al. [33]. In particular, that study focused on increasing the addressable size and complexity of standalone scientific applications executed within map-reduce-based wrappers.

Caíno-Lores et al. [33] proposed to use BD tools to scale up scientific workflows. It is important to emphasize that there could be considerable differences between the analytics and scientific worlds that might require novel approaches to migrate scientific simulations to BD infrastructures. Hence, the suitability of these data-oriented mechanisms for scientific workflows shall be assessed first.

4.2 State of the art

Following this trend, big data infrastructures arose as alternatives to traditional HPC infrastructures for some major types of scientific applications, especially those with many loosely-coupled tasks or heterogeneous tasks with some interdependences [138]. For instance, cloud computing appeared as an affordable possibility to build flexible infrastructures on-demand. This is a popular paradigm that relies of resource sharing and virtualisation to provide the end user with a transparent and scalable system.

Given the benefits of cloud environments, several areas of science and industry are trying to migrate their legacy applications to clouds in order to support scalability beyond their private infrastructure. Nevertheless, it is necessary that the cloudification procedure is able to manage resources and data in such a way that scientific applications benefit from the underlying infrastructure without hurting performance and resiliency.

A key aspect in large-scale computing, especially for data-intensive applications, is data locality, defined as the minimisation and optimisation of information transmissions in order to reduce transfer latencies [173]. The degree of data locality in a distributed application has a major impact on its overall scalability, hence, we believe that cloudification must pay special attention to inter-datacenter and inter-node data locality.

4.2.1 Data analytics tools

According to Zikopoulos et al. [182], big data (BD) applies to information that cannot be processed or analyzed by using traditional data analytics tools. BD can be defined as high volume, velocity and variety of data, which require new ways of high performance processing. Supercomputing and high performance computing have been widely used by scientists to analyze large datasets and simulate mathematically complex models. With the rise of big data, standard data processing and analytics tools face considerable challenges, i.e., a growing demand for high performance, optimization of time and cost, new or additional algorithms to preprocess, analyze, and fit data into memory. In 2003, Google MapReduce arose as the first framework, which enabled large datasets processing. This data analytics tool was directed at processing and generating large datasets in an automatic and distributed way by using two major primitives *map* and *reduce*. Then Apache Hadoop emerged as the most popular open source implementation of MapReduce, which maintains the key MapReduce features. MapReduce and its well known implementations will be presented in more detail and analyzed below. While in Subsection 4.2.2, BD analytics tools will be analyzed from the perspective of their applicability to HPC environments and applications.

MapReduce

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Its first version was built by Google in February 2003, since then it was successfully tested, and officially presented to the scientific community on the

OSDI'04: Sixth Symposium on Operating System Design and Implementation in December 2004 [44]. It is a simple programming interface, which enables parallelization and distribution of large-scale computations. Even though MapReduce was primarily developed for coping with the complexity of data parallelization, distribution, and failure handling for the search engine domain, but, according to Dean et al. [44], it is also widely used for data-intensive applications, i.e., machine learning, data mining.

MapReduce programming model

MapReduce is based upon two major functional programming primitives *map* and *reduce*. First, a programmer applies the *map* function to a potentially large input dataset. Second, the *map* function produces intermediate data and passes it to the *reduce* function. The *reduce* function accepts the data, merges it, and produces a smaller output dataset. Consequently, too large lists of values could fit in memory [44]. In MapReduce, the key advantage is associated with the fact that *map* and *reduce* functions are written and executed as single nodes programs, while the subsequent data parallelization and synchronization are internally implemented without any participation from the programmer's side. In comparison to MapReduce, MPI can offer the parallel data processing but it will require a programmer to take care of the implementation of data management, parallelization, and synchronization [60].

Analysis of MapReduce implementations

Considering its popularity, it is important to present MapReduce implementations for different environments that have recently evolved.

Apache Hadoop is best known for Apache Hadoop MapReduce and its distributed file system HDFS, but also it covers many related projects within the infrastructure for distributed computing and large-scale data processing. It is one of the most popular open-source MapReduce implementations that uses distributed file system HDFS. It stores data across local disks of the computing nodes and presents through the HDFS API a single file system view. HDFS aims to deploy on unreliable commodity clusters and achieve reliability through the replication of data. To optimize data communication while executing MapReduce programs, Hadoop uses the data locality information from the HDFS file system to schedule computations near the data. Hadoop architecture consists of the master node, multiple worker nodes, and a global queue for scheduling computations. The MapReduce model mitigates data transferring bottlenecks through the *reduce* step. To achieve fault-tolerance, Hadoop duplicates executions of slower tasks. Meanwhile, Hadoop reruns the failed tasks using different workers in order to handle failures [76].

Apache Spark was originally developed by Zaharia et al. [177] in UC Berkeley's AMPLab and open sourced in 2010. They proposed Spark as a framework, which can support a class of iterative machine learning algorithms and interactive data analysis tools, which reuse a working set of data across multiple parallel operations. While supporting these applications, Spark retain the scalability and fault tolerance of MapReduce. In comparison to Apache Hadoop, Spark performance in iterative machine learning jobs can be as much as 10x higher. Spark MapReduce implementation is developed in Scala, a statically typed high level programming language designed to interact well with Java and C# environments [123], with a functional programming interface. Spark is based on

the main abstraction of a resilient distributed dataset (RDD), which represents a read-only collection of objects partitioned across machines and easily-rebuilt in case of loss. RDD is loaded in memory across a set of machines and reused in MapReduce parallel operations [177]. In comparison to Twister, an iterative MapReduce implementation, Spark is both fault-tolerant and more general, as the Spark program can define multiple RDDs and run operations on them in sequence. As a consequence of its architecture, Spark supports not only MapReduce but other programming models too.

MapReduce-MPI is an open-source framework that implements the MapReduce operation on top of standard MPI message passing. The MapReduce-MPI library is developed in C++ and could be callable by high level programming languages, i.e., C++, C, Fortran. By the use of MapReduce-MPI library a regular MPI program can be compiled, thus this framework requires no special support for HPC-based programs and HPC cluster environment in general [150]. Plimpton et al. [134] analyzed the performance of MapReduce-MPI and underlined its conceptual difference from a typical cloud computing model like Hadoop. In the MapReduce-MPI case, a programmer can potentially control which processor owns which data at various stages of computation. Whereas in a typical cloud-computing model, it is hidden from the user. However, MapReduce-MPI completely lacks fault tolerance or any failure-recovery mechanisms.

MARIANE (MapReduce Implementation Adapted for HPC Environments) is an academic MapReduce implementation developed in Java and suitable for HPC environments. Fadika et al. [61] designed and implemented MARIANE, capable of making use of parallel networked file systems (GPFS, NFS), shared-disk, POSIX and various cluster subsystems. MARIANE offers not only traditional benefits of a MapReduce framework, i.e., data synchronization, ease of programming and deployment, but also high performance under failing conditions and MapReduce applicability to a wider range of HPC-based applications. The key challenge with MARIANE is that it is a closed academic project with the implementation not widely adopted.

MARISSA (MapReduce Implementation for Streaming Science Applications) is an academic MapReduce implementation that supports any executable binary and suitable for large-scale scientific applications and data analysis workflows. Dede et al. [46] state that MARISSA can perform at the same level or even better than Hadoop and support various POSIX-compliant file systems widely adopted in scientific applications. According to Dede et al. [46], MARISSA provides an iterative support for an application to access its output and schedule further computations; the ability to run different input datasets on various nodes; the capacity for all or some of nodes to run same task duplicates and at the *reduce* stage to select a result. It is worth mentioning that MARISSA is also a closed academic project and its implementation is not widely adopted.

Phoenix is a shared memory MapReduce implementation originally developed in C/C++ at Stanford University and directed at multi-core and multiprocessor systems [141]. Phoenix uses shared memory threads to implement parallelism. Compared to Hadoop MapReduce, Phoenix workers communicate by accessing a shared address space, which results in lower communication overheads. However, a shared memory communication has also some limitations for large-scale parallel computation due to the way how threads access memory and perform I/O could considerably impact the overall performance [175].

Twister is an iterative MapReduce implementation. Its key idea is to enable a programmer to configure *map* and *reduce* only one time and then run them in as many iterations as

necessary. If there is a need for only one iteration, then the execution of a computation is identical to Hadoop MapReduce. When there is a need for two or more iterations, then the output from the *reduce* function is collected by a "combine" method and sent as a new dataset of key/values to the next iteration. Twister implementation does not provide an overall fault-tolerance but rather in certain job stages. That means that a job-specific task can be restarted but not rescued if one of the computing nodes fails [54, 60].

Amazon Elastic Map Reduce (EMR) provides a MapReduce as an on-demand service on top of the Amazon IaaS, i.e., Amazon EC2 [50]. EMR is a service that provides fully managed Apache Hadoop MapReduce framework, which uses Amazon EC2 for computing power and Amazon S3 for data storage [76]. With EMR there is no need to install and configure a Hadoop cluster. A programmer can simply execute Hadoop MapReduce computations in clouds by using a web interface and API command line. If a programmer already has an existing Hadoop program, it will require minimal changes for execution on EMR. If there is a need for jobs debugging, EMR provides options of uploading Hadoop log files into S3 and state information to SimpleDB [76]. EMR pricing model includes the cost for EC2 computing power, S3 data storage, an optional cost for the SimpleDB, and a cost per hour of EMR service usage.

AzureMapReduce is a distributed decentralized MapReduce runtime on top of the Microsoft Azure Platform. The AzureMapReduce implementation benefits from the Microsoft Azure Platform scalability, high availability, and distributed services that help to avoid failures, bandwidth bottlenecks, and management overheads [76]. In comparison to Spark, Hadoop, Twister, etc. that are centrally controlled and use the master node to anticipate and withstand failures, AzureMapReduce is designed around a decentralized control model without a master node. Hence, it avoids a single possible point of failure [76]. Moreover, AzureMapReduce offers the capability to scale up and down computing instances in a dynamic way at any time of the MapReduce computation. It also offers to schedule dynamically *map* and *reduce* tasks in a global queue.

Cloud MapReduce is a MapReduce implementation based on Cloud OS. Cloud MapReduce demonstrates considerable advantages, i.e., a simplicity (because it is easy to operate with MapReduce and beyond it), an incremental scalability (as it offers programmers to add servers while computations run), symmetry and decentralization (because Cloud MapReduce has no master node and all its workers hold the same set of responsibilities), and heterogeneity (as each computing node could have different computation capacity) [107].

Among all the presented MapReduce implementations, widely adopted and used are Apache Hadoop, Apache Spark, and MapReduce in Clouds, i.e., Amazon EMR, AzureMapReduce, Cloud MapReduce.

4.2.2 HPC and big data paradigms convergence

Recently, several papers have been devoted to the application of BD frameworks, specially MapReduce, to the HPC world. Most attempts have tried to adapt MapReduce to use distributed file systems available in HPC environments. For instance, Maltzahn et al. [113] studied the single name-node limitation of the Hadoop file system, when HDFS provides

only one name-node to store the entire file system namespace in memory. This limitation restricts the amount of metadata, which can be stored, and impedes Hadoop scalability and functionality. To cope with this limitation, an object-based parallel file system Ceph has been offered as a scalable alternative to HDFS. Ananthanarayanan et al. [20] studied the performance of traditional file systems (i.e., IBM's GPFS) and compared them with distributed file systems (i.e., Googles GFS, HDFS, Kosmix's KFS) that can provide extreme scalability and reliability but lack a POSIX interface or consistency semantics. This study indicates that traditional file systems are deficient in support of data-intensive applications due to two considerable challenges: function shipping (no support for shipping computation to data and small block sizes) and high availability (data protection techniques that restrict the recover across multiple nodes or disk failures). Wang et al. [166] compared the traditional compute-centric paradigm for HPC applications with the new data-centric paradigm for big data analytics and evaluated the performance of data-centric analytics framework (i.e., Spark) running on a compute-centric HPC environment that relies on high performance file systems. Both studies shed light on performance issues and storage bottlenecks.

There have also been several papers studying the performance of HPC applications on data-centric environment (i.e., Clouds). For example, Evangelinos et al. [57] evaluated the performance of a scientific HPC application on Amazon EC2 and showed that the performance of network in cloud is worse than that of HPC by one to two orders of magnitude. This was also proved by Gupta et al. [78] who conducted a set of experiments on different cloud platforms i.e., Open Cirrus and Eucalyptus Cloud. Both studies show that raw performance difference between compute centric HPC and data centric cloud is pronounced.

Recently, researchers have also shown an increased interest in the use of the BD tools, i.e., Apache Spark, Apache Hadoop, to create workflow execution engines and scientific workflow management systems (SWfMS) for current state of the art workflows [31, 179]. However, this topic is still fairly new and the experience of applying these techniques is still limited. The previous studies have contributed with guidelines and methodological approaches to make the design of scientific workflows easier and more efficient from a user-centric and visual perspective [56]. For example, Dede et al. [45] analyzed the migration of common HPC-oriented workflows to a BD processing platform, i.e., Apache Hadoop. In this theoretical analysis, the authors implemented six representatives of common scientific workflow patterns in Apache Hadoop environment and discussed implementation challenges and Hadoop environment applicability for each of the basic patterns. Whereas Nuthula et al. [122] determined that for the scalability and performance required by modern simulators, it is important to avoid I/O bottlenecks in the cloudification procedure. Considering the complexity of many state of the art simulators for scientific computing, Srirama et al. [148] proposed a workflow partitioning strategy to reduce data communication in the resulting cloud deployment. In this research, the authors focused on task scheduling and underlying peer-to-peer data sharing mechanisms required for their efficient communication.

The aim of the next subsection is to present a cloudification methodology and how it could be enhanced in order to migrate a wider range of scientific workflows to cloud environment with a focus on data locality. This methodology is based on a data-centric approach with data locality playing a critical role in the final performance and scalability of cloud-based applications.

4.3 Cloudification methodology

As discussed in Chapter 3, modern simulators are usually CPU- and memory-intensive. However, their scalability is often restricted not only by the available hardware resources but also by the appropriateness of the chosen programming paradigm. The latter might become particularly crucial in the process of migrating an HPC application to cloud environment. To cope with this crucial restriction, Caíno-Lores et al. [33] proposed a cloudification methodology, which focuses on minimizing internal data transfers by imposing data-locality through a MapReduce framework. The main objective of the methodology is to achieve virtually unlimited scalability of simulators and allow scientists to benefit from clouds with minimal development efforts and modernize systematically their legacy applications.

4.3.1 Description of the original methodology

The key idea of the cloudification methodology is to divide an application into n-number of simulations that run with the same simulation kernel but on n-number of fragments of the original input dataset. This allows to run multiple independent instances of the simulation in parallel. To divide the simulation, an expert shall evaluate the application, determine an independent variable, and use it as an index to partition input data and the following procedures. It is important to underline that the key to this cloudification methodology resides within the application's input data, which shall be partitioned and stored as pairs (instance, parameters) in an indexed structure.

The cloudification methodology consists of two steps: an analysis of the original application to find an independent variable (a partitioning key) and cloudification of the original application, which consists of two sub-steps: adaptation and simulation. These steps are described below in more detail:

1. **Application analysis**: at this step, an expert analyzes the original simulation application in order to find an independent variable, and uses it as an index to partition input data and the following procedures.
2. **Cloudification**: as soon as the independent variable is found, the application is considered suitable for cloudification that leads to two operations described below:
   (a) **Adaptation**, this operation reads the input data, indexes its parameters by the defined independent variable, and splits original data into fragments (independent variable, parameters). Consequently, the following simulations can run autonomously and in parallel on partitioned data fragments.
   (b) **Simulation**, this operation runs the simulation kernel for each partitioned data fragment, generates output files, organizes these files by unique identifiers/keys of each data fragment, gathers the output, and provides final results in a reduced format.

The relevance of this cloudification methodology is clearly supported by the case study and its promising results. For the case study, Caíno-Lores et al. [33] have chosen a real-world simulator [35] with such characteristics:

1. It represents a general sort of engineering simulators commonly used to test and verify different scenarios.

2. The tool requires a considerable amount of computing power to perform complex matrix operations.
3. The application is memory-bound.

During the application analysis, its input dataset was investigated. Every specification file was identified as comprising initial and final time of the simulation. The time parameter was accepted as an independent variable. During the cloudification stage, the initial input data was adapted and rearranged into multiple datasets, each containing necessary information to run a simulation. When the data was prepared, the application was ready for two MapReduce jobs being executed sequentially.

To evaluate the performance of the cloudified simulator, Caíno-Lores et al. [33] conducted experiments in both cluster and cloud (EC2) environments. Overall, the application deployed in clouds showed outstanding results in terms of performance (e.g., the simulation execution stage was 47% faster for the largest experiment) and scalability (linear scalability is achieved under the condition that the problem size increases proportionally with the number of worker nodes) in comparison to the original application.

These promising results prove the viability of the cloudification methodology. In order to improve and extend it to a wider range of applications, we propose to enhance it in the next subsection by dealing with complex iterative workflows that cannot be defined as suitable following the initial cloudification procedure.

4.3.2 Enhancing the methodology: cloudification of complex iterative workflows

The examination of an application is a critical methodology step as an expert must evaluate the application's structure, procedures, data input and define a partitioning key. If no independent variable can be defined, it means that the application is not suitable for this cloudification methodology.

Taking into account the complexity of modern simulators described in Subsection 3.3.3, the identification of an independent variable for the entire simulation might be considerably restricted by the need of executing not only one simulation workflow but also many additional sub-workflows, especially when they are triggered based on outcomes of the previous sub-workflow. A good example of such a simulation could be LEAD, which consists of two sub-workflows: weather forecasting and data post-processing by a data mining component, which can also trigger the execution of two supplementary forecast workflows or even steer remote radars for extra localized data [139].

To define an independent variable in such complex scientific workflows, we propose that during the application analysis step of the original methodology, an expert conducts a multilayered analysis. At the first layer, the expert evaluates the entire simulation. If it is not possible to identify an independent variable for the entire application, then at the second layer, the expert conducts a more profound analysis of the application based on its characteristics (e.g., size, constraints, structural patterns) and searches separately for an independent variable in a simulation part (e.g., a workflow, a sub-workflow, a task).

We aim to enhance the original methodology by enlarging its application to a greater number of applications, specifically complex and iterative ones. The enhanced methodology is directed at guiding complex and iterative workflows to clouds and maintaining a comparable level of performance against a traditional infrastructure. The enhanced version of the original methodology preserves all its key features and advantages. It is built in a data-centric manner,

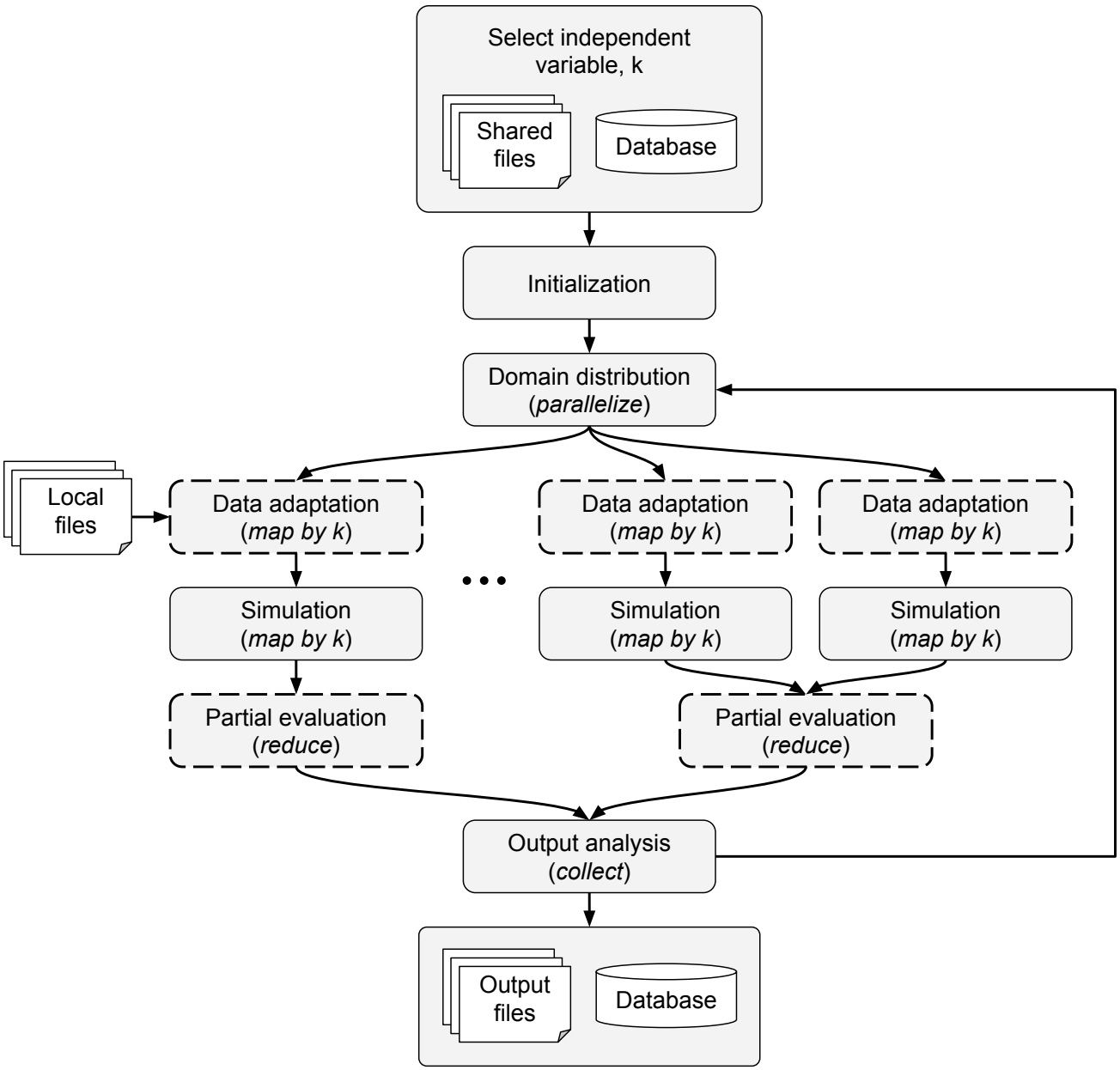


Figure 4.1: Overview of the big data inspired cloudification methodology. Dashed boxes indicate optional stages, which may not be necessary for every iterative workflow.

because data locality plays a major role in the final performance and scalability of cloud-based applications. The enhanced methodology suits well cloud environments, which are engineered to keep both computation and storage on nodes, i.e., VMs. Hence, it can provide a high degree of parallelism with regard to target infrastructure and data management.

The proposed methodology is depicted in Fig. 4.1. It consists of the following steps:

Key Selection. First, it is necessary to conduct a multilayered analysis of the original application in order to find a domain or its part(s), e.g., a workflow, a sub-workflow or a task, suitable for parallelization. This entails finding only one independent variable to act as partitioning key (k, in Fig. 4.1), which will guide the distribution of the defined area and the following stages. It is important to mention that if an independent

variable was found in more than one simulation part, then the methodology shall be applied multiple times for each parallelizable area.

Domain Partitioning. Once the parallelizable area is selected, we can model how the input will be distributed across the nodes. This parallelization stage distributes the proper portion of the input, for each value of k. This sets the fraction of the input data or model that will be processed for each instantiation of the guiding independent variable.

Simulation. One or more simulation stages wrap the kernels involved in the simulation workflow to simulate each portion of the parallelizable area in an independent and autonomous way. This yields the execution of not one, but many smaller simulations. The large number of simulations to be executed factors the inherent complexity of the simulation process, yet it can be massively distributed due to the independent and autonomous nature of each simulation. Additionally, if there is a need for processing key-specific input, we can exploit data locality to minimize data transfers in this stage by scheduling the pipeline where the data is present.

Partial Reduction. Optionally, one or more reduction stages can be defined to filter or join partial outputs before the overall collection and evaluation of results to reduce contention in the synchronization point.

Output Processing. This constitutes a collection stage followed by processing and analysis methods in charge of creating the input for the next iteration. In this step, the output evaluation must be defined to reflect the end criteria, generation of the following input, and validity of the results per iteration.

The aforementioned steps are aimed at finding a parallelizable simulation area, where we are able to identify an independent variable to act as an index for subsequent steps. This must support the parallelization of that specific area in a key-value manner, so that further simulation pipes and optional partial evaluations can take place independently, as seen in massively-parallel data analytics frameworks. Of course, any partition-specific data will only concern the node, which is going to process such domain partition. Hence, we can schedule the computation on the proper node to support data locality. This is particularly interesting if several stages of one parallelizable area are involved, since they can be scheduled together to benefit from local intermediate files. After these procedures, partial results can be filtered and assessed in parallel as well, again following the initial area distribution. Finally, these partial results can be analyzed and processed to build the next iteration. Although the effects of this synchronization point can be alleviated by previous partial reductions.

4.4 Enhanced methodology application to EnKF-HGS

Following the enhanced model of the cloudification methodology described in Subsection 4.3.2, we proved its viability by applying it to the original EnKF-HGS application. As described in Subsection 3.4.2, EnKF-HGS application is an MPI implementation of a data assimilation process via the ensemble Kalman filter. Specifically, data assimilation minimizes the deviation of a numerical model from the real physical system and supports the continuous improvement of HGS stochastic simulations by incorporating the most recent field measurements.

4.4.1 Simulator analysis

Following the enhanced cloudification procedure, we begin with an application of the first step – a simulator analysis. The simulator's input is tightly-coupled. It describes a single hydro-geological real-world system, which we cannot decompose. Since we could not define an independent variable for the entire simulator, we continue with a second step of the multilayered analysis.

Analyzing the simulator's workflow, we could see that it is iterative and each iteration consists of two phases: forward propagation and filtering. At the forward propagation phase, we operate with a set of realizations, which constitute instantiations of the underlying model. These realizations are simulated independently, and the output is gathered afterwards for processing. Here it is worth mentioning that the forward propagation phase could be divided into two layers. The first layer consists of the Monte Carlo simulation, while the GROK and HGS simulation kernels compose the second layer. In brief, GROK is a preprocessor that prepares the input files for HGS, which makes GROK an I/O intensive application. While HGS is an integrated hydrological modelling simulator, which mainly relies on the CPU to solve differential equations. Considering the definition of the Monte Carlo simulation method described in Subsection 3.3.2, every process is completely independent. Hence, the EnKF-HGS first layer is totally parallelizable. However, the second layer can be neither analyzed nor modified, because both kernels are proprietary binaries. Hence, it is not suitable for the methodology. At the filtering phase, we update each of the realizations with the environmental field measurements to adjust the deviated models towards the actual state of the real-world system. This process represents a set of matrix operations and requires multiple data synchronizations, which leads to tight coupling and interdependency of processes. Considering their interdependency, it is difficult to completely parallelize this phase. Following the methodology, we aim at parallelizing only the forward propagation phase and select the realization identifier as a key and the collection of the model's data per realization as a value. These key-value pairs could be independently distributed and executed across computing nodes. Their results will be gathered in the master process for the filtering step.

4.4.2 Cloudification procedure

As described in Subsection 4.4.1, the EnKF-HGS workflow is iterative and has multiple data-synchronization points. Hence, MapReduce representative should be carefully selected out of all available implementations described in Subsection 4.2.1. We chose Apache Spark 1.6.0, which is currently a major representative of a BD programming model and execution engine. Moreover, Spark was developed particularly to overcome significant performance penalties during iterative jobs and frequent data accesses by using in-memory datasets, and natively supports iterative pattern.

Execution models: Spark vs. MPI

Figure 4.2 depicts the stages that belong to the final implementation of the workflow in Spark. The procedures executed in the Spark driver process are identified using letters (from A to D), while numbers (1 to 6) refer to tasks that are computed distributively in the Spark executors. The most relevant design and implementation details are described in the following paragraphs.

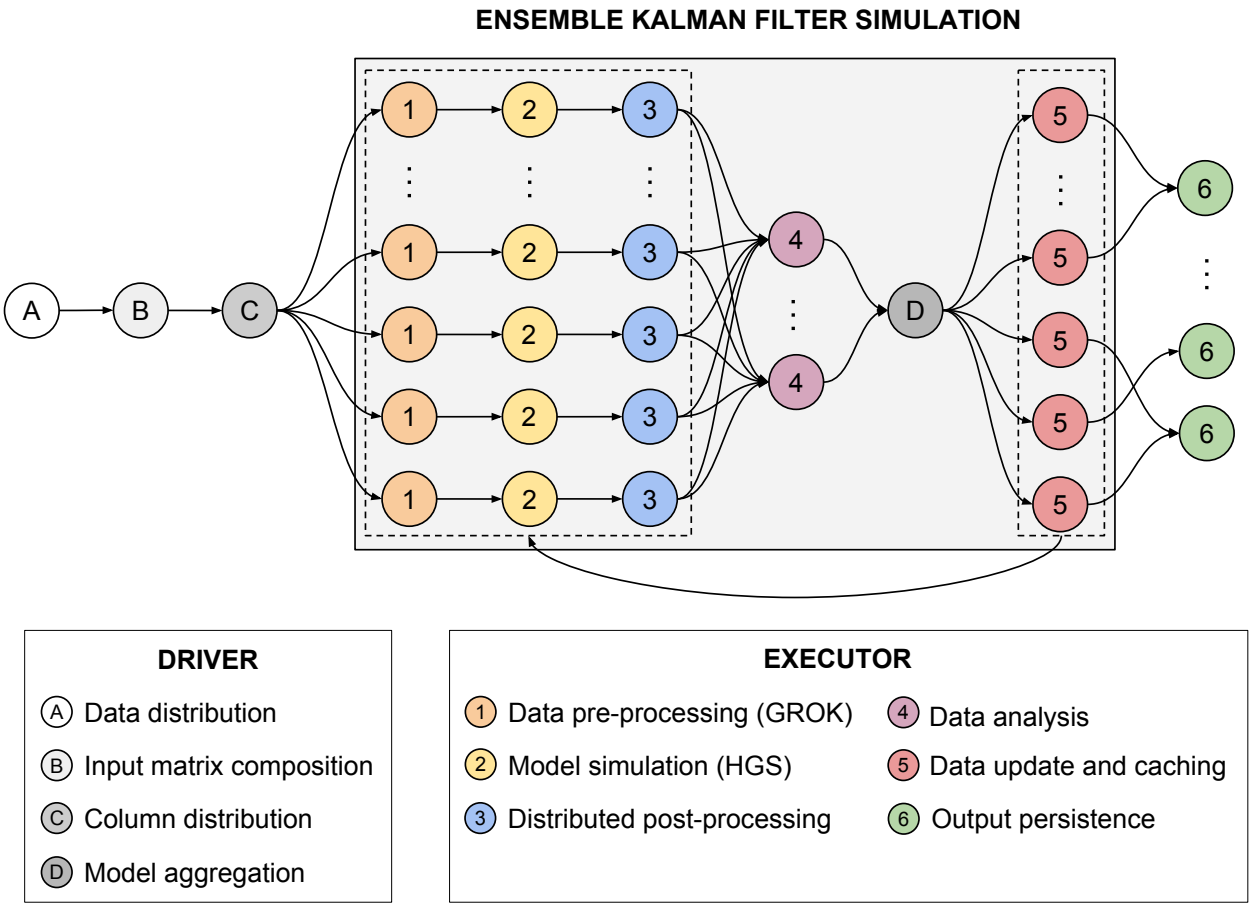


Figure 4.2: Execution model of the cloudified EnKF-HGS. Dashed lines indicate that the operations are executed over a distributed dataset.

Ensemble preparation stage

A. Data distribution

The first step is to load the necessary auxiliary files that every executor will need to properly run its data partition. For instance, this includes the kernel binaries. Spark guarantees that these files will be available for the worker nodes in their current working directory.

B. Input matrix composition

Input data is read in the driver process in order to initialize the base model consisting of two main matrices, M_1 and M_2, in which each column $c_{1,r}$ and c_{2_r} corresponds to an instantiation, r, of the model. Additional data structures are created and initialized and the parameters of the simulation are obtained.

C. Column distribution

Both matrices are distributed by columns in order to build the realization set, R. Each realization r is composed of the corresponding columns from both matrices, $C_{1,N}$ and $C_{2,N}$. Figure 4.3 depicts the realization data distribution process, which yields the distributed dataset that will be transformed in the following stages and iterations. The rationale behind distributing the workload this way is that each realization can

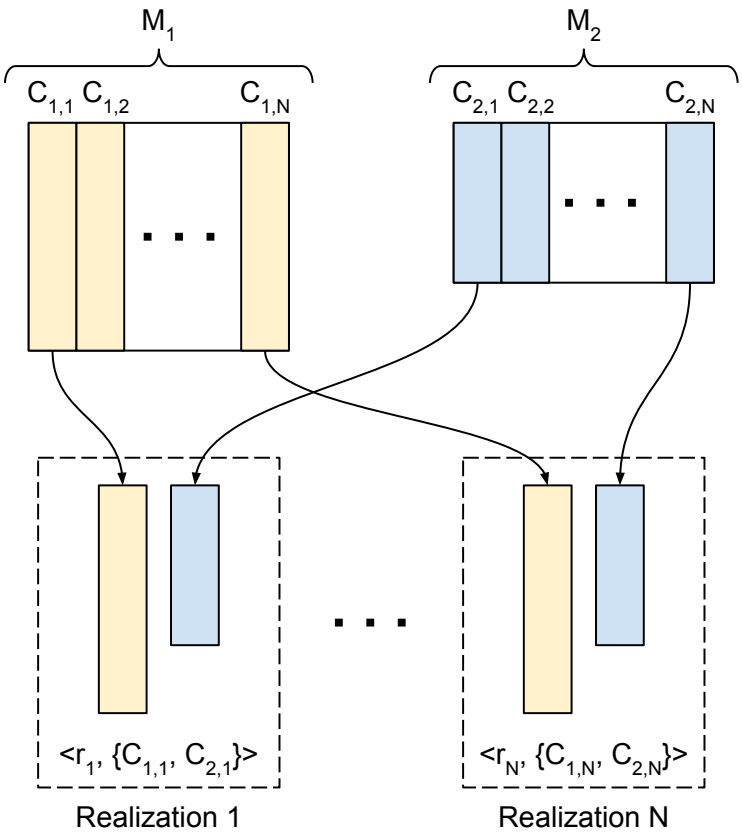


Figure 4.3: Column distribution procedure. Both matrices are split column-wise, and realizations are built with the corresponding columns from both matrices.

be simulated independently from the others, without any further communication. Additionally, we forced each partition to hold the data for a single realization in order to induce fine-grained parallelism.

Iterative ensemble simulation

1. **Data pre-processing (GROK)**

   After realizations are distributed, the GROK kernel writes the realization input data to local files. HGS will read these files in order to conduct the simulation of the model.

2. **Model simulation (HGS)**

   With the input files from GROK, HGS simulates the model and writes its output for subsequent analysis. In steps 1 and 2 we must ensure that both binaries will be executed in the same node to exploit data locality. To achieve this, we run GROK and HGS in the same map function, which is an indivisible task in Spark. Thus, they act as an inner pipeline within the workflow.

3. **Distributed post-processing**

   The post-processing stage is partially distributed. First, the output from each HGS execution is read in each executor in order to create an updated realization set, R'. With this information we create a distributed matrix M_1' and conduct several distributed operations to avoid gathering the whole matrix in the driver.

4. **Data analysis**

   Further operations with auxiliary matrices are executed in the driver in order to filter and randomize the input for the following iteration. The goal of this stage is to minimise the size of the dataset that needs to be collected in the driver prior to the model update. Note that to achieve this, significant data shuffles must be executed.

D. Model aggregation

Since not every stage of the analysis could be distributed, there is a step, in which we aggregate the final matrix that will be used to compute an update matrix. This matrix is distributed afterwards, so that we can update the realizations without gathering the whole dataset in a single node.

5. **Data update and caching**

   The distributed update matrix is used to update every realization in parallel. The resulting realization set is persisted to the local storage of the nodes as a fault-tolerance measure and the following iteration starts.

Output management

6. **Output persistence**

   After every iteration is executed, the output is stored to HDFS. This is executed in parallel, as every partition is stored independently.

Considering the initial EnKF-HGS MPI implementation described in Subsection 3.4.2, now we can compare in general terms both MPI- and Spark-based execution models. Regarding the Spark's nature, it can offer such considerable benefits as:

1. Distributed file system with failure and data replication management;
2. Set of tools and libraries for data analysis and management that will facilitate the use, deployment, and maintenance of the parallelized workflow;
3. Unrestricted addition of nodes at runtime.

Overall, the Spark-based execution model took a full advantage of fault-tolerance, automatic re-execution of failed tasks, and dynamic adaptation to resource pool size. However, considering the EnKF-HGS execution of two proprietary binaries GROK and HGS, the Spark implementation is based on the current MPI model that might restrict the exploitation of the Spark platform to its full extent. Also, it is worth to mention that the MPI-based model benefits from more light-weight processes.

4.4.3 Evaluation

The goal of our evaluation was to assess the benefits and drawbacks of the application of the techniques discussed in Section 4.4. We focused on absolute execution time and speed-up to analyse the effects of memory and virtualization overheads of Spark and clouds, respectively. In addition, we conducted further experiments to evaluate the influence of I/O saturation on the application performance, which might occur due to a lack of high speed network connection between distributed resources.

4.4.3.1 Implementation and simulator configuration

Following the general comparison of the original MPI and Spark execution models, we propose to compare in depth original and Spark implementations in a traditional cluster against two types of cloud infrastructures: a private cloud running OpenNebula, and the AmazonEC2 public cloud. Hence, we will be able to evaluate the scalability, and analyze the behavior of the platform and the infrastructure as the problem size increases.

A particularity of the kernel binaries is that they are pre-build black boxes. An effect of this is that they rely on hard-coded input paths for the intermediate files they handle. This constitutes a limitation, as we are forced to execute both binaries on the same machine to ensure data locality per realization. To achieve this, we exploited Apache Spark's partitioning mechanisms to ensure that each full realization is computed on the same node in a pipelined manner. As seen in Figure 4.4, the current workflow forms a collection of independent pipelines that map to each realization with the proper input data. There is a possibility to further optimize this but currently it is a promising approach that can be generalized to many applications that make use of this filter.

4.4.3.2 Experimental setup

In order to assess the performance and scalability of the application, we selected three different execution infrastructures: a cluster, a private cloud running OpenNebula, and a virtual cluster on the AmazonEC2 public cloud. The specifications and limitations of these testbeds are described as follows:

Testbed A: local cluster

This testbed comprised 8 worker nodes for the MPI platform and 8 worker nodes plus 1 additional driver node for the Spark platform (see Table 4.1). Each worker node holds 8GB of RAM and two Intel Xeon E5405 @2.00GHz processors with four cores each. The driver node was necessary to host the driver process of the Spark implementation, which required 7GB in the largest experiment we conducted. This means that the container in charge of running the driver would require 7GB plus a 10% memory overhead (as configured by default in the platform), 512MB extra memory for heap space, and other overhead sources like serialization buffers. Since Spark adds significant memory overhead to drivers and executors, we had to add a larger node to bypass the memory constraints in the worker nodes. Consequently, we added a node with an overall amount of 94GB of RAM and four Intel Xeon E7-4807 @1.87GHz processors with six cores each (see Table 4.2). The local cluster already had a pre-installed network file system GlusterFS 3.7.11 (a scalable and production-ready network file system [12]), which was necessary for the MPI implementation execution.

Platform	**Spark**		**MPI**
Node role	**driver**	**worker**	**worker**
Type	xlarge	large	large
Amount	1	8	8

Table 4.1: Cluster machine selection for the Spark and MPI platforms.

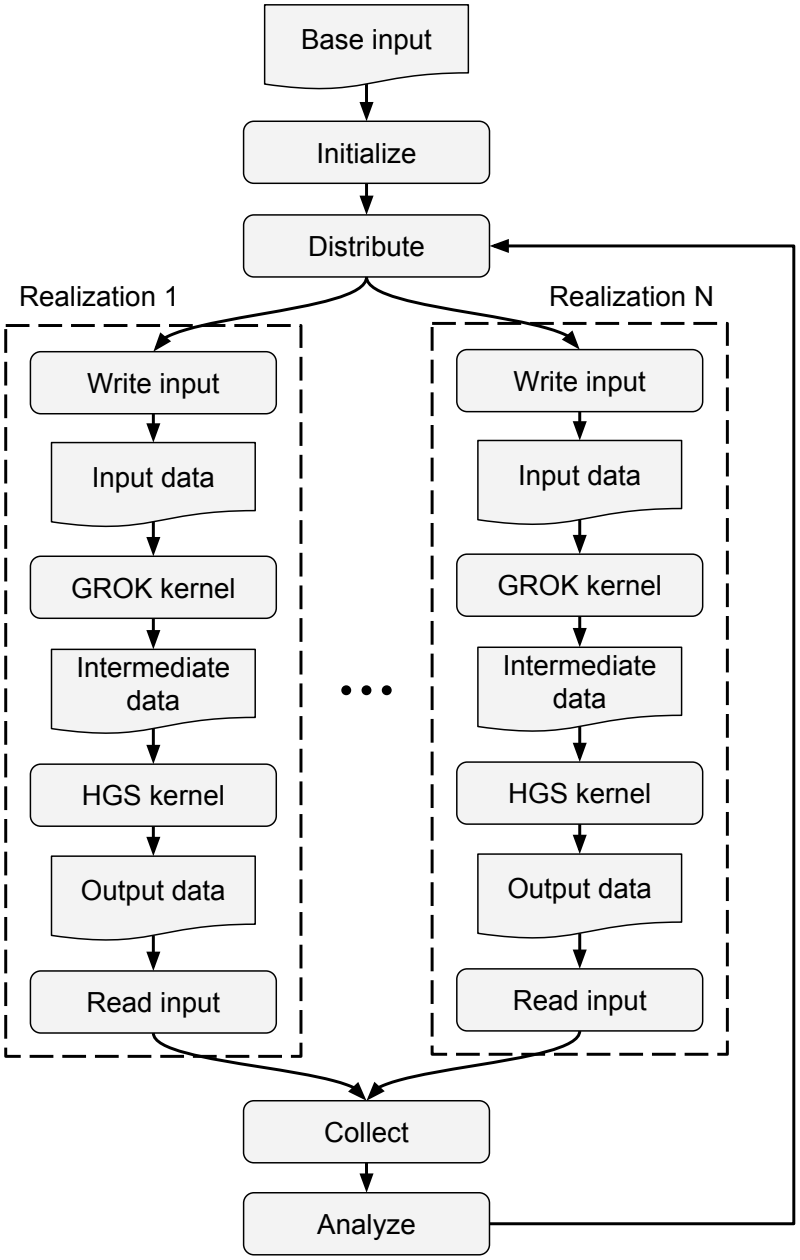


Figure 4.4: Final Spark workflow for the cloudified EnKF-HGS.

Type	**Processor**	**CPU Cores**	**Memory**	**Storage**	**Network Performance**
large	2 x Intel Xeon E5405 @2.00GHz	8	8GB	HDD 2 x 1000GB	Ethernet 1Gb/s
xlarge	4 x Intel Xeon E7-4807 @1.87GHz	24	94GB	HDD 2 x 1000GB	Ethernet 1Gb/s

Table 4.2: Technical specifications of the selected machines (testbed A).

Testbed B: private cloud

We relied on a public cloud running OpenNebula with the nodes configuration and hardware configuration described in Table 4.3 and Table 4.4 respectively. Notice that the main difference between this infrastructure and the cluster is the clock speed, which would benefit this testbed in the evaluation. To build a virtual cluster, we spawned 8 (and 9 in the case of Spark) 8-core VMs with 7.5GB of RAM each, the maximum available memory per VM. Notice that this memory limitation is relevant, as there was no workaround to fit the driver safely in the largest experiments. For the network file system, we deployed the latest version (3.7.14) of GlusterFS on 3 additional storage nodes, which were organized in a distributed volume with no data-replication in order to maximize the storage performance. On the computing nodes side, we exploited the FUSE-based Gluster Native Client for highly concurrent access to the file system.

Platform	**Spark**		**MPI**	
Node role	**driver**	**worker**	**worker**	**storage**
Type	large	large	large	medium
Amount	1	8	8	3

Table 4.3: Private cloud VM selection for the Spark and MPI platforms.

Type	**Processor**	**vCPU Cores**	**Memory**	**Storage**	**Network Performance**
large	2 x Intel Xeon L5420 @2.50GHz	8	7.4GB	HDD 500GB	Ethernet 1Gb/s
medium	Intel Xeon L5420 @2.50GHz	4	3.5GB	HDD 500GB	Ethernet 1Gb/s

Table 4.4: Technical specifications of the selected VMs (testbed B).

Testbed C: public cloud

We selected *c4.2xlarge* instances as workers due to their balance between number of cores and amount of memory and a *r3.xlarge* instance with larger memory to hold the driver. The virtual cluster for Spark consisted of 16 workers (128 cores in total), an *m3.medium* master, and the additional dedicated VM for the driver. MPI ran on a virtual cluster built with 16 identical workers, plus 3 additional *c4.large* storage nodes running GlusterFS with the configuration similar to the previous testbed. Each storage node was provisioned with a 5GB general purpose SSD brick. Table 4.5 shows a summary of the selected instances and their assigned roles in both the Spark and MPI execution platforms. In addition, Table 4.6 describes the hardware characteristics published by the provider [11].

Platform	Spark			MPI	
Node role	**master**	**driver**	**worker**	**worker**	**storage**
Type	m3.medium	r3.xlarge	c4.2xlarge	c4.2xlarge	c4.large
Amount	1	1	16	16	3

Table 4.5: AmazonEC2 instance selection for the virtual clusters running the Spark and MPI platforms.

Type	Processor(*)	vCPU Cores	Memory (GiB)	Storage (GB)	Network Performance
m3.medium	Intel Xeon E5-2670 v2 @2.5GHz Intel Xeon E5-2670 @2.6GHz	1	3.75	SSD	Moderate
r3.xlarge	Intel Xeon E5-2670 v2 @2.5GHz	4	30.5	SSD	Moderate
c4.2xlarge	Intel Xeon E5-2666 v3 @2.9GHz	8	15	EBS	High
c4.large	Intel Xeon E5-2666 v3 @2.9GHz	2	3.75	EBS	Moderate

(*) More than one item is listed if VMs can be indistinctively launched on different physical processors.

Table 4.6: Technical specifications of the selected public cloud instances (testbed C), as provided by the AmazonEC2 documentation.

4.4.3.3 I/O saturation per infrastructure

Before evaluating the scalability of two implementations and comparing their performance, it is important to measure the effect of concurrent access to a network file system on execution time of the overall application. As described in Subsection 3.4.2, the forward propagation phase of the EnKF-HGS workflow runs an ensemble of concurrent HGS model realizations, which use a file-based data exchange mechanism via a network storage. It differs from the Spark implementation, where each execution runs on local data by design. In order to measure this, we had to guarantee that the number of I/O operations was identical for every instance of the GROK and HGS kernels. Since both kernel binaries are pre-built black-boxes, we modified the EnKF-HGS to compute one model realization N times, where N is the number of ensemble members. Thus, instead of computing N different model realizations, which most likely would result in a different number of the I/O operations for each individual realization, we ensured that each instance of the simulator performed the same I/O operations by making each input identical.

We have measured the relation between the number of concurrent processes (i.e., GROK and HGS kernels) that are using the network file system and the execution time in three testbeds presented above. Additionally, for the testbed C, we repeated the experiment with the Elastic File System (EFS), which is a native network file system for AmazonEC2 environment. For the experiment's execution, we modified EnKF-HGS to run 1 to 64 MPI processes, where each process was provided with one CPU core. Figures 4.5, 4.6, 4.7, and 4.8 show the impact of concurrent accesses to network storage on the execution time of the kernel binaries (GROK and HGS separately). Surprisingly, even in experiment 1, which is conducted in the Testbed A, in an "MPI-friendly" environment, the performance of the

I/O intensive kernel (GROK) drops 5 times in case of 64 concurrently running instances of the kernel. The results of experiments 2, 3, and 4 illustrate that in cloud environment the situation is even more dramatic. Because a similar performance drop happens not only for the I/O intensive kernel but also for the compute intensive one. From these experiments, we can clearly see that typical MPI-based application assumptions, i.e., an availability of a high performance network file system and a low-latency broadband network connection, should be seriously taken into consideration. Because a violation of these assumptions might lead to a tremendous drop of an application performance. In this experiment, we artificially equalized the number of the concurrent I/O operations, which resulted in a large performance drop. For a realistic ensemble, consisting of individual model realizations, the computations most likely would not last the same, which in turn would reduce the number of concurrent I/O operations, consequently, reducing the saturation.

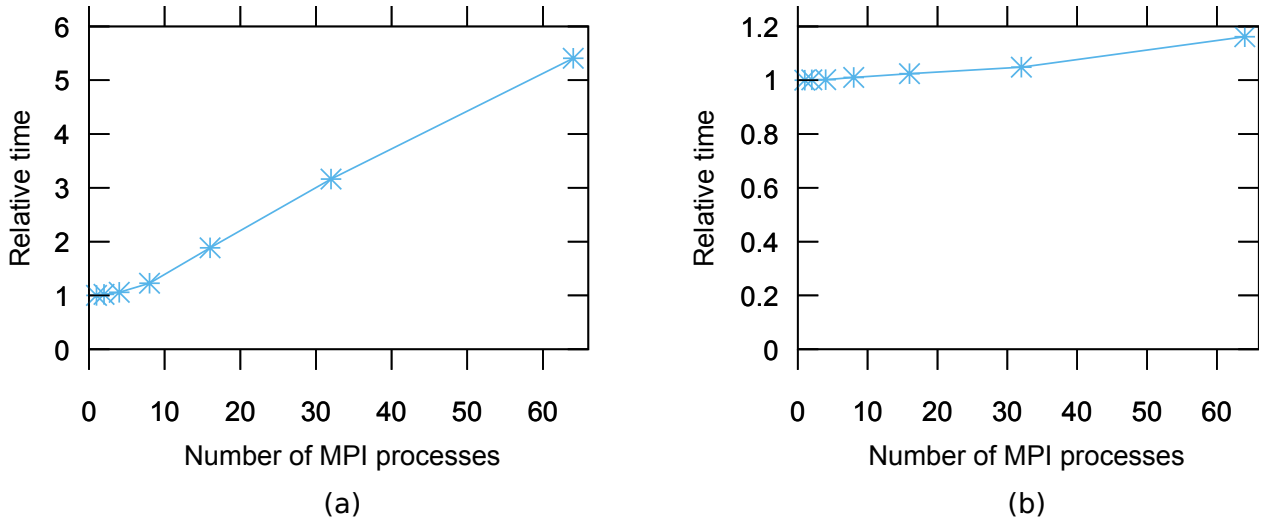


Figure 4.5: Experiment 1. Relative execution time of concurrent (a) GROK and (b) HGS kernels running on local cluster with GlusterFS.

The following experiments will illustrate the performance and scalability of two implementations in the presented testbeds.

4.4.3.4 Performance analysis: cluster

The objective of these experiments is to detect the effects the execution models have on the performance of the workflow execution. We analysed the MPI-based implementation of EnKF-HGS, and its data-centric version built in Spark, both running in the local cluster formerly described.

We allocated Spark executors with one core in order to fairly compare scalability against single-core MPI processes. We experimented with increasing number of realizations and executors. We measured the absolute execution time for a single execution (including the job launch time required by Spark) and computed the speed-up. The results for this execution on a local cluster are shown in Figure 4.9, in which experiments (a) and (c) correspond to MPI, while experiments (b) and (d) correspond to Spark. Remarkably, Spark yields better execution times for every experiment, and its speed-up is better the larger is the experiment for a given number of workers. This is the result of storing data on a local storage instead of a network one. However, the speed-up in Spark for the largest experiment (i.e.,

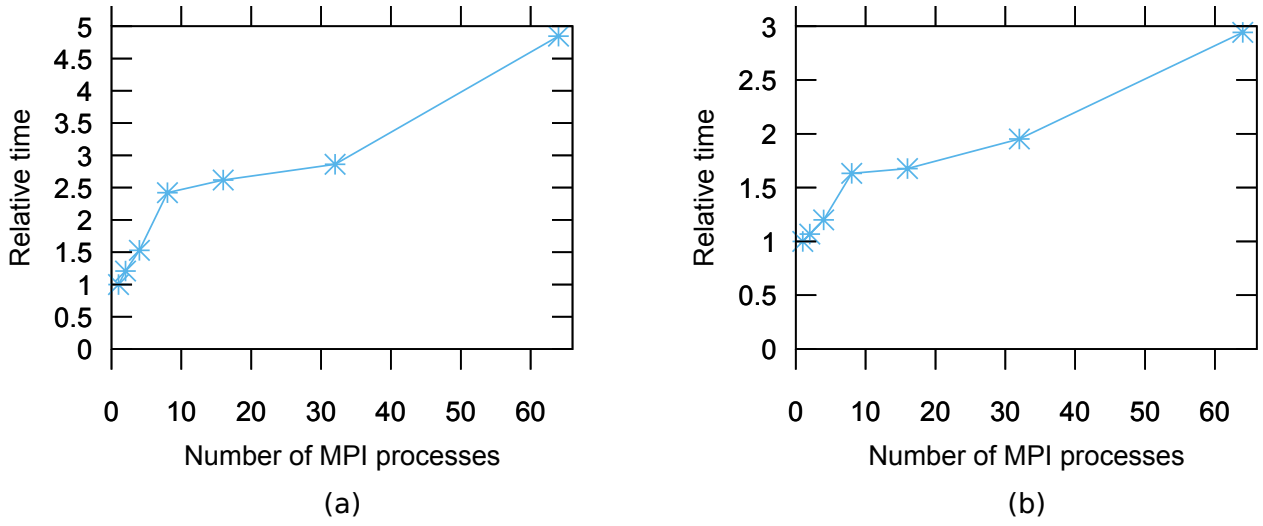


Figure 4.6: Experiment 2. Relative execution time of concurrent (a) GROK and (b) HGS kernels running on OpenNebula with GlusterFS.

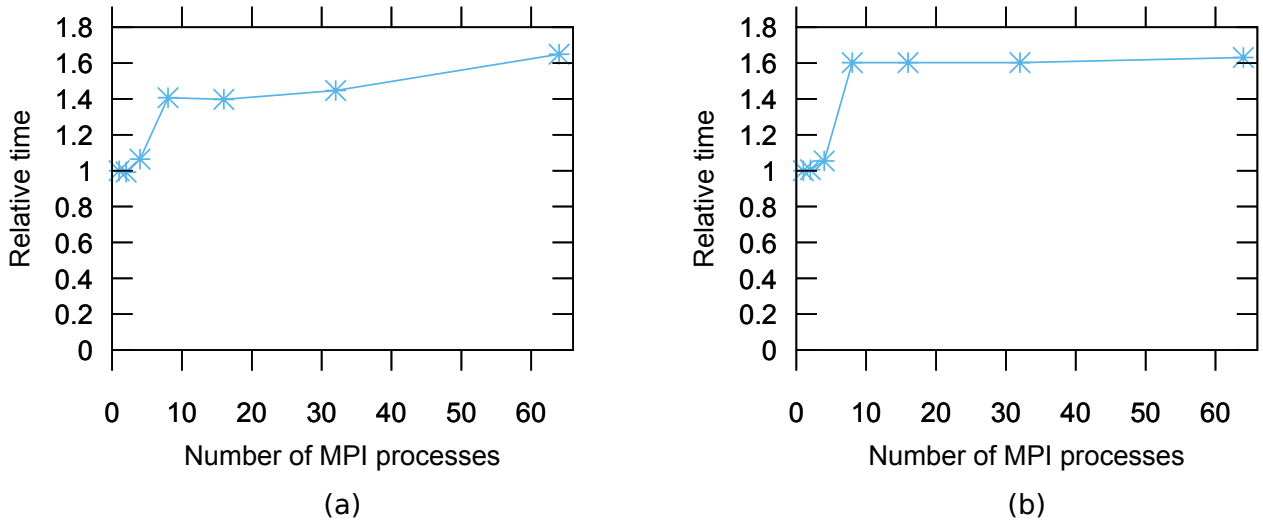


Figure 4.7: Experiment 3. Relative execution time of concurrent (a) GROK and (b) HGS kernels running on AmazonEC2 with GlusterFS.

64 realizations on 64 executors) is lower than in the MPI case. In this case, the problem is that the 64 executors cannot be scheduled at once due to their large memory requirements. The main conclusion from this experiment is that, while the BD-inspired approach shows outstanding performance results, the memory overhead of the execution framework hurts scalability because less parallel executors can be allocated in the same infrastructure. As a result, the slimmer MPI processes seem more suitable for large scale execution of this workflow. Another interesting aspect is related to the post-processing stage of the workflow and its effect on the overall execution time. At some point, with the growing number of the parallel executors, the post-processing computation becomes shorter in time than the data transferring time. As a result, the post-processing stage might affect the overall execution time, when there is a greater number of parallel executors.

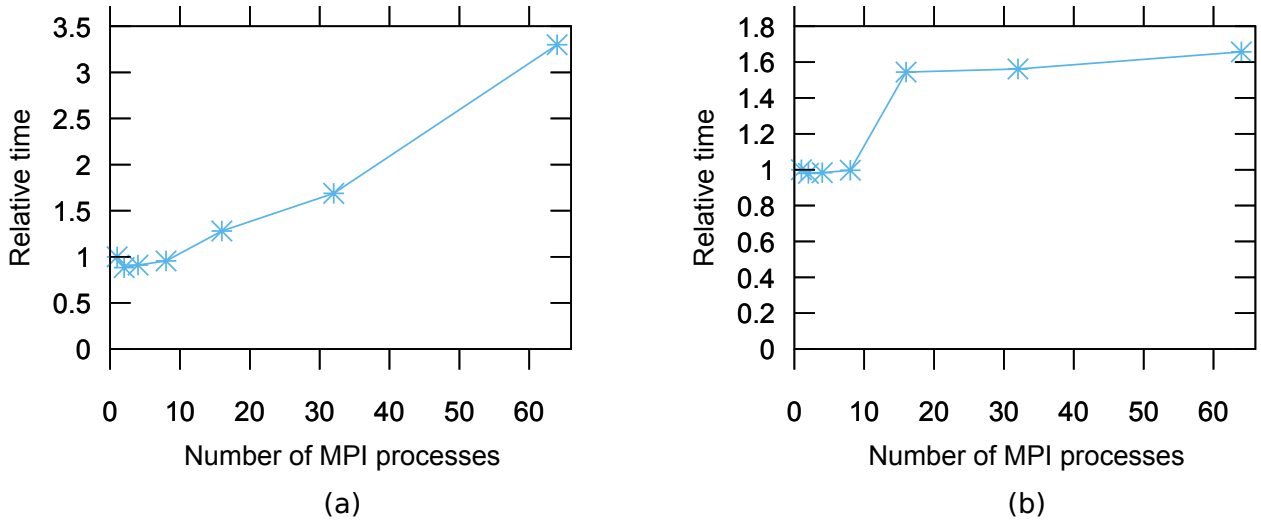
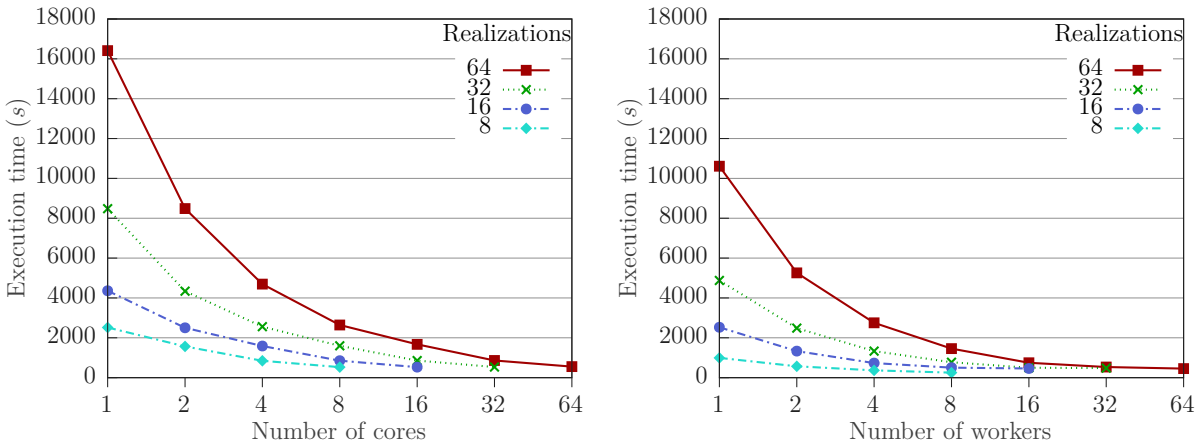
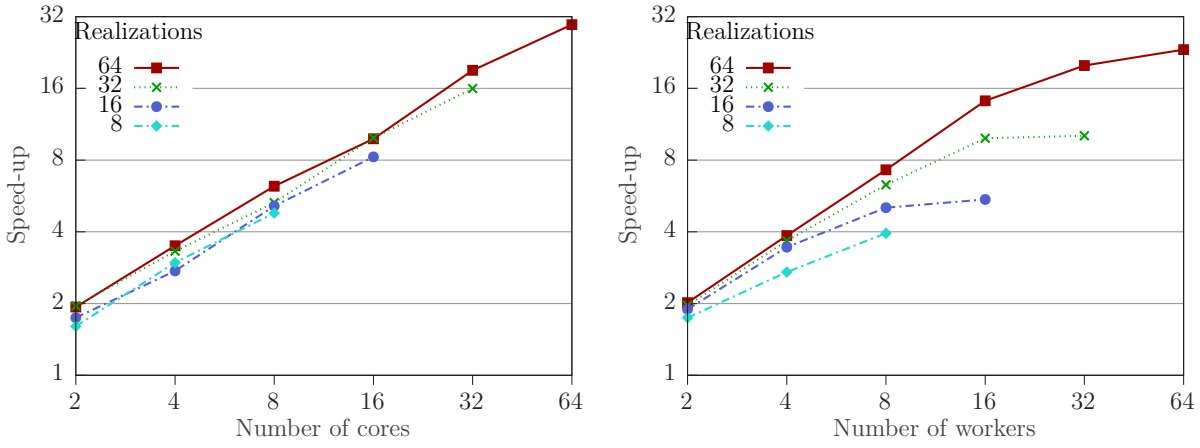


Figure 4.8: Experiment 4. Relative execution time of concurrent (a) GROK and (b) HGS kernels running on AmazonEC2 with EFS.



(a) Execution time (MPI).

(b) Execution time (Spark).



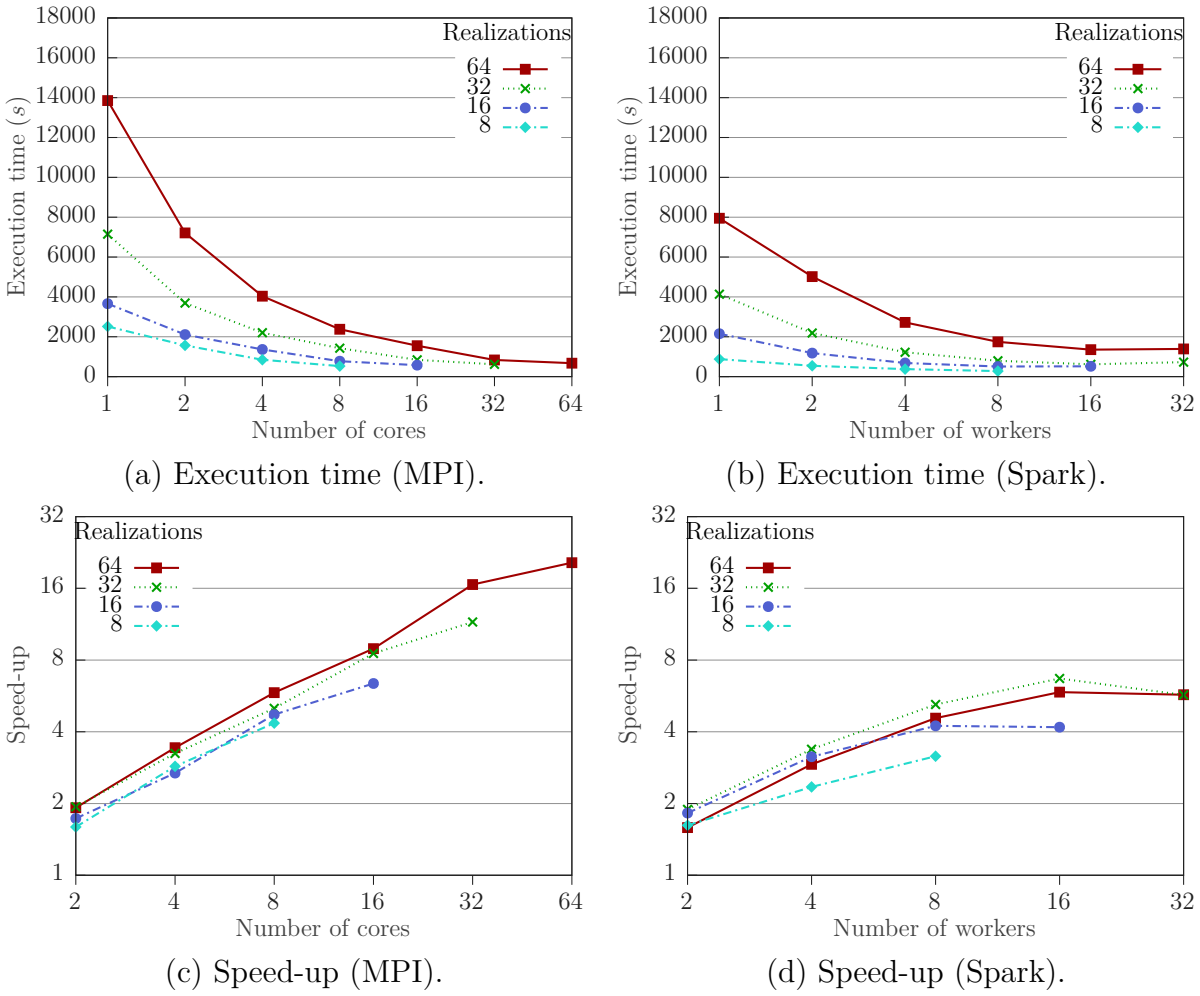
(c) Speed-up (MPI).

(d) Speed-up (Spark).

Figure 4.9: Execution time and speed-up for the MPI and Spark implementations, running on a local cluster.

4.4.3.5 Performance analysis: private cloud

After analysing the programming models, we focused on the underlying computing models to assess their impact on performance. Hence, we conducted further experiments on the OpenNebula private cloud with the Spark and MPI implementations.



(a) Execution time (MPI).

(b) Execution time (Spark).

(c) Speed-up (MPI).

(d) Speed-up (Spark).

Figure 4.10: Execution time and speed-up for the MPI and Spark implementations, running on a private OpenNebula cloud.

Besides the cluster results shown in the experiments (Figure 4.9), Figure 4.10 reflects the same metrics for the experiments conducted in OpenNebula for a single execution. Spark benefits most from BD-oriented environments like Clouds, while keeping an overall evolution of speed-up similar in relative terms. However, we can also see that in OpenNebula Spark results are much more extreme with lower execution times and smaller speed-ups. Interestingly, MPI also shows a similar behaviour with lower execution times and degraded speed-ups. These results could be associated with the larger frequency of the physical processors of the virtual cluster, which yields faster executions, and the lower memory per core ratio, which hurts scalability.

With respect to the results for Spark, there are two remarkable exceptions. The first exception is related to the 64-realization execution with 64 workers. This experiment failed because the container corresponding to the driver violated the limit of the system's memory. Hence, this result is not included. The other exception is associated with the experiment of

32 realizations with 32 workers. Its result is identical to the experiment of 16 realizations with 16 workers. We noticed that this happened because the heartbeat of several executors was missed in the driver. Consequently, that caused the executors' assigned task to time-out. Under these circumstances, Spark considers that the worker is dead. Hence, it reschedules the task in a different node. Since there are no more resources to launch a new container, it has to wait for an executor to finish, and then launch the former tasks. Considering the synchronization point after the concurrent computation of the realizations, task rescheduling leads to the doubled execution time, which corresponds to the time obtained for 16 executors.

4.4.3.6 Performance analysis: public cloud

Given the resource limitations in our private infrastructures, we moved both execution paradigms to the AmazonEC2 public cloud to test further scaling. We incrementally increased the size of the virtual cluster and conducted experiments until one of the implementations showed significant scalability issues that made further increases in problem size or number of workers impractical. Our goal was to determine which paradigm showed the most limitations to support large-scale executions.

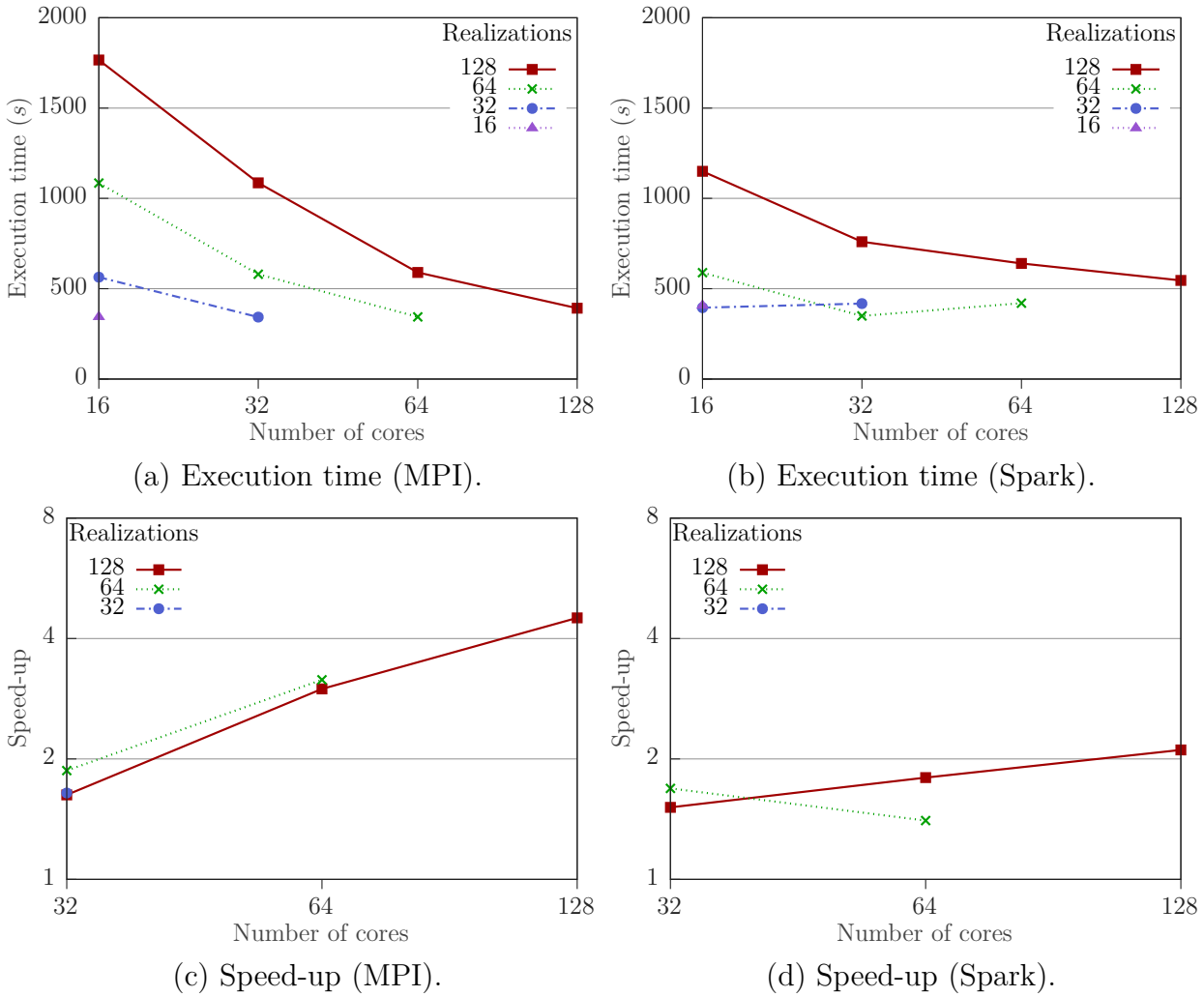
We considered the set-up described above enough to run 64 realizations smoothly in both MPI and Spark. We attempted 128 realizations to check whether we could exploit the whole cores in the system with the Spark implementation.

Figure 4.11 shows the results coherent with the fast speed-up degradation shown in OpenNebula, when driver and executor memory sizes increase to the point, in which all the realizations cannot be computed in parallel (i.e., memory per core increased beyond 2GB). The major problem is not a performance degradation. It is rather a case of not being able to run the experiment because a job will not simply finish when there is not enough memory to host the driver or executor container, or their related overheads. The former experiments clearly indicate that the Spark implementation will not scale due to its memory requirements when running on the Spark stack. This is also problematic if cost is taken into consideration, as it would be required to select machines with larger memory per core ratio. Thus, it will be more expensive. On the other hand, the MPI-based workflow does not show either an outstanding performance or speed-up. It is able to scale further with less resources and in a stabler manner.

We also found that Spark has limited capabilities to establish simultaneous connections among many nodes. Our first approach to make data integration was to use memory mechanisms instead of files. However, it requested to create an all-to-all communication pattern. Even if this pattern is straightforward and efficient in MPI, we were unable to scale this approach up to 256 nodes in Spark. That happened because the Spark underlying RPC system failed due to the impossibility of creating enough connections.

4.4.3.7 Results of the enhanced methodology application to EnKF-HGS

Overall, BD programming models (i.e., Hadoop and Spark) and infrastructures (i.e., clouds) might be used to improve the efficiency and scalability of some types of scientific applications, specifically simulators relying on parameter-sweep and partitionable domains, and kernel-based workflows comprising many, loosely-coupled tasks. The detailed evaluation of the original workflow and Spark implementations in different infrastructures (i.e., a cluster and clouds) indicates that the BD techniques are still far from disrupting scientific applications. Specifically, this statement is relevant to the vast set of complex workflows with major



(a) Execution time (MPI).

(b) Execution time (Spark).

(c) Speed-up (MPI).

(d) Speed-up (Spark).

Figure 4.11: Execution time and speed-up for the MPI and Spark implementations, running on a virtual cluster on the AmazonEC2 cloud.

synchronization and resource requirements, when the scale increases. This experience and the current state of the literature is summarized in Tab. 4.7 and detailed as follows.

Among different MapReduce implementations described in Subsection 4.2.1, we used Spark as it is specifically designed for iterative processes and provides such competitive advantages as fault-tolerance, automatic re-execution of failed tasks, and dynamic adaptation to resource pool size. Regardless of its advantages, the Spark-based iterative workflow was not able to scale properly, mainly due to the significant memory overhead introduced by the execution environment. We believe that an extensive memory consumption was due to the deep component stack and its dependence on the JVM [73]. The additional resources in the memory stack are needed to mitigate the requirements of new workloads and support of the emerging in-memory and caching mechanisms coming from data-aware computing.

Regarding workflow development and deployment, we experienced a considerable reduction of time on building a Spark-based solution and deploying it on AmazonEC2. Without any user-related or software compatibility issues, the solution was tailored to meet all the experiment requirements. Moreover, Spark with its underlying resource manager and distributed file system substantially facilitated data distribution and task management. We only had to consider the design of the workflow and some specific implementation

	Big data paradigm	HPC paradigm
Pros	Fault-tolerance by design	Low resource consumption
	Transparent data locality	Efficient communication
	Job and task scheduling at platform level	Generalist and tailorable processes
Cons	Low control of resource management	Limited parallel abstraction
	Significant memory overhead	No native provenance or replication
	Poor integration with kernels	Steep learning curve
	Key-value only	
	Deep software and communication stack	

Table 4.7: Summary of the main features detected during the evaluation and literature review for the BD and the HPC paradigms.

particularities of the language and the available functionalities. The data-centric nature of Spark allowed us to exploit the parallelism of the workflow and get good experimental results as shown in Subsection 4.4.3. Due to the scale of our experiments, we could not detect network issues in MPI. However, we found that I/O management could become a bottleneck. At small scale, Spark showed a competitive advantage in distributing data and getting access to it locally.

The advantages described above could benefit scientists, who want to focus on the problem, rather than on the computational elements of the work. Although, these benefits come at the cost of large amounts of memory overhead. To cope with this drawback, it would be required to select machines with larger memory per core ratio. However, the cost will become more expensive.

Another issue worth mentioning is the infrastructure's size. It is common for small- and medium-sized research groups to have limited infrastructures. Hence, we consider our infrastructure to be representative of the capabilities of many small- and medium-sized research groups in science and data analysis. Even though it was not a large-scale evaluation due to the limited infrastructure size, it was still sufficient to detect major scalability issues and contrast it with larger experiments in the public cloud.

It is also important to mention that the original workflow relies upon two third-party kernels, which could be considered as black-boxes with unknown data schemes and proprietary code. Hence, the Spark implementation is based on the current MPI version of the workflow that might restrict the exploitation of the Spark platform to its full extent.

In general, we proved the viability of the enhanced methodology based on the results obtained. Though it is still far from being a universal solution to a vast set of complex workflows due to its limitations described above.

4.5 Summary

In this chapter, we evaluated a trend of combining the traditional high performance computing (HPC) with big data (BD) analytics paradigm [108]. Following that trend, we investigated

whether it is relevant to the scientific applications, especially those with many loosely-coupled tasks [43], or heterogeneous tasks with few interdependences [138]. Moreover, we overviewed the cloudification methodology proposed by Caíno-Lores et al. [33]. This methodology is directed at migrating the parameter-based scientific simulations to cloud environment and scaling them up. The critical step of this methodology is the identification of an independent variable for the entire simulation. Regarding the complexity of scientific simulations described in 3.3.3, we suggested to enhance the original cloudification methodology by enlarging its application to a greater number of applications, where it is not possible to identify an independent variable for the entire application. In order to evaluate the enhanced cloudification methodology, we applied it to the original EnKF-HGS application, which is an MPI implementation of a data assimilation process via the ensemble Kalman filter. In general, we proved the viability of the enhanced cloudification methodology based on the results obtained. However, it is still far from being considered as a universal solution due to its limitations.

Chapter 5

Service-oriented cloudification methodology

5.1 Introduction

According to Buyya et al. [32], computing was significantly transformed over the last half century. In 1969, in the UCLA press release announcing the birth of the Internet, Leonard Kleinrock [100] was quoted as saying: "As of now, computer networks are still in their infancy. But as they grow up and become more sophisticated, we will probably see the spread of computer utilities, which, like present electric and telephone utilities, will service individual homes and offices across the country". Basically, Leonard Kleinrock articulated the vision of computing as the 5th utility after water, electricity, gas, and telephony. Like the four existing utilities, the computing utility provides a general community with the basic level of computing services regardless where they are hosted or how delivered. To make this vision possible, several computing paradigms, i.e., cluster, grid, and cloud computing, have been proposed. Each aforementioned paradigm is presented in more detail in Sections 2.1 and 2.2. Overall, there is a set of characteristics that could specify every paradigm. This set covers such important concepts as resource availability, resource provisioning, scheduling, load balancing, and monitoring. For instance, in clusters, resources are located in a single administrative domain with a single management policy. While, in grids, resources are geographically distributed across multiple domains with distinct management policies and goals. Considering the scheduling concept, in clusters, schedulers are responsible for the entire system and directed at enhancing the system performance. Whereas, resource brokers – schedulers for distributed data-intensive applications on grids – focus on enhancing the performance of geographically distributed applications by matching available resources to the user's request. Cloud computing possesses all these characteristics with its special attributes and capabilities presented in Section 2.2, i.e., potentially unlimited scalability, on-demand resource provisioning, and multitenant environment. Considering the dynamic nature of cloud computing, there is a recurring and growing concern about building reliable mechanisms for the aforementioned concepts, i.e., resource availability, resource provisioning, scheduling, load balancing, and monitoring. Especially, it is the case when cloud services can be turned offline due to power conservation policies, unexpected power outages or possible denial of service attacks.

The service-oriented computing (SOC) paradigm arose as a response to the aforementioned concern. According to Papazoglou et al. [128], SOC is a computing paradigm that utilizes services as fundamental elements to support rapid, low-cost development of distributed applications in heterogeneous environments. SOC provides a semantically rich and

flexible computational model and relies upon service-oriented architecture (SOA) to develop, use, and combine interoperable services. The design principles of SOA require services to be self-contained, loosely-coupled, platform-independent, dynamically discoverable, invokable, and composable [70]. Basically, SOA splits software development in three independent but collaborative abstractions, i.e., a service requestor, a service broker, a service developer [154]. The service developer simply builds services. The responsibility of the service broker is to publish or market available services. While the service requestor is responsible for finding available resources through service brokers and using the developed services. Overall, this allows software developers to build a complex application consisting of easily customizable modules through interfaces with clearer semantics [82]. According to Channabasavaiah et al. [37], organizations could gain distinct benefits from deploying SOA, specifically:

1. **Continuous business-process improvement** due to service characteristics, i.e., coarse granularity, modularity, loose coupling, platform independence.
2. **Leveraging existing assets** means that a service can be constructed by using already existing components, applications. Using a service will only require users to know its interface and name.
3. **Faster time-to-market** means that time to build and deploy new services diminishes drastically, when existing services and components are reused.
4. **Risk mitigation** is gained due to the reduced possibility of failures in a new service if it is based on leveraging the existing components.
5. **Reduced cost** of application development and integration.

Even though SOA is a promising concept, which can bring considerable benefits, the migration of traditional vertically-integrated applications to SOA presents a number of challenges ranging from purely technical issues to adoptability constraints. According to Mahmood [111], the greatest challenge of migrating applications to SOA is associated with its inherent characteristics. Considering the coarse granularity of SOA services, it may become difficult to test and validate all combinations of service components under every possible condition. Even though loose coupling makes a service distributed and fault-tolerant, it also leads to a new level of complexity. The most common implementation of SOA is by using web services to exchange messages over the Internet. Taking into account SOA requirement for interoperability, the current implementation might raise a critical concern associated with the integration of services in a heterogeneous environment.

Considering all the advantages presented above, service-oriented computing paradigm could provide an alternative way to create an architecture of a scientific application that reflects components' autonomy, granularity, and heterogeneity. When combined with clouds, it can surely be regarded as a possible alternative execution environment for traditional HPC-based scientific applications. However, it is also important to take into account potential challenges, which come from the nature of scientific applications and fundamental differences in such concepts as resource availability and provisioning, workload management, scheduling, load balancing, and monitoring for cloud computing and HPC environments.

5.2 State of the Art

Scientific software development can be substantially influenced by two areas: cloud computing presented in Section 2.2 and service-oriented computing (SOC) introduced in the previous

section. SOC provides a solid computational model, which enables the academia to reuse existing data- and compute-intensive applications, develop new ones, and combine them into interoperable services. While cloud computing promises to provision scientists with unlimited computing resources (e.g., hardware, software, networking, storage) as a service. However, it could be challenging to migrate traditional scientific applications to clouds as presented in Chapter 3. These challenges come from the nature of scientific applications as well as the fundamental differences of cloud computing and HPC environments. The challenges of migrating scientific applications to clouds include architectural differences, software and hardware incompatibilities, tight coupling, and dependencies on external libraries. While fundamental differences of cloud computing and HPC environments are associated with resource availability and provisioning, workload management, scheduling, load balancing, and monitoring. Specifically, in this section, we cover various academic mechanisms and industry solutions directed at overcoming these differences.

According to Daniel J. Abadi [14], one of the cloud computing advantages is elasticity and scalability of computing resources. For instance, unlimited number of computational resources can be allocated to a scientific simulator on the fly to handle an increased demand in computational power. However, getting and managing additional resources is not as simple as it might be expected. The challenge here is that scientific applications shall be able to manage additional resources provisioned on demand. For example, in a traditional cluster environment, there is a resource management system consisting of a resource manager and a job scheduler, which manages the workload by preventing jobs from competing for limited computing resources. Traditionally, the job scheduler communicates with the resource manager on workload queues and makes scheduling decisions based on resource availability. When migrated to clouds, the scientific application might need an additional system to manage resource availability and provisioning.

Computing resource management is a vast research area, which has been investigated from different perspectives. There is a wide range of scientific works that created models and mechanisms to deal with resource provisioning, utilization, and job scheduling. We have only selected few studies related to the topic of resource management. For instance, Van et al. [119] presented a virtual resource management solution for service hosting providers that automates dynamic provisioning and placement of virtual machines and supports heterogeneous hosted applications and workloads. Considering the fixed size of virtual resources provided, Lim et al. [104] investigated a problem of building an effective controller of resource provisioning for dynamic applications hosted in clouds. They recommended the proportional thresholding approach to control resource provisioning by using a dynamic target range, which adjusts according to the number of accumulated virtual machine instances.

Niu et al. [121] introduced an alternative solution for academic, research, or business organizations, which use fixed capacity clusters, to use and dynamically resize a cloud-based cluster with a familiar batch scheduling interface. They proposed a semi-elastic cluster (SEC) computing model. The model allows organizations to maintain a set of cloud instances, schedule jobs within the current capacity, flexibly adjust the capacity level according to the application workload, responsiveness requirement or service provider charges. The advantage of the SEC model is an integration of dynamic cluster size scaling with a common HPC batch job scheduling strategy. Niu et al. proved the feasibility of the SEC model by integrating it with SLURM, an open-source resource management solution for Linux-based clusters. According to Yoo et al. [174], SLURM (Simple Linux Utility for Resource Management) is a solution for cluster management and job scheduling. The SLURM Workload Manager [6] is used by many of the TOP500 supercomputers (e.g., IBM BlueGene/L, IBM Purple) and

academic communities [9]. It has such characteristics as simplicity (easy to understand the source code and modify it with optional plugins), open source (free and available under the GNU General Public License), portability (initially written for Linux but also portable to other systems e.g., FreeBSD, NetBSD, OS X, Solaris), scalability (initially designed for clusters with 1'000 nodes), fault tolerance (handling failures without terminating workloads), security and user-friendliness. Overall, SLURM is a flexible industry solution widely used in academia to investigate challenges associated with resource provisioning, workload management, and scheduling.

The problem of load balancing in clouds has also been widely studied. In cloud environment, load balancing research area is directed at finding mechanisms to enhance the overall performance by efficiently distributing workload across available cloud nodes. Proper load balancing in clouds can help to optimally use available resources, avoid under- or over-provisioning of resources to jobs, and reduce response time. According to Al Nuaimi et al. [16], existing load balancing mechanisms can be classified as static or dynamic algorithms.

Among the available static load balancing algorithms, it is worth starting with the round robin. Firstly, this algorithm was introduced by John Nagle [116] for fair queuing of packet switches with infinite storage. The round robin algorithm uses only the network parameters of incoming traffic for traffic allocation without taking into consideration information from other components, e.g., the current load of an application, server. In general, the round robin algorithm pairs an incoming request to a specific server, virtual machine (VM) by cycling through the list of available servers, VMs capable of handling the request. Radojević et al. [136] suggested the Central Load Balancing Decision Module algorithm (CLBDM), which aims to improve the round robin. The goal of the CLBDM is to monitor all components of the computer system and automatically administer traffic allocation. CLBDM monitors client sessions including all the requests sent and received, calculates the connection time between the client and server, if the connection time exceeds the threshold, terminates the connection, and forwards the task to another server. Overall, CLBDM acts as an automated administrator. Several static load balancing mechanisms were developed according to the ant colony optimization (ACO), which applies models of ants' collective intelligence to various combinatorial optimization and control problems. According to Sim et al. [147], in network routing and load balancing, ACO is typically implemented as a small program, which represents an artificial ant. Migrating from node to node, the program communicates the shortest path to resources, updates routing information, and records the number of ants/incoming traffic that pass the node. Overall, static algorithms can produce reliable results in homogeneous and stable environments and assign tasks to nodes according to the node ability to process new requests.

Dynamic algorithms are more flexible than static ones and, hence, better suitable for heterogeneous and dynamic environments. For instance, the Index Name Server algorithm (INS) proposed by Wu et al. [171] is directed at minimizing data duplication and redundancy on a cloud storage. Due to extensive coverage and domains of cloud computing-based architecture, it is common that the number of files stored multiplies, when the number of nodes increases. Consequently, that could lead to a heavy workload and waste of hardware resources. INS calculates the optimum selection point by assessing many parameters (e.g., hash code of the block data, transition quality) and workload of the connection (availability to handle additional nodes). Whereas Wang et al. [165] proposed the Load Balance Min-Min (LBMM) scheduling algorithm, which takes as a foundation the Min-Min scheduling algorithm directed at minimizing the completion time of all tasks. The LBMM goal is to split a task into some number of subtasks and distribute them among service managers that will

assign subtasks to suitable service nodes of the shortest execution time. In general, dynamic algorithms assign tasks according to network bandwidth and various node attributes, which are gathered and calculated in real time. Because dynamic algorithms require constant node monitoring, they are more accurate but harder to implement.

Another important research area is monitoring, which is essential for efficient management of large-scale cloud resources. Monitoring of physical and virtual resources can help to identify and solve issues associated with resource availability, virtual machine's capacity to handle a workload, and other quality requirements. Monitoring can be defined as a process, which identifies the root cause of an event by collecting the required information at the lowest cost in order to determine the state of a system [62]. Fatema et al. [62] examined existing monitoring systems and classified them into two categories: general purpose tools (e.g., Nagios, Collectd, Zabbix) and cloud specific monitoring tools (e.g., Amazon Cloud Watch, Azure Watch, Nimsoft). They proposed to evaluate the efficiency of these tools according to the desirable cloud monitoring capabilities, i.e., scalability, portability, non-intrusiveness, robustness, multi-tenancy, interoperability, customizability, extensibility, shared resource monitoring, usability, and affordability. In general, Fatema et al. [62] emphasized that general purpose monitoring tools can be efficiently used in cloud environment, however, at the same time, they may not suit well for the full cloud operational management, including resource and application monitoring. Cloud specific monitoring tools are mainly designed for monitoring in cloud environment, hence, they can better address the desirable cloud monitoring capabilities. Interesting observation from Fatema et al. [62] is that cloud specific monitoring tools are mainly commercial and vendor-dependent, consequently, they are less portable and affordable in comparison with open source general purpose monitoring tools.

Overall, every aforementioned challenge is well studied. There are distinct academic mechanisms and well functioning industry solutions to all of them. However, when we apply some suitable solutions to one computing system, the system's complexity increases drastically. Kephart and Chess [98] addressed this complexity issue by presenting an autonomic model capable of dealing with large, complex, heterogeneous cloud-based systems. According to Kephart et al. [98], autonomic system comprises four distinct concepts:

1. **Self-configuration** means that systems configure themselves automatically by following high level policies.
2. **Self-optimization** means that systems continuously evolve and improve its performance and efficiency.
3. **Self-healing** means that systems are able to detect, diagnose, and solve problems resulting from bugs or failures.
4. **Self-protection** means that systems are self-protecting from malicious attacks and able to prevent and mitigate internal risks.

The autonomic model proposed by Kephart et al. [98] continuously executes four phases, i.e., monitoring, analysis, planning, and execution. That allows the system to learn and continuously improve its performance. In theory, this model addresses well all the challenges mentioned above, adds a new level of abstraction to the system and, consequently, copes with the complexity problem. Although, there is not much evidence of this approach being widely implemented in current commercial solutions. Weingärtner et al. [168] addressed this issue by proposing an autonomic distributed management framework, which can implement all four phases of the autonomic model by using cloud orchestration platform CloudStack (an open source cloud operating system). This framework was designed to act on the cluster level for

different purposes, i.e., migration of VMs to reduce networking overhead, balance of workload among hosts of a cluster, SLAs management or as a VM consolidation tool. The framework natively integrated with CloudStack has proved its feasibility by showing promising results during test cases. There is one inherent limitation, i.e., it was initially designed to be only a part of CloudStack. Because it is very platform-specific, its further integration with other cloud operating systems – e.g., Helion Eucalyptus, OpenStack, OpenNebula, Amazon EC2 – is limited.

Therefore, in the following section, we will propose a methodology, which can address the issue of cloud orchestration and allow us to migrate traditional HPC-based scientific applications, i.e., Monte Carlo simulations, to clouds managed by various cloud operating systems – e.g., OpenStack, CloudStack, OpenNebula, or Amazon EC2.

5.3 Cloudification methodology for Monte Carlo simulations

As discussed in Subsection 3.3.2, there are four main simulation problem types, three of which (i.e., equation-based, agent-based, and multiscale simulations) are deeply studied in terms of execution in both HPC and cloud environments. These three types of simulations are most commonly used and well adopted by the academic community. The forth type of simulations – Monte Carlo – is far less trivial to implement as not every system can be simulated by this method. Moreover, it introduces an extremely high demand for computing power when combined with an expensive and long-running individual sampling procedure. It is even more the case when the Monte Carlo method is implemented in an iterative framework like the ensemble Kalman filter. Considering cloud computing potential discussed in Section 2.2, we introduce a CPU resource utilization-optimized cloudification methodology for iterative Monte Carlo simulations.

5.3.1 Methodology description

One of the most popular computing paradigms for cloud environment is the service-oriented computing (SOC) introduced in Section 5.1 that heavily relies on service-oriented architecture (SOA) principles. SOC requires a workflow to be composed of invocations of self-contained, loosely-coupled, and potentially replaceable services. Hence, the application could be maintained and upgraded with less effort. Moreover, this also allows the application to better scale horizontally if multiple invocations of the same service are performed in parallel. However, in some cases, the end user might restrain the pool size of available computing resources in order to balance between the application execution time and operational expenses. Then there are two available options – either to run several service invocations sequentially or enable resource oversubscription. The former option might result in a poor resource utilization if the number of sequential invocations is assigned incorrectly for each individual resource. While the latter will most certainly result in an unpredictably degraded performance of the whole application.

By the definition, Monte Carlo simulations run a large number of identical but completely independent processes. We can see the Monte Carlo sampling processes as independent invocations of the same core service, which is called with different input parameters. The key idea of this methodology is to allow an external scheduling service to set an optimal number of invocations to perform on each individual computing resource. This methodology will allow us

to optimize resource utilization and, respectively, improve the overall application performance, i.e., execution time. Similarly to the big data inspired cloudification methodology described in Section 4.3, this methodology also consists of two main steps: the analysis of an original application and the actual cloudification of that application. These steps are presented in more detail below:

1. **Application analysis**: at this step an expert should analyze the Monte Carlo simulation workflow, and carefully examine the individual sampling procedure. The cloudified application would show a noticeable performance gain in terms of the execution time if and only if two important conditions presented below are fulfilled:

   (a) Execution time of every individual sampling process t_i varies with different input parameters provided;

   (b) The average execution time t_{avg} of individual sampling processes is much greater than the cloudification overhead t_{ov}, which comes from the paradigm change. This condition is noted as $t_{avg} \gg t_{ov}$.

2. **Cloudification**: If and only if these two conditions are fulfilled, the application is considered as suitable for cloudification. The cloudification step can be split into three following substeps:

   (a) **Workflow decomposition,** this operation separates the individual sampling procedure from the simulation workflow and encapsulates it into an independent service, i.e., individual sampling service. Consequently, the procedure is executed on different computing resources though the service invocation, while the main workflow only retrieves the execution results of the individual sampling service.

   (b) **Scheduling algorithm selection,** an expert should select a suitable scheduling algorithm based on additional information about the individual sampling procedure, e.g., process execution time, priority, additional conditions.

   (c) **Simulation,** this operation runs the simulation workflow. When the individual sampling procedure should be executed, the individual sampling service is called instead. It runs the procedure and returns the results back to the main workflow, which resumes its execution.

Figure 5.1 depicts the application of the cloudification methodology to an abstract Monte Carlo simulation. The workflow starts with an abstract initialization service, where the input data is read and global data structures are initialized. Then the prephase service generates input parameters for individual sampling processes. After that, the scheduling service distributes all executions of the individual sampling service over available computing resources. When all sampling service instances finish their execution, the results are aggregated in the postphase service and postprocessed. The next step is very specific for iterative workflows. It is the analysis service, where the current iteration results are evaluated, then either the process starts again or the execution stops with the termination service.

In order to apply the cloudification methodology to a concrete iterative Monte Carlo simulator, i.e., EnKF-HGS, there is a number of additional requirements, which should be satisfied. In particular, the execution environment should have such features:

1. Easy and straightforward access to an external scheduling service;

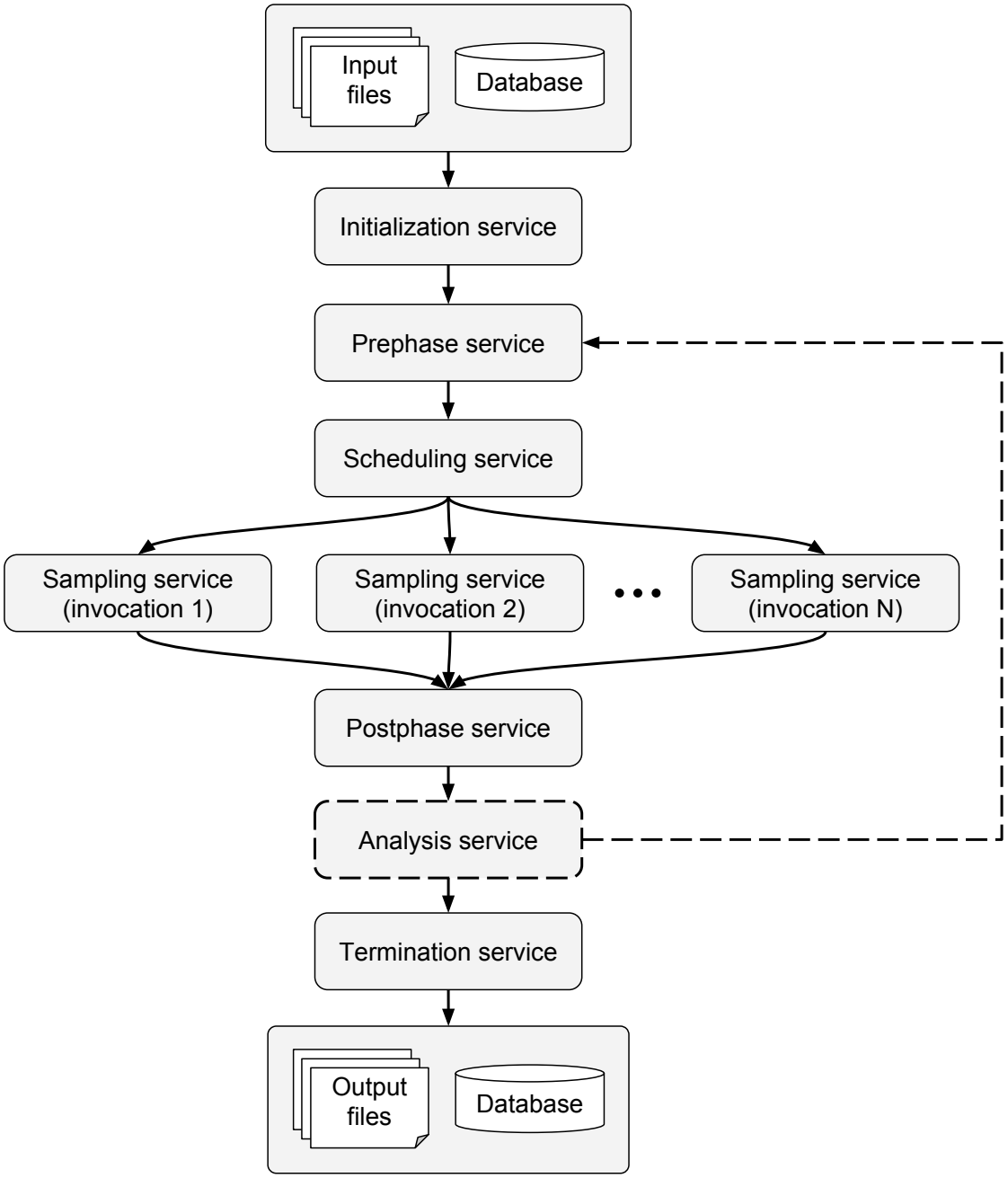


Figure 5.1: Overview of the service-oriented cloudification methodology applied to an abstract Monte Carlo simulation. Dashed boxes and arrows indicate optional stages and transitions, which may not be necessary for every Monte Carlo simulation.

2. Available encapsulation mechanism, which would enable a quick organization of a subworkflow into a self-contained service;
3. Simple procedure to expand and shrink the pool of computing resources.

For instance, a platform, which can provide the EnKF-HGS simulator with the aforementioned features, is the CLAUDE-building platform described in the following subsection.

5.3.2 CLAUDE-building platform

We present a lightweight, modular, and highly extensible platform for building cloud-based computing-oriented services (CLAUDEs). The platform allows the user to construct highly configurable computing-oriented cloud services, which are able to individually address each requirement listed above.

5.3.2.1 Motivation

Considering all the advantages of cloud computing presented in Section 2.2, there is an immense potential for the scientific community in migrating traditional HPC-based scientific applications to clouds. Clouds can be considerably cheaper than supercomputers, more reliable than grid computing, and much easier to scale than clusters. Taking into account SOC distinct benefits presented in Section 5.1, the potential value of migration of a traditional scientific application to clouds combined with an application of the SOC principles can be remarkably high. Numerous studies presented in Section 5.2 have examined this question by targeting various aspects and characteristics of clouds. Different solid solutions have been elaborated and suggested to solve specific cloud tasks, i.e., resource availability, resource provisioning, scheduling, load balancing, and monitoring. However, it is worth mentioning that these solutions can be considerably different and often incompatible. Moreover, a scientist or researcher who applies these solutions to a specific computing system, may face several challenges, specifically:

1. What solution(s) to choose when many of them have an overlap in features?
2. How to make the solutions compatible?
3. How to cope with an increasing computing system's complexity?
4. How to reduce a possibility of system's failure when integrating external solutions?

To deal with these concerns, we propose a CLAUDE-building platform, which will serve similarly to the autonomic distributed framework suggested by Weingärtner et al. [168]. The clear advantages of the CLAUDE-building platform are its compatibility with different cloud operating systems, e.g., OpenStack, OpenNebula, Amazon EC2, as well as modularity, extensibility, and relatively easy integration with external solutions.

5.3.2.2 Platform description

Two core principles of the platform are: (i) modularity and (ii) extensibility. This means that the resulting service should consist of multiple small and self-contained modules with clear and not overlapping functionality rather than be a single monolithic piece of complex software. In the context of CLAUDE-building platform, we call these modules – *service applications*, which communicate with each other over *communication protocols*. Respecting the platform core principles allows us to build extensible and highly configurable SOC-oriented cloud services by modifying and/or reusing existing service applications.

Abstract service application

Abstract service application is an essential building block of the CLAUDE-building platform, which lays a foundation for all service applications. For instance, an internal organization of a simple service application may be realized through the state pattern implementation.

In this pattern, each state would represent a particular piece of application logic, while state transitions would be triggered by either incoming/outgoing protocol commands or internal application's events. At the higher level, a service application represents a loosely-coupled and self-contained piece of service logic that can communicate with other applications over communication protocols. At the lower level, it comprises multiple core elements: (i) application layer, (ii) system layer, and (iii) messaging gateway.

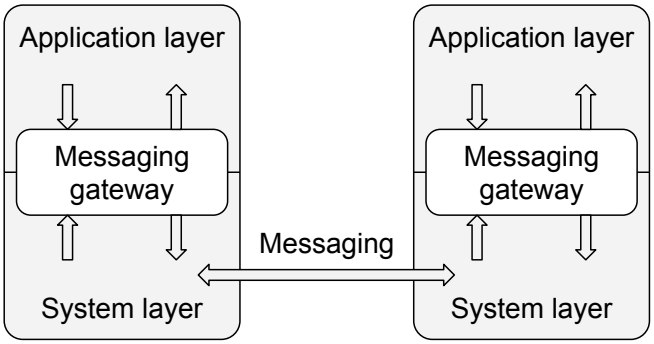


Figure 5.2: Application communication model.

Figure 5.2 illustrates a general application communication model. In this model, all internal application logic lays at the application layer, while all communication and message routing logic is at the system layer. The messaging gateway represents a unified communication interface between two layers.

A more detailed view of the system layer is presented in Figure 5.3. There we can distinguish multiple lower level components (e.g., channel endpoint, system router, endpoint proxy). However, their presentation is beyond the scope of this thesis.

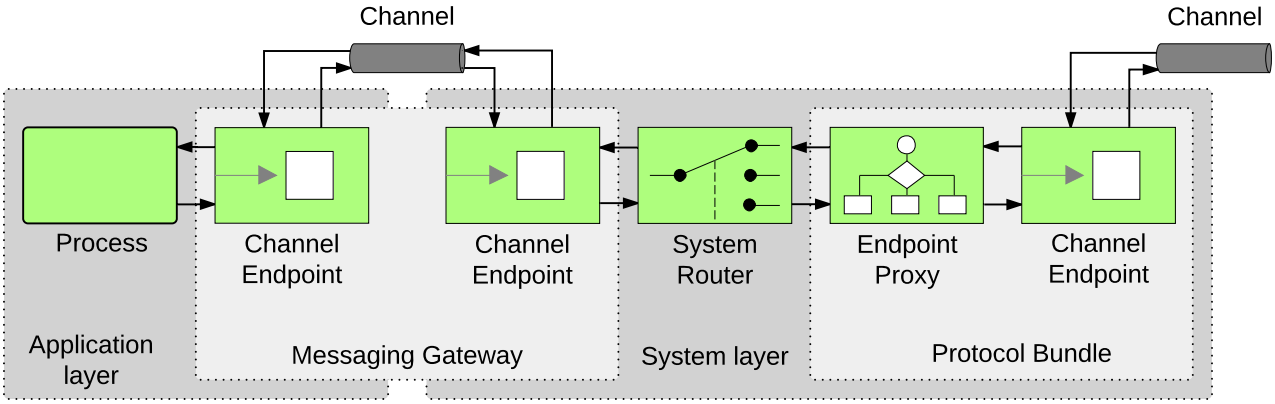


Figure 5.3: Detailed application communication model.

Abstract communication protocol

Abstract communication protocol is the second essential building block of the CLAUDE-building platform. It aims to provide a foundation for developing standardized communication channels between two or more service applications. The resulting communication protocol may implement one of three available communication patterns: (i) One-to-One, (ii) One-to-Many, (iii) Many-to-One, or a combination of them. From the architectural point of view, a communication protocol consist of three core elements: (i) channel, (ii) channel endpoint,

and an optional (iii) endpoint proxy that may contain partial protocol handling logic in order to automate handling of certain protocol commands at the system layer (see Figure 5.3).

Discovery group

Discovery group is a special CLAUDE-building platform concept that helps to combine service applications by a certain application aspect (e.g., type of application). It aims to group a set of service applications in order to automatically propagate updates of a changing application's parameter within the group. There are three types of parties defined in a discovery group: (i) spectators – applications, which desire to receive group's updates, (ii) members – applications, which hold the changing parameter, and (iii) server – an application, which maps spectators and members and propagates parameter updates. Additionally, a discovery group performs as an application discovery service, which facilitates the discovery of application's information unknown beforehand (e.g., service application IP address).

CLAUDE discovery protocol (CLAUDE-DSCVRY protocol)

CLAUDE-DSCVRY protocol was developed to operate as a standardized communication channel for the aforementioned discovery group parties. It inherits features of the parameters propagation from One-to-Many abstract communication protocol and features of the parameters monitoring from Many-to-One abstract communication protocol. From the conceptual point of view, CLAUDE-DSCVRY protocol implements a publish-subscribe messaging pattern with an intermediary information collection point and an integrated hearbeating mechanism. The goals of CLAUDE-DSCVRY protocol are:

1. To allow service members to join a server's discovery group;
2. To allow service spectators to subscribe on updates of a server's discovery group;
3. To automatically detect and propagate any availability or internal parameter changes of members in a server's discovery group;
4. To have a built-in authentication mechanism for the discovery group access control using Simple Authentication and Security Layer (SASL) [115].

Overall, CLAUDE-DSCVRY protocol allows service developers to concentrate more on the actual functionality of the service rather than on necessary but routine features like data propagation mechanism or security layer implementation.

5.3.2.3 The CLAUDE-0 architecture

As a proof of concept, we have designed and implemented a cloud-based computing-oriented service called CLAUDE-0 by using building blocks of the CLAUDE-building platform described above. This service allows us to cloudify the EnKF-HGS simulator using the previously presented cloudification methodology. Following the platform principles and general SOA guidelines, we defined a set of service applications in such a way that each application encapsulates only a specific part of the service logic. Thus, it enables modifications or replacement of any application without making changes to others. In order to do that, first, we derived a set of service functionality and reliability requirements:

Requirement 1: The end user should be able to control the computing resource pool size.

Requirement 2: The end user should be able to submit software input, request software execution, and then retrieve software output.

Requirement 3: The end user should be able to monitor software execution progress.

Requirement 4: The service should automatically distribute and schedule software for execution.

Requirement 5: The service should automatically detect any software execution failures and perform re-execution if necessary.

Requirement 6: The service should automatically detect any service application failures and be able to recover from them.

Requirement 7: The service should automatically deny an unauthorized access to service applications.

In order to fulfill these requirements, we relied on core platform building blocks, i.e., abstract service application, abstract One-to-One communication protocol, discovery group, and CLAUDE-DSCVRY protocol. Using these primitives, we developed seven service applications and connected them with six application layer communication protocols (see Figure 5.4):

Client application is the end user interface to interact with a service. Through client application, user performs available operations, e.g., requests software execution, monitors running software, controls resource pool size. At the application layer, client application interacts with job manager application over **CLI-TO-JM** protocol and resource controller application over **CLI-TO-RC** protocol.

Resource controller application controls the pool of computing resources. It can expand or/and shrink the resource pool by spawning or/and terminating VMs through an Amazon EC2 endpoint. At the application layer, resource controller application interacts with client application over **CLI-TO-JM** protocol and pushes configuration commands to resource daemon application over **RC-TO-RD** protocol.

Job manager application keeps track of all user requests for software execution and persists changes of the software execution status. In terms of communication with other applications, at the application layer, job manager application interacts with client application over **CLI-TO-JM** protocol and job scheduler over **JM-TO-JS** protocol. Also, job manager application receives updates of the software execution status from resource daemon application over **RD-TO-JM** protocol.

Job scheduler application is responsible for scheduling software execution on available computing resources. At the application layer, job scheduler application receives new software execution requests from job manager application over **JM-TO-JS** protocol, and pushes the actual execution commands to resource daemon application over **JS-TO-RD** protocol.

Resource daemon application runs the software and monitors its execution status. At the application layer, resource daemon application receives configuration commands from resource controller application over **RC-TO-RD** protocol and software execution commands from job scheduler application over **JS-TO-RD** protocol and reports the software execution status updates to job manager application over **RD-TO-JM** protocol.

Resource monitor application is a registry for available computing resources. It keeps track of running resource daemon applications and reports any status changes.

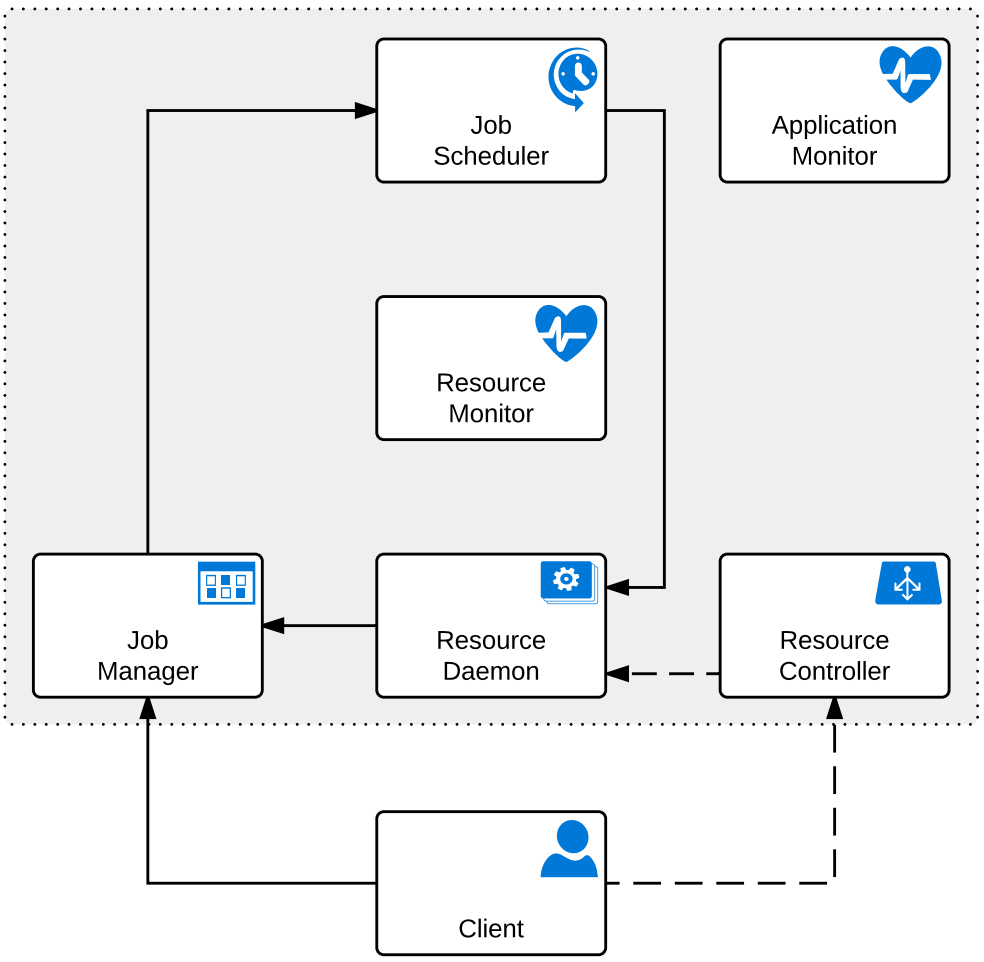


Figure 5.4: CLAUDE-0 applications and their interconnection over the application layer communication protocols. Arrowheads show the connection initialization direction, while dashed lines indicate currently not implemented protocols.

Application monitor application is a registry for service applications. It monitors resource monitor, job scheduler, job manager, and resource controller applications and reports any status changes.

At the system layer, we organized applications into two individual discovery groups by using CLAUDE-DSCVRY protocol. The first discovery group is named application discovery group. Its members and the server form the CLAUDE-0 core. The second discovery group is named resource discovery group. Its members form the pool of computing resources. Two discovery groups and applications that compose them are presented in Table 5.1, while Figure 5.5 illustrates the interconnection of applications.

Such organization of service applications allows us to fulfill all seven functionality and reliability requirements presented above, in particular:

Requirement 1: The end user monitors running VMs by receiving status updates from resource monitor application over CLAUDE-DSCVRY protocol. While, resource controller application gives full control over resources through CLI-TO-RC protocol.

	Spectators	Members	Server
Dscvry Group 1	Client Job manager Job scheduler Resource controller	Job manager Job scheduler Resource controller Resource monitor	Application monitor
Dscvry Group 2	Client Job scheduler Resource controller	Resource daemon	Resource monitor

Table 5.1: CLAUDE-0 discovery groups and their comprising applications. *Dscvry Group 1* corresponds to application discovery group, while *Dscvry Group 2* corresponds to resource discovery group.

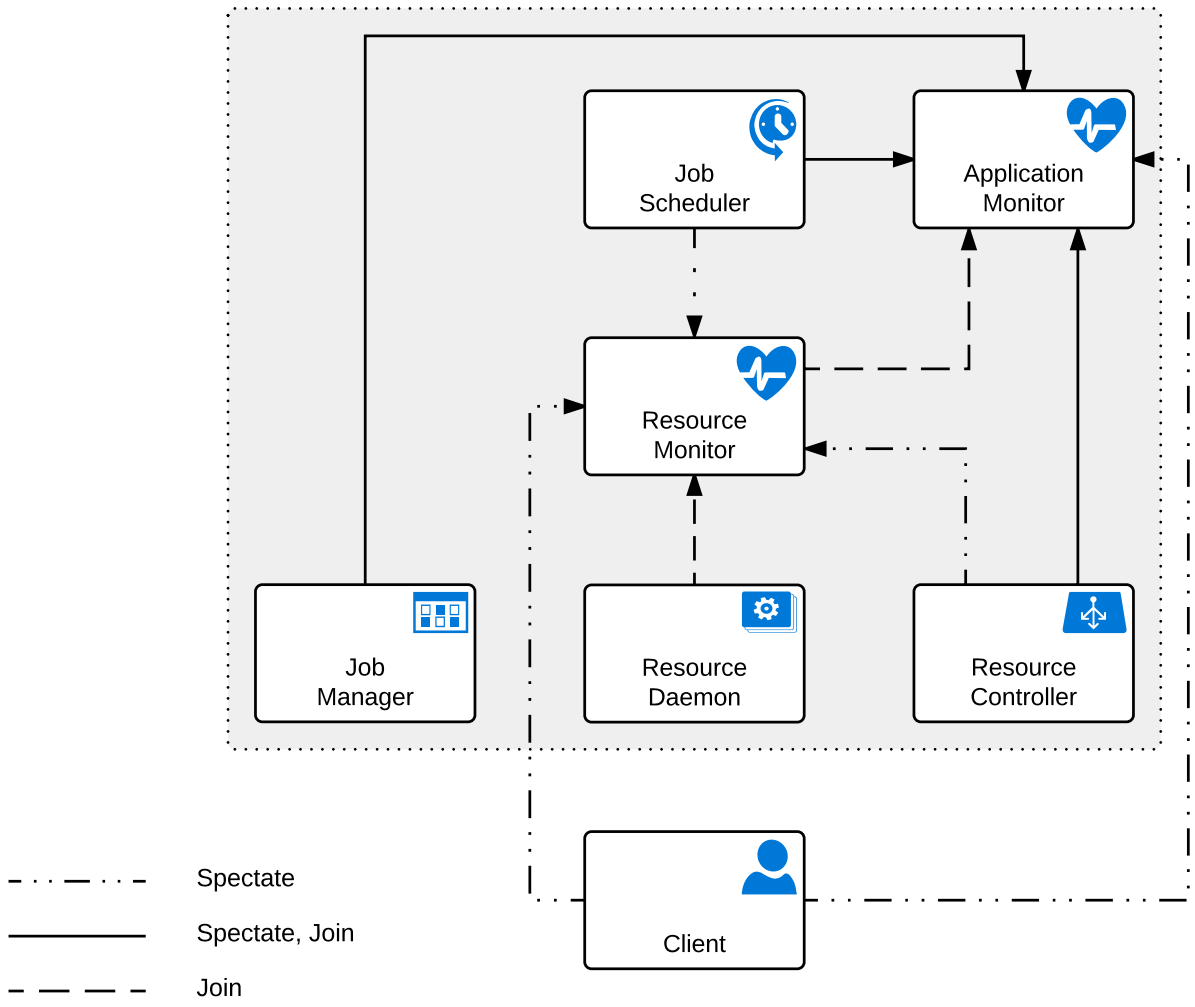


Figure 5.5: CLAUDE-0 applications and their interconnection over the CLAUDE-DSCVRY protocol. Arrowheads show the connection initialization direction.

Requirement 2 and 3: Job manager application and CLI-TO-JM protocol give full control over the software execution life cycle as well as enable the execution history retrieval through run/terminate/monitor operations.

Requirement 4: Job scheduler application is responsible for scheduling and distributing software execution over JS-TO-RD protocol. It also automatically receives status updates of the available resources from resource monitor application over the CLAUDE-DSCVRY protocol.

Requirement 5: Job manager application receives software execution status changes from resource daemon application over RD-TO-JM protocol. It also receives resource daemon application status changes from resource monitor application over CLAUDE-DSCVRY protocol. In case of a failure, job manager application is able to reschedule either the failed software or the software, which was running on the failed resource.

Requirement 6: Application monitor application reports any status changes of service applications over CLAUDE-DSCVRY protocol. In case of a service application failure, all other applications will be informed automatically and can react accordingly.

Requirement 7: CLAUDE-DSCVRY protocol has a built-in authentication mechanism through SASL. This allows the user to implement very quickly any desired authorization / authentication system.

In the end, CLAUDE-0 covers key computing environment concepts like monitoring, scheduling, resource provisioning, persistence, fault-tolerance, protocol design, and distributed communication. The modular service design and replaceability of service applications allows us to enhance or simply experiment with each of the concepts separately. Overall, it provides a perfect testbed environment for further research activities.

5.3.2.4 Implementation and deployment

As described in Subsection 2.2.1, cloud computing offers SaaS as a major service delivery model. SaaS may provision applications that use a web browser or predefined application program interfaces (APIs) as their main communication interface. CLAUDE-0 also implements a typical provisioning model for applications delivered as SaaS. Overall, CLAUDE-0 is an easily migratable SaaS-based service, which is compatible with a wide range of cloud platforms (e.g., Amazon EC2, OpenNebula, OpenStack). Moreover, CLAUDE-0 allows its users to benefit from the features of cloud environment for ordinary, non-interactive, computation-intensive applications commonly used in the high performance computing domain. Both CLAUDE-building platform and CLAUDE-0 are implemented in Python and rely on ZeroMQ framework as a communication medium for scalable and high performance distributed messaging. Basically, to deploy CLAUDE-0 we need a physical or virtual machine for the installation of the CLAUDE-0 core, access to a cloud data storage that supports the Amazon S3 protocol for input/output data storage, and access to a cloud resource pool through an Amazon EC2 endpoint (see Figure 5.6). Moreover, it allows the end user to minimize and adjust both the execution time by exploiting parallel execution (e.g., for Monte Carlo applications) and the amount of consumed resources, consequently, the final costs.

From the user's perspective, CLAUDE-0 has been specifically designed in a very user-friendly way. By using CLAUDE-0, the user can easily access and launch the software without any prior knowledge how to run applications on clouds. Once the CLAUDE-0 core is installed and configured, an operating system (OS) image for the virtual machine (VM) shall

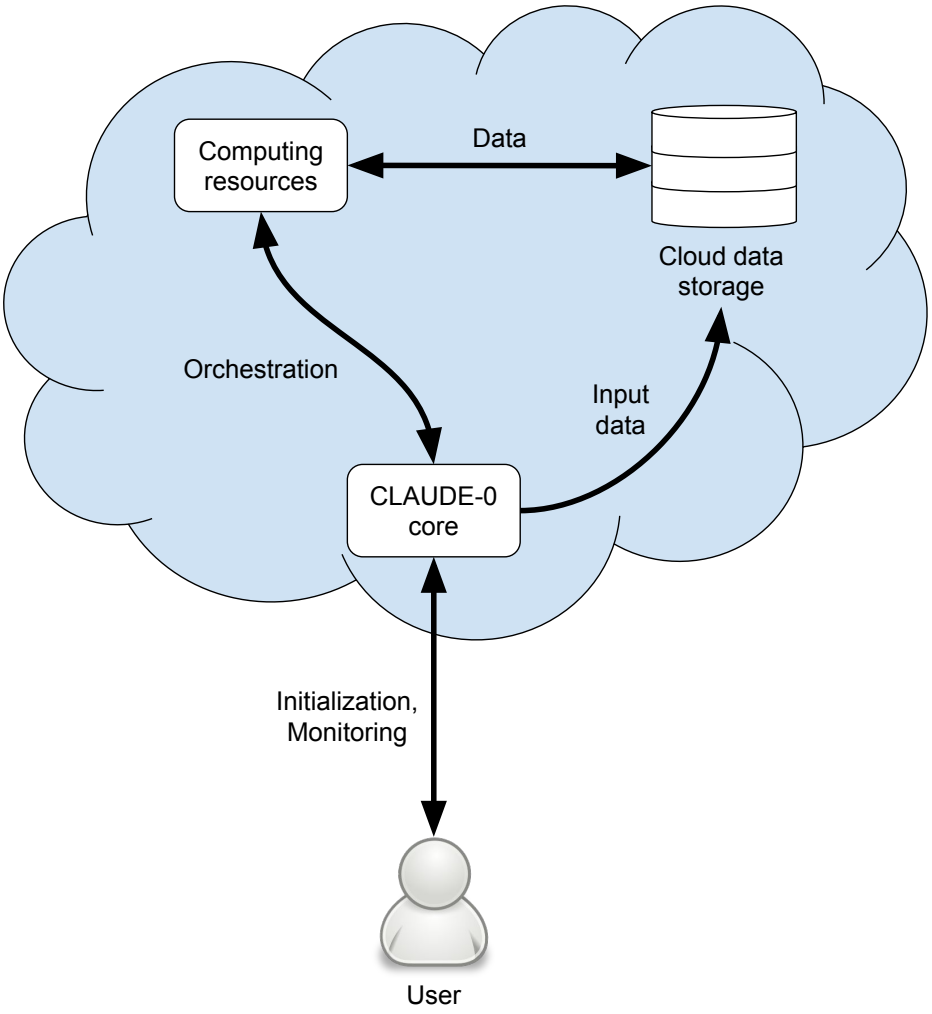


Figure 5.6: CLAUDE-0 deployment model.

be prepared. Then the required software and resource daemon application shall be installed. After the OS image shall be uploaded to the cloud repository. The uploading process may slightly differ from one cloud platform to another. Although, most popular providers (e.g., Amazon EC2, OpenNebula, OpenStack) have step-by-step tutorials explaining the entire process. Additionally, the user is expected to provide the CLAUDE-0 core with a Python script that defines software execution parameters and input/output locations. When all these steps are completed, the system is ready for execution. To minimize necessary changes of the original application source codes, CLAUDE-0 provides the user with a considerable advantage – a set of communication drivers, which implement communication operations in the cloud. The CLAUDE-0 drivers interact with the CLAUDE-0 core through the same application layer communication protocols as client application. Therefore, CLAUDE-0 makes it easy to port an existing application, which was not designed for a cloud environment, to clouds.

5.4 Application to EnKF-HGS

Following the service-oriented cloudification methodology described in Section 5.3 and using CLAUDE-0 presented in Subsection 5.3.2, we proved its viability by applying it to the original EnKF-HGS simulator. As described in Section 3.4, EnKF-HGS combines an iterative Monte

Carlo method with an integrated hydrological modelling software HydroGeoSphere. This combination allows the simulator to continuously improve the deviating numerical model through a data assimilation process. In order to deal with an extremely high demand for computing power, original EnKF-HGS implementation uses the Message Passing Interface (MPI) framework for the workload distribution.

5.4.1 Simulator analysis

EnKF-HGS implements the data assimilation process through the ensemble Kalman filter, which is an implementation of the Monte Carlo method. In this case, individual sampling processes of the general Monte Carlo method correspond to the ensemble members (i.e., individual model realizations) of the ensemble Kalman filter. Starting with the analysis step of the methodology, there are two conditions, which should be fulfilled in order to consider the application suitable for the methodology.

The first condition of the methodology requires the execution time of every individual model realization t_i to vary when provided with different input parameters. In order to verify whether the EnKF-HGS simulator fulfills this condition, we executed it with an input model and the following parameters: 144 method iterations, 100 individual model realization. Figure 5.7 presents the results of the execution and shows the maximal, average, and minimal execution times on each iteration. We can clearly see the variation of execution times, thus, we consider the condition to be fulfilled.

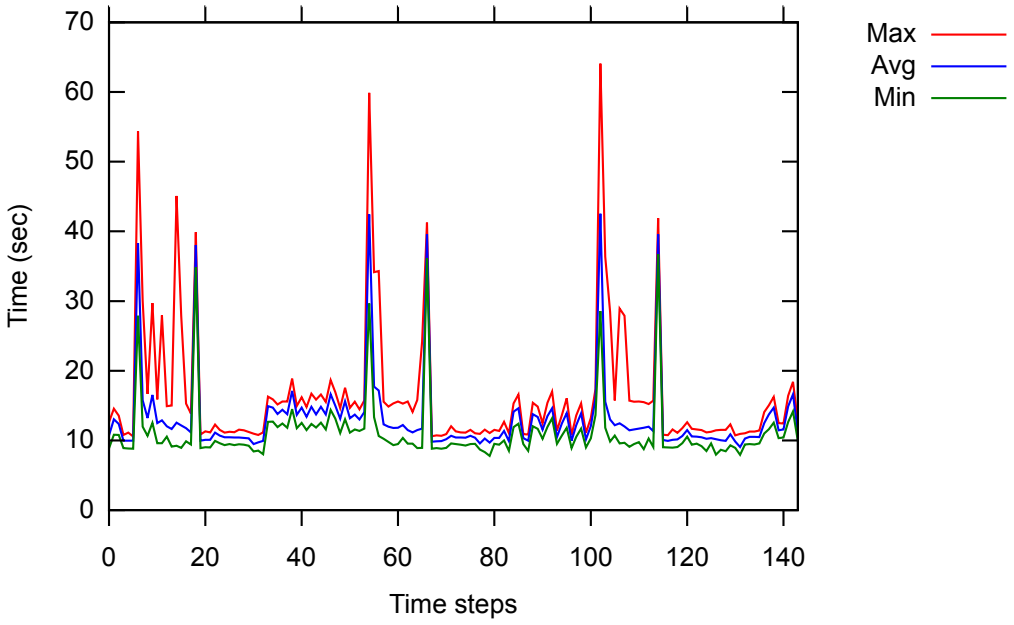


Figure 5.7: Variation of the execution time of individual model realizations.

The second condition of the methodology requires the average execution time t_{avg} of individual model realizations to be much greater than the cloudification overhead t_{ov}. Otherwise, the benefits of the methodology will be neglected by the accumulated overhead. This condition heavily depends on the performance of the external scheduler, which in our case is CLAUDE-0, and much less on the simulation workflow itself. At this moment, CLAUDE-0 cannot show an outstanding scheduling performance as it is only in a proof of concept state. However, its extensibility and modular architecture would allow the user to improve the

necessary components without affecting the whole service. We discuss this in more detail in Subsection 5.4.3.

5.4.2 Cloudification procedure

Execution models: CLAUDE vs. MPI

In order to port EnKF-HGS to clouds, the CLAUDE-0 driver was integrated into the program source code and all the direct execution calls of HGS were replaced with the CLAUDE-0-based counterparts. This cloudification procedure changes the initial MPI execution model of EnKF-HGS. However, before looking at the final execution model, it is important to understand the general CLAUDE-0 execution logic. Figure 5.8 depicts interactions between CLAUDE-0 components presented in Subsection 5.3.2. The CLAUDE-0 components are identified with Roman numerals (from I to IV), while a number (1) refers to the user's software executed in the cloud.

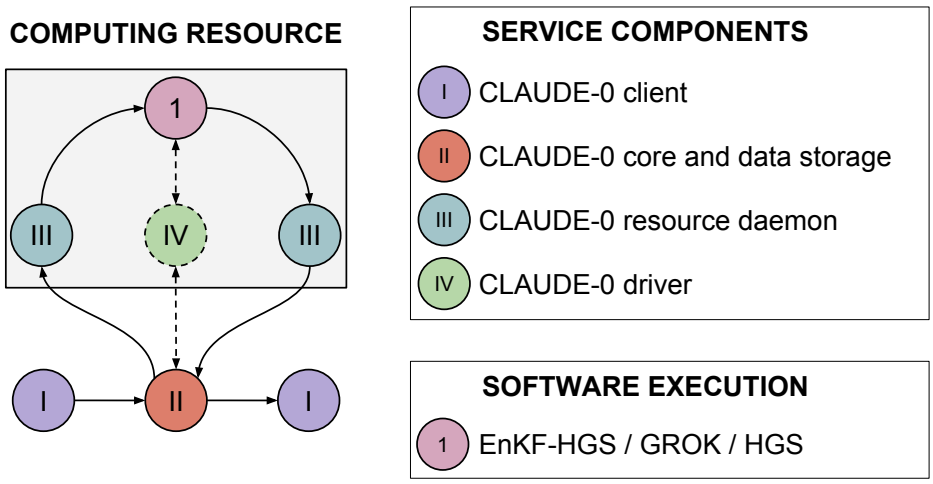


Figure 5.8: CLAUDE-0 components invocations.

Execution request

I. Request submission

Through client application, the user sends a software execution request to the CLAUDE-0 core.

II. Scheduling

When CLAUDE-0 core receives the software execution request, it finds an available computing resource according to the internal scheduling policy and forwards the request to the computing resource.

Software execution

III. Execution preparation

Resource daemon application receives the execution request, prepares the execution environment, downloads the input, and then launches the requested software.

1. Software execution

One of the three available software binaries is executed, i.e., EnKF-HGS, GROK, or HGS, on the previously created input directory.

IV. Subworkflow execution

In case of EnKF-HGS, the CLAUDE-0 driver is called for a subworkflow execution. The driver interacts with the CLAUDE-0 core similarly to client application. The execution continues from stage **II. Scheduling** of the **Execution request** block. While the driver waits for the execution termination signal in order to continue the execution of EnKF-HGS.

III. Clean up

When the software execution is finished, resource daemon application cleans up the execution environment, uploads the output, and sends a signal to the CLAUDE-0 core.

Results retrieval

II. Status update

The CLAUDE-0 core receives the execution termination signal from resource daemon application and forwards it to the originator of the execution request. (In case of EnKF-HGS subworkflow execution, the originator of the request is the CLAUDE-0 communication driver.)

I. Execution results

The user receives the execution termination signal through client application.

The cloudified EnKF-HGS separates the main EnKF-HGS simulation loop from the HGS instances, which run on remote computing resources. For this purpose, the CLAUDE-0 driver transmits the relevant input for each HGS model realisation to the cloud data storage, and requests the CLAUDE-0 core to execute corresponding simulations on available cloud computation resources. When all the simulations are completed, the CLAUDE-0 driver retrieves the relevant output from the cloud data storage. Then data is returned to EnKF-HGS, which continues the data assimilation loop.

Figure 5.9 depicts the final execution model of the cloudified EnKF-HGS. The important ENKF-HGS workflow steps are identified with numbers (1 to 3), the auxiliary steps are identified with letters (A to B), while CLAUDE-0 driver invocations are identified with a Roman numeral (IV).

Ensemble Kalman filter simulation

A. Initialization / Data distribution

The first step is to read input files, initialize global data structures, and generate realizations specific files in separate directories. This step combines the initialization and data distribution steps of the original workflow.

IV. Model simulation (GROK & HGS)

Instead of running an ensemble of model simulations, CLAUDE-0 driver requests the CLAUDE-0 core to execute them on available computing resources and waits for the results.

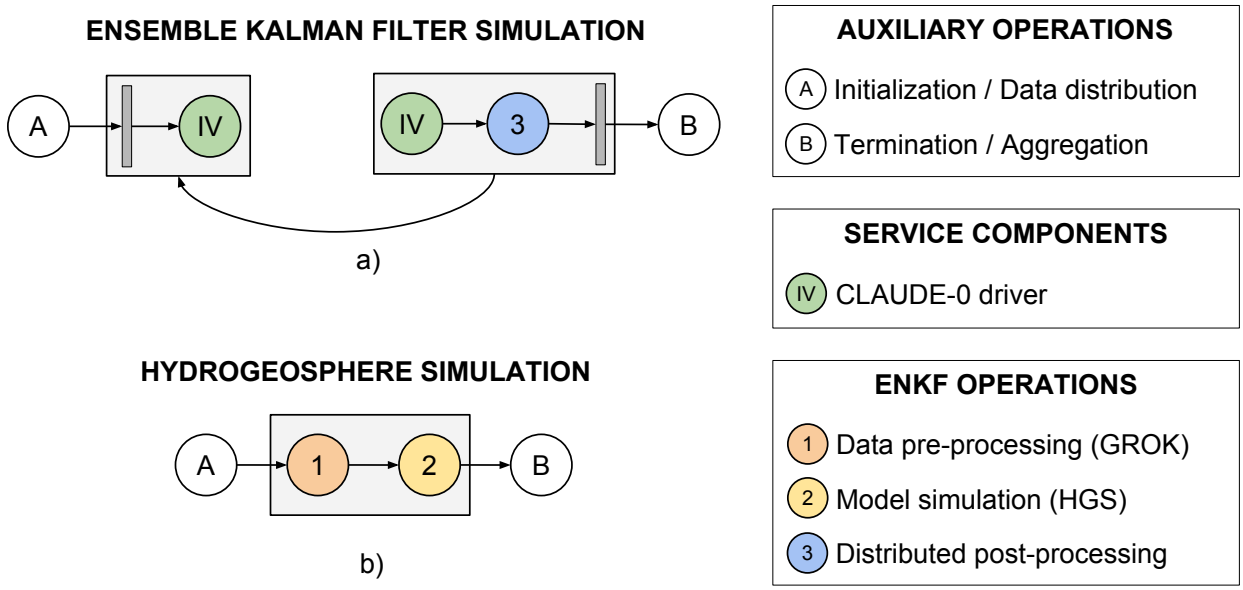


Figure 5.9: Execution models of cloudified a) EnKF-HGS and b) HydroGeoSphere.

IV. Ensemble simulation completed

When all individual simulation results are retrieved, EnKF-HGS workflow execution continues.

3. Distributed post-processing

Filtering phase of the ensemble Kalman Filter workflow is executed similarly to the original EnKF-HGS workflow.

B. Termination / Aggregation

After all iterations are completed, the output is written to the disk and the program terminates. This step is similar to the one in the original EnKF-HGS workflow.

Hydrogeosphere simulation

A. Initialization

Individual model simulation does not need any additional initialization. All relevant input files are already extracted to the working directory.

1. Data pre-processing (GROK)

This step is identical to the corresponding step of the original workflow. GROK is executed in order to generate HGS-specific input files.

2. Model simulation (HGS)

This step is identical to the corresponding step of the original workflow. HGS reads the output of GROK, runs the model, and writes the model output.

B. Termination

Individual model simulation does not need any additional termination or clean up.

This service-oriented execution model allows us to dynamically adjust the amount of computing resources to momentary workload conditions. Moreover, one can benefit from an external

scheduling component for better resource utilization. Additionally, this leads to minimization of the network input/output (I/O) operation because all intermediate data files are locally stored on the computing resource.

5.4.3 Evaluation

The EnKF-HGS simulator combines a Monte Carlo simulation method with an iterative process. As a result, the simulated model is continuously improved. In every iteration, two essential phases can be identified: (i) simulation of an ensemble (i.e., a set) of model realisations (referred to as the forward propagation phase), and (ii) an update, or feedback, phase for state vectors (and optionally parameters) (referred to as the filtering phase). The forward propagation phase computes all individual model realisations of the ensemble. Computing a single model realisation requires executing two proprietary binaries, i.e., GROK and HGS. GROK is a pre-processor of HGS, which prepares the input files for HGS. The runtime of each HGS realisation strongly depends on input parameters and model complexity. Normally, HGS is a comparably long-running and compute-intensive process due to a high non-linearity of the hydrological processes that are simulated. Moreover, due to a large number of parallel model realisations in the forward propagation phase, the main demand for computing power in the EnKF-HGS workflow comes from the execution of HGS. In the cloudified version of the EnKF-HGS simulator, the forward propagation phase is, therefore, distributed over multiple cloud computing resources. On the other hand, the filtering phase remains centralized. The focus of this analysis is to evaluate the performance of both implementations (i.e., original and cloudified). First, we will assess the execution model performance of the original implementation. Then we will evaluate two essential phases mentioned above. We will present and discuss the execution results of the forward propagation and filtering phases. Then we will evaluate the cloudified implementation and overview the execution models' specifics, i.e., cloudification overhead and scheduling aspects. In the end, we will present the overall results of the methodology application to EnKF-HGS.

5.4.3.1 Experimental setup

In order to compare the performance of two implementations, we executed both versions of EnKF-HGS on a private cloud operated by OpenNebula. In the experiment, the model performs 144 iterations with an ensemble of 100 model realisations. Since all realisations are independent HGS instances, a set of 4 experiments with varying numbers of CPU cores was conducted: (a) 10, (b) 20, (c) 50, and (d) 100. In each experiment, one single CPU core was assigned per HGS instance. In the original MPI-based implementation of EnKF-HGS, this resulted in 10, 5, 2, and 1 HGS execution per CPU core, for (a), (b), (c), and (d) respectively. In the cloudified version of EnKF-HGS, CLAUDE-0 adaptively distributed the execution of the HGS instances over available computing resources for every iteration, by using the Round Robin algorithm, which improves resource utilization. The specifications of the testbed are described below.

Testbed: private cloud

For the MPI platform, we relied on (a) 2, (b) 3, (c) 7, and (d) 13 worker nodes. While for the CLAUDE-0 platform, we relied on two types of worker nodes (i) worker-1, and (ii) worker-2. One instance of the worker-1 node was dedicated to the EnKF-HGS core execution. While (a) 2, (b) 3, (c) 7, (d) 13 instances of worker-2 nodes were dedicated to the HGS simulator

execution as explained in Subsection 5.4.2. Regarding the network storage, each platform required a different type of a network storage. For the MPI platform, we deployed the latest version (3.7.14) of GlusterFS on 3 additional storage nodes as the application required a POSIX-compliant file system. The nodes were organized in a distributed volume with no data replication in order to maximize the storage performance. On the computing nodes' side, we exploited the FUSE-based Gluster Native Client for a highly concurrent access to the file system. While for the CLAUDE-0 platform, we deployed an S3-complaint Riak CS solution [1] on one additional storage node. Table 5.2 shows a summary of the selected nodes and their assigned roles in both CLAUDE-0 and MPI execution platforms. In addition, Table 5.3 describes the hardware characteristics of these nodes.

Platform	**CLAUDE-0**			**MPI**	
Node role	**worker-1**	**worker-2**	**storage**	**worker**	**storage**
Type	small	large	medium	large	medium
Amount	1	2/3/7/13	1	2/3/7/13	3

Table 5.2: Private cloud machine selection for the CLAUDE-0 and MPI platforms.

Type	**Processor**	**vCPU Cores**	**Memory**	**Storage**	**Network Performance**
small	Intel Xeon L5420 @2.50GHz	1	2.5GB	HDD 500GB	Ethernet 1Gb/s
medium	Intel Xeon L5420 @2.50GHz	4	3.5GB	HDD 500GB	Ethernet 1Gb/s
large	2 x Intel Xeon L5420 @2.50GHz	8	7.4GB	HDD 500GB	Ethernet 1Gb/s

Table 5.3: Technical specifications of the selected machines.

5.4.3.2 Performance analysis of the original implementation

As discussed before, the original EnKF-HGS was designed for execution in a traditional cluster environment, which normally provides: (i) a low latency broadband network connection and (ii) access to a high performance network file system. Violation of these assumptions might lead to a dramatic performance drop as it was illustrated in Subsection 4.4.3. The second important aspect of the MPI implementation is the statically assigned number of model realisations per MPI process. This is a typical design choice for an MPI-based application due to a lack of load balancing, e.g., through a built-in job-stealing mechanism. In an ensemble with individual simulations of heterogeneous duration, EnKF-HGS might result in a relatively poor performance. This happens because some MPI processes might finish all assigned simulations sooner than others, while EnKF-HGS shall wait for all individual realisations to finish before starting the filtering phase. Therefore, the CPU idle time during the forward

propagation phase was measured. Figure 5.10 shows the relative CPU idle time per iteration for the four experiments (a), (b), (c), and (d) described above. Heterogeneous execution time of realizations is additionally amplified by the I/O saturation problem described in Subsection 4.4.3 and has the following average results: (a) 8.87%, (b) 23.84%, (c) 23.09%, and (d) 20.37% of idle time for one or more CPU cores. Considering these numbers, it is evident that the statically assigned number of model realisations per MPI process results in a non-optimal utilization of CPU resources.

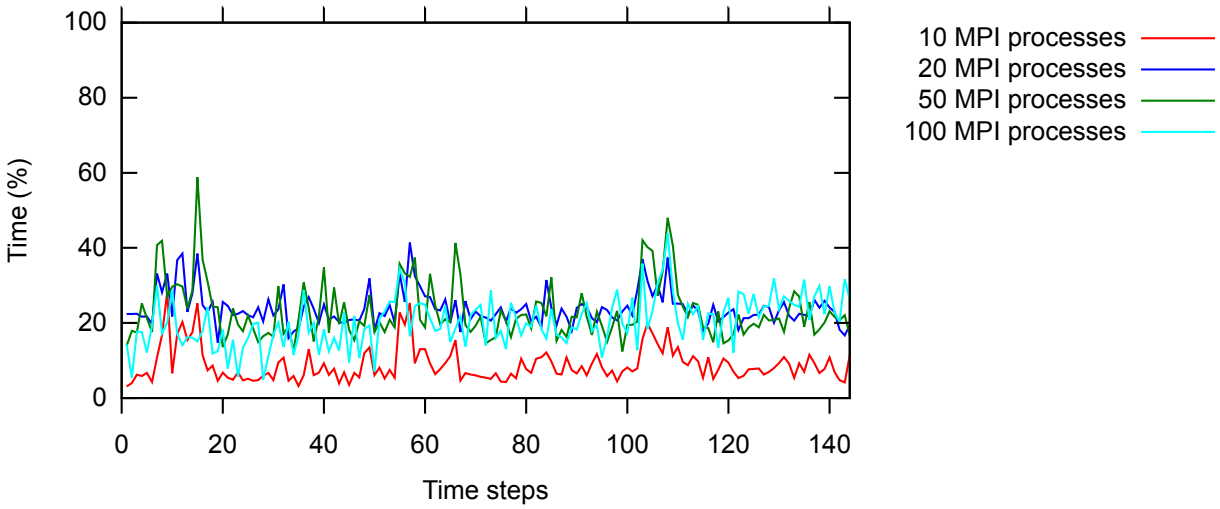


Figure 5.10: Relative CPU idle time per iteration.

5.4.3.3 Performance analysis of the forward propagation phase

In this experiment, only the duration of the forward propagation phase is considered. Figure 5.11 illustrates the computation time ratio between two implementations against the iteration step. In the original implementation, this time is defined as the elapsed time from the beginning of the forward propagation phase to the end of the longest running MPI process. However, in the cloudified version, it is a time interval between the beginning of the forward propagation phase and the end of the last finishing HGS instance. It is worth mentioning that the data transmission and cloud scheduling times shall also be considered in the cloudified version. The execution time of the forward propagation phase for two implementations is comparable. A small difference of 4-15% in favour of the original MPI-based approach can be observed in the experiments (c) and (d).

5.4.3.4 Performance analysis of the filtering phase

In contrast to the forward propagation phase, the filtering phase was not distributed, as it requires significantly less computing power and, thus, much shorter execution times. EnKF-HGS performs filtering on initially allocated system resources. In the original implementation, it includes the entire resource pool ranging from 10 to 100 CPU cores. In the cloudified version, only one CPU core is initially provided to the filtering step (see Table 5.2). The relative execution time ratios of the filtering phase of two implementations are shown in Figure 5.12. Surprisingly, in the experiment (d), the filtering phase running on 100 CPU cores is 20% slower than on a single CPU core. This difference in performance is a result of the network connection of a virtual cluster in the cloud environment. A regular physical cluster

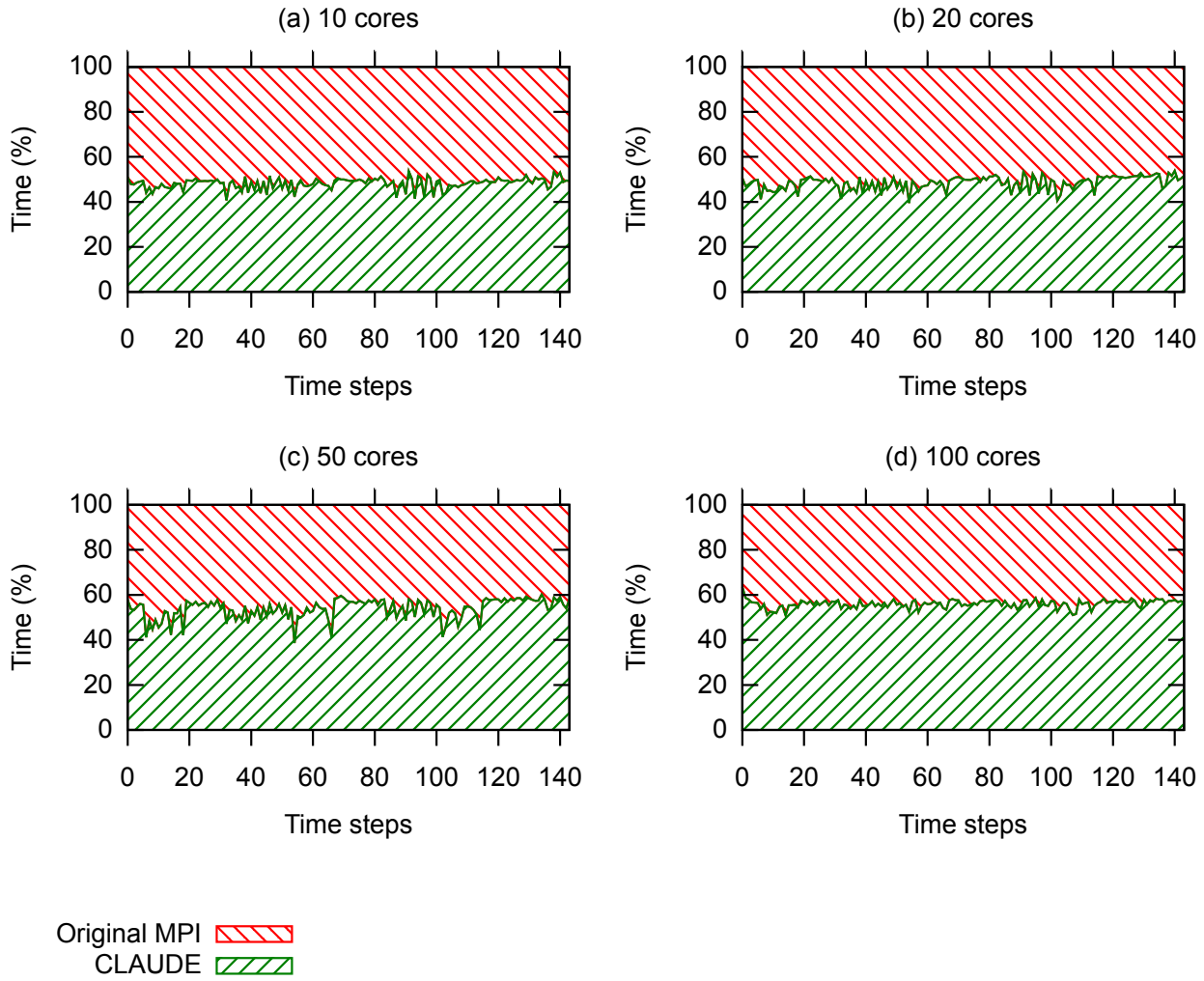


Figure 5.11: Relative computation time of the forward propagation phase for two different execution approaches.

normally guarantees high throughput and low latency connection. On a cloud, physical machines can be geographically distributed, hence, they might be connected with links of lower capacity and higher latency. In our experiments, cloud workers were connected with a regular 1 GbE-T Ethernet connection, which is considered slow from the point of view of a regular cluster environment. As a result, the data transmission time between multiple MPI processes surpassed the benefit of parallel computation. Therefore, the regular MPI-based EnKF-HGS setup is less suited for cloud computations with heterogeneous or low capacity resources.

5.4.3.5 Performance analysis of the cloudified version

The forward propagation phase in the cloudified EnKF-HGS accommodates three individual, time-consuming processes: (i) input/output (I/O) data transmission, (ii) scheduling, and (iii) HGS execution. In the initial four experiments, the duration of every process was measured. Moreover, we separated the worker initialization time (i.e., worker overhead) from the actual simulation time. Figure 5.13 shows the relative duration of all these processes. The time spikes in the simulation phase correspond to long-running realisations. Figure 5.14 shows the absolute execution time of the nonsimulation processes. Considering these two

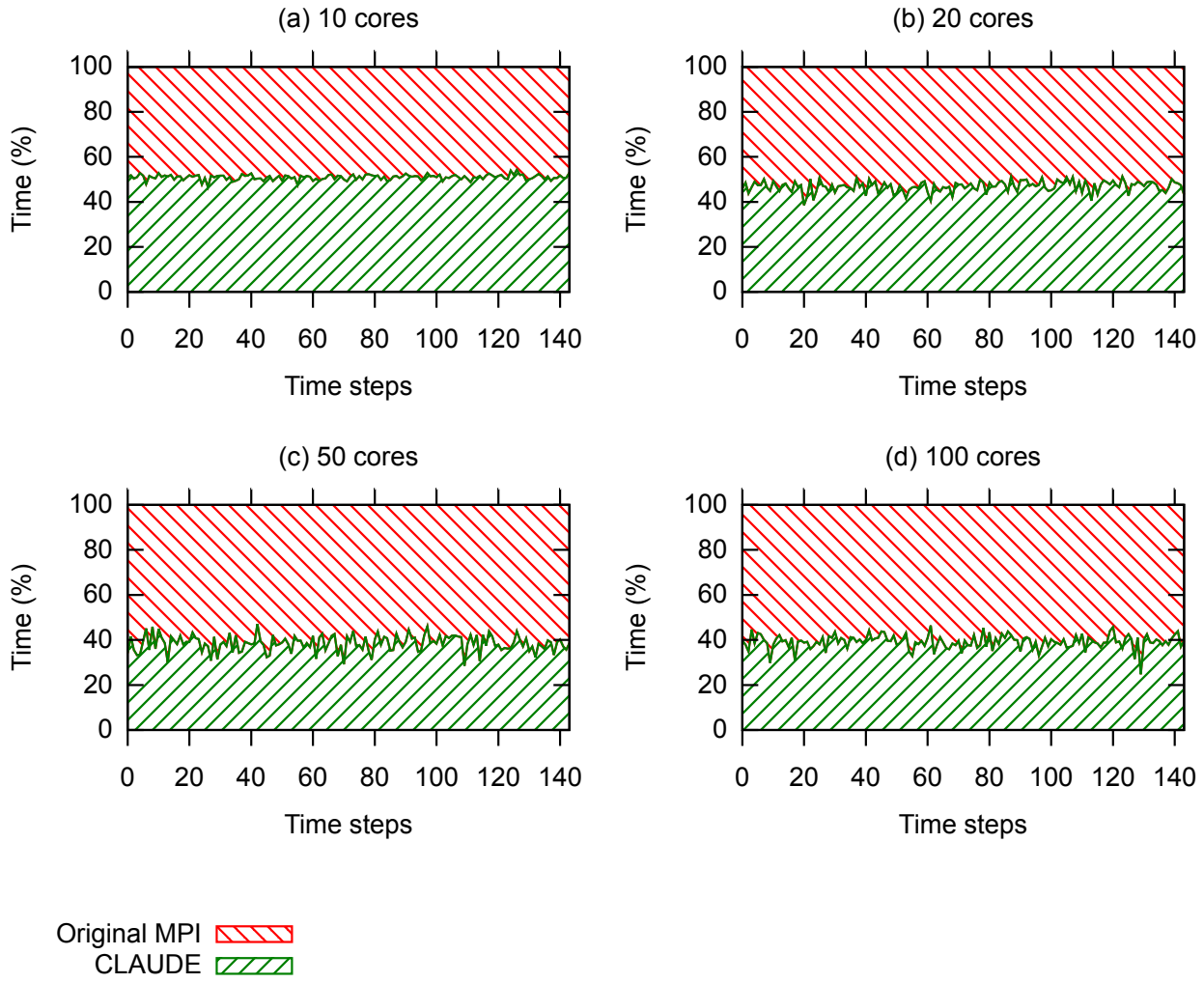


Figure 5.12: Relative computation time of the filtering phase for two execution approaches.

figures, the I/O data transmission time is almost constant for the full ensemble execution as well as for the number of available CPU cores. The total nonsimulation time slightly decreases with the growing number of available CPU cores but it is almost constant for a fixed amount of computing resources. The scheduling time grows with the number of available CPU cores, however, it is also almost constant for a fixed amount of computing resources. In practice, this allows parallel execution of complex hydrogeological models with long-running individual realizations in cloud environment, keeping a relatively constant duration of nonsimulation processes. While CLAUDE-0 is still in a proof of concept state, it can already indicate the benefits of the service-oriented execution approach with external scheduling and enhanced data locality techniques. By using an I/O-optimized implementation and a high performance scheduler, the total nonsimulation time can be significantly reduced. At the moment, the total nonsimulation time corresponds to 40% to 80% of the overall execution time for the given experiment.

5.4.3.6 Results of the methodology application to EnKF-HGS

Considering the features of cloud computing, i.e., on-demand resource provisioning and potentially unlimited scalability, there is a high potential for running compute-intensive simulations on clouds as there will not be any financial and personal overhead for acquiring and

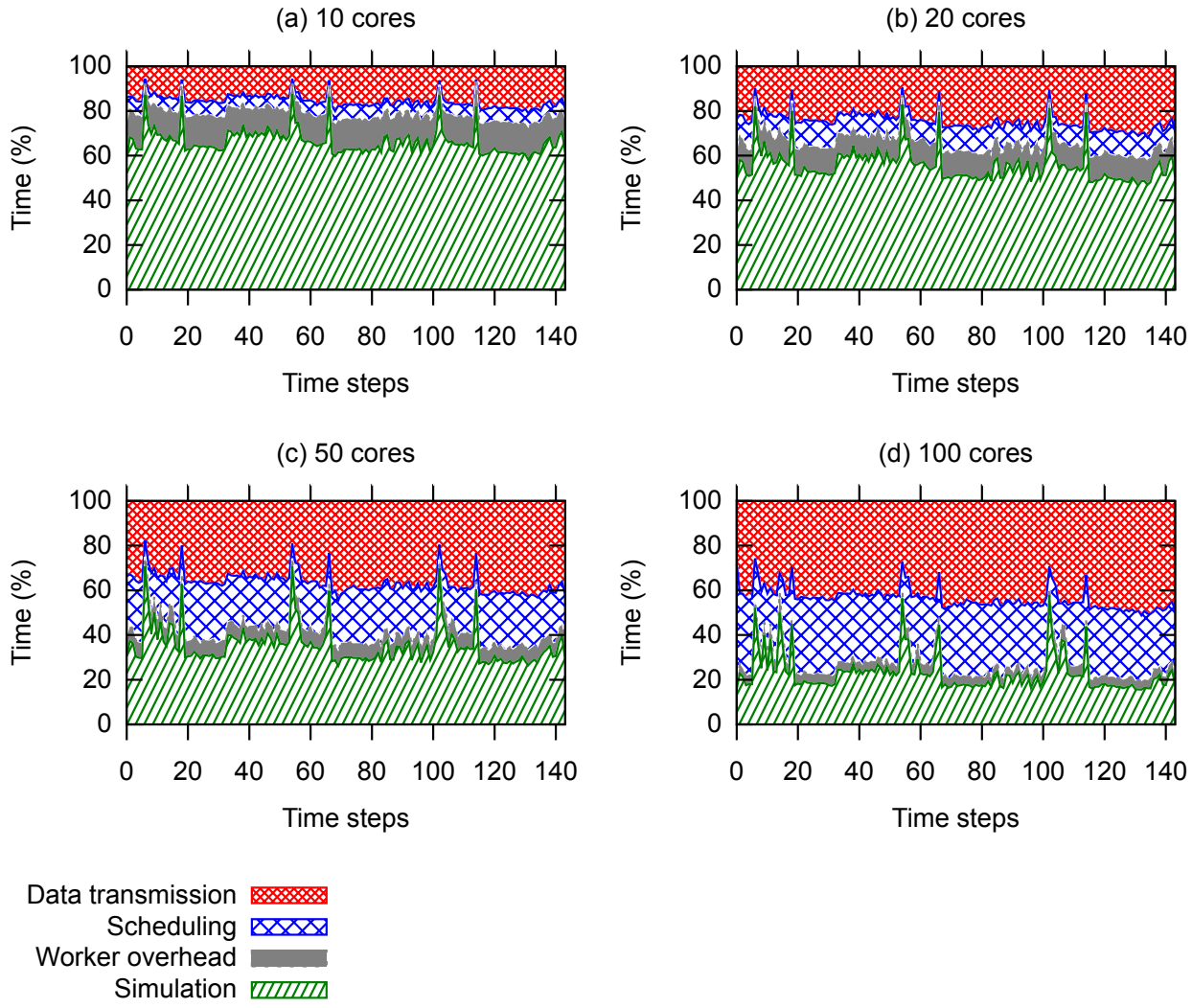


Figure 5.13: Processes involved in the forward propagation phase of the cloudified simulator.

maintaining the required computational infrastructure. The primary goal of this cloudification methodology is to make an iterative Monte Carlo simulation benefit from the optimal resource utilization and, hence, improve the overall application performance. To apply the cloudification methodology to a typical iterative Monte Carlo simulation, we have decided to migrate the cluster-based EnKF-HGS data assimilation system to clouds with minimal changes in the original simulator source code and a linkage to pre-existing management tools for a cloud-based execution. The implementation changes were relatively straightforward because the EnKF-HGS simulator implements a Monte Carlo-based data assimilation approach. Considering the definition of the Monte Carlo simulation method described in Subsection 3.3.2, every process is completely independent. Hence, the ensemble of model realizations can easily be distributed among available CPUs and nodes in a cloud infrastructure. Therefore, it provides an ideal environment for such tasks.

The performance of the EnKF-HGS simulator clearly shows that the cloudified implementation produces an affordable overhead with respect to data transfer and execution times. Considering cloud execution, the easily migratable cloud-based service CLAUDE-0 was developed to distribute the subworkflows over available cloud resources. With CLAUDE-0, we obtained a successfully cloudified version of EnKF-HGS, which executes comparatively well

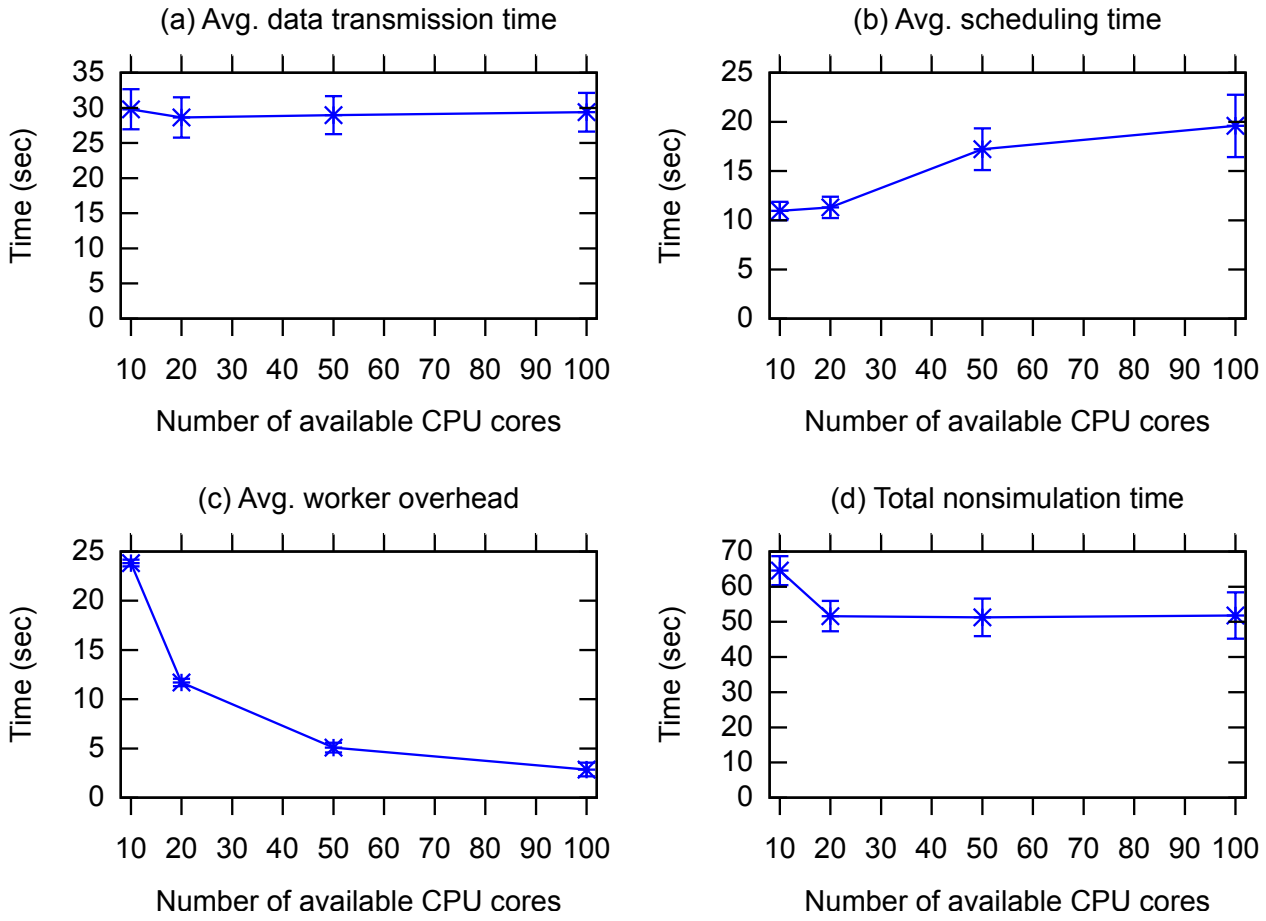


Figure 5.14: Absolute time of the nonsimulation processes.

and has a predictable service-related overhead. CLAUDE-0 provides us with such advantages as:

1. Access to external services for automatic job distribution, efficient job scheduling and optimal utilization of computational resources;
2. A well integrated data storage service.

Overall, CLAUDE-0 facilitates the EnKF-HGS application performance, eases the application development process, and lowers the demands for the target infrastructure. It is worth mentioning that for the proof of concept, the simulator was executed with a relatively simple model setup. This setup has considerably smaller problem size and execution time than a real-world EnKF-HGS simulation. With a real-world input model, the number of model grid cells could substantially increase the size of data packages, which shall be transferred in clouds. Consequently, that will lead to a longer data transmission time. However, at the same time, the simulation time for more complex real-world models will be considerably higher than for a simple model we used. Overall, it means that the ratio of data transfer to simulation execution time remains comparable with the increased complexity of input model.

5.5 Summary

In this chapter, we presented a vision of computing as the 5th utility, which provides a broad community of users – i.e., individuals, households, enterprises – with a basic level of computing services regardless of location or delivery means. A number of computing paradigms, i.e., supercomputers, clusters, grids, clouds, could deliver this vision. Even though each paradigm is different, all of them could be characterized through such important aspects as resource availability, resource provisioning, scheduling, load balancing, and monitoring. The service-oriented computing (SOC) has been introduced as a paradigm, which considers the dynamic nature of clouds and facilitates the development of reliable mechanisms to efficiently realize the aforementioned aspects. Altogether clouds and SOC were regarded as alternative paradigms for scientific community to gain potentially unlimited amount of resources on the fly, build and reuse applications with interoperable, loosely-coupled components. Considering potential challenges in migrating traditional scientific applications to clouds, we overviewed numerous studies. In Subsection 5.2, we presented various academic and industry solutions targeted at different aspects of the cloud computing environment – i.e., resource availability, resource provisioning, scheduling, load balancing, and monitoring. In Section 5.3, we proposed a methodology, which facilitates the migration of a specific type of scientific applications – iterative Monte Carlo simulations – to clouds. Moreover, we suggested a platform, which could provide a scientific application with an access to an external scheduling service, a reliable mechanism to organize application components into self-contained and independent services and an ability to resize the pool of computing resources. By using the primitives of the platform, we built CLAUDE-0, which is a lightweight and cloud platform-independent, modular and highly extensible computing-oriented service. In Subsection 5.4, we successfully applied the cloudification methodology to a typical iterative Monte Carlo simulator, i.e. EnKF-HGS, with the help of CLAUDE-0.

Chapter 6

Case study: hydrological real-time data acquisition and management in the clouds

6.1 Introduction

According to George Johnson [91], all science is computer science. Researchers from different disciplines consider computation as the third paradigm of scientific investigation, together with theory and experimentation. Because it could significantly contribute and enable further scientific advancements in various fields. Apart from being a research domain, computer science has a strong connection with diverse research disciplines, i.e., biology, physics, geology, hydrology, social and economic sciences. Computer technologies become indispensable when there is a need to model, design, simulate or/and build any complex system, e.g., from a hydrological environmental system to a skyscraper. Due to the high complexity of these systems, there is a number of challenges and difficulties, – e.g., real-time data collection and management, integration and orchestration of external heterogeneous services, modeling and deployment of simulations, retrieval of dynamic simulation predictions – which researchers solve by relying on specialized supporting software and high compute power. These difficulties are very common in the hydrological domain. For instance, hydrological environmental systems are highly heterogeneous in terms of physical characteristics and parameters. While simulating them, they heavily rely on complex multiscale non-linear processes and matrix operations. Furthermore, to provide good quality and accurate predictions, the computer-aided models of these hydrological systems integrate and process huge amounts of real-time data gathered from geographically distributed sensors. Often, it is also critical to tackle these hydrological computations in a timely manner.

Therefore, in collaboration with various universities and institutions, specifically:

1. Institute of Bio- and Geosciences (IBG-3): Agrosphere, Forschungszentrum Jülich GmbH;
2. Centre for High Performance Scientific Computing in Terrestrial Systems (HPSC-TerrSys), Geoverbund ABC/J;
3. Centre for Hydrogeology and Geothermics (CHYN) of University of Neuchâtel;
4. Communication and Distributed Systems (CDS) group of University of Bern;
5. Department of Earth and Environmental Sciences of University of Waterloo;
6. Aquanty Inc.

we developed a conceptual framework, which helps to solve problems associated with the complexity of dynamic hydrological and hydrogeological systems. This framework consists of two essential parts. The first part is a cheap, reliable, and portable solution for environmental data acquisition from remote locations, i.e., wireless mesh network (WMN) equipped with sensors. A real-time simulation platform deployed in clouds provides the second cornerstone of our conceptual framework.

6.2 Conceptual framework for cloud-based hydrological modeling and data assimilation

Our conceptual framework for cloud-based hydrological modeling and data assimilation offers two principal functionalities, i.e., (i) access to real-time measurement data, (ii) dynamic stochastic simulations, which are continuously improved by using the data assimilation approach. The first functionality comprises acquisition, transmission, and storage of measurement data in a hydrological database. While the second functionality is the stochastic real-time predictions of hydrological variables by the EnKF-HGS simulator presented in Section 3.4 and cloudified in Sections 4.4 and 5.4.

6.2.1 Data acquisition through wireless mesh networks

In Switzerland 80% of drinking water is taken from ground sources close to rivers. Because groundwater abstraction could substantially affect the water balance in river catchments, it is important to consider critical parameters of the river dynamics and the interactions of the river-aquifer system. A perfect example is the pre-alpine Emme river catchment (about 200 km^2) in the Emmental valley, where the groundwater is abstracted to provide Bern with roughly 45% of its drinking water. In fact, during low flow periods, groundwater abstraction often causes the river to dry up. The stream water levels in the upper Emme are strongly affected by seasonality and are highly sensitive to dry periods. The efficiency and sustainability of the Emme water supply management is directly interconnected with the amount of water pumped from the aquifer.

To optimize the pumping rates in a dynamic environment, a quantitative approach of the system simulation is required. Data assimilation can complete the optimization task by continuously incorporating real-time measurement data into the model simulation and correcting potential model biases or inaccuracies. The key components of the data assimilation systems are a communication network that provides field observations in real time; a data storage infrastructure; and numerical models that predict, for example, how groundwater abstraction schemes may affect the river baseflow. Regarding these simulations, the pumping rates can be regulated in an optimal way, thus providing a sustainable water supply management. The key goal of this case study is to build and deploy a cheap, reliable, and portable infrastructure for real-time data acquisition and collection.

6.2.1.1 Architecture and deployment

Climatic or hydrological systems are driven by highly dynamic forcing functions. Quantitative numerical frameworks such as simulation models are powerful tools to understand how these functions control the systems' response. However, models are always imperfect descriptions of reality and, therefore, model calculations increasingly deviate from the real physical conditions

of the simulated environmental system. To alleviate theses biases, the data assimilation technique could be used to integrate the field data into the modeling framework. In this case, constant monitoring of the concerned geographical area is required. The technology should provide high performance even in case of harsh meteorological conditions (snow, low temperatures, fog, strong winds, etc.), other location and infrastructure related limitations (high altitude, lack of access to the power grid), and limited accessibility (resulting in long access delays and inducing significant installation/maintenance costs).

In order to comply with the imposed requirements, a portable and robust data acquisition system was specified. Figure 6.1 presents its architectural overview, consisting of two components. Its major component is a Wireless Sensor Network for environmental monitoring while a cloud-based data collection gateway serves as an interface for accessing the collected data.

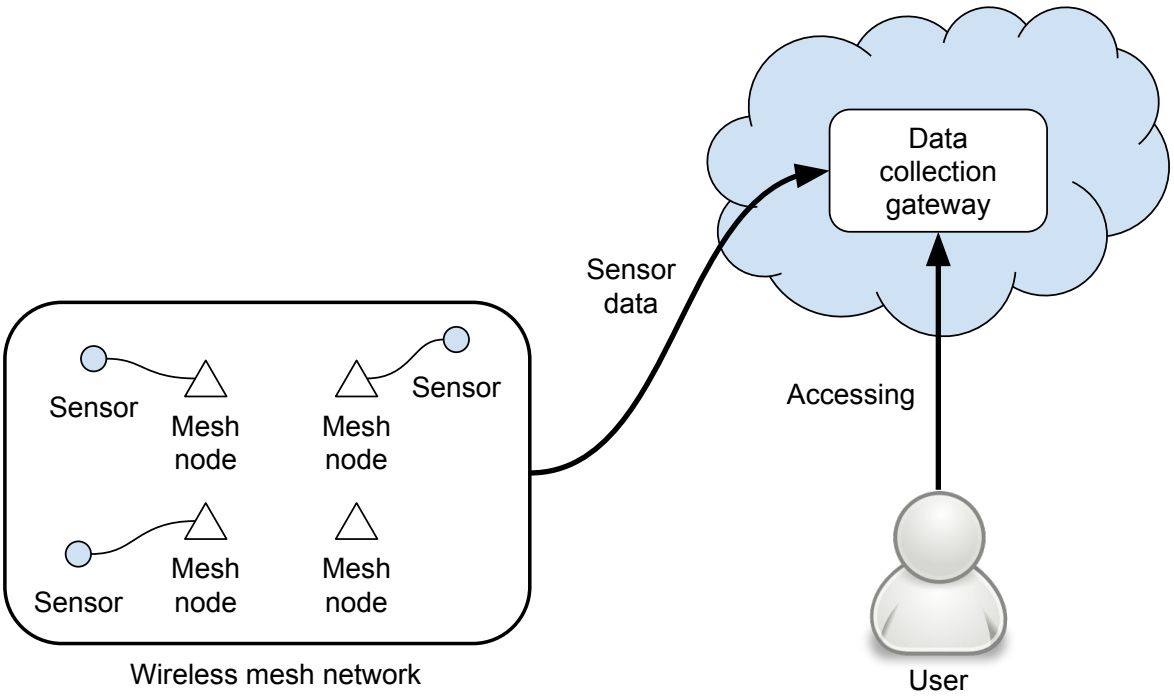


Figure 6.1: Architecture of the proposed data acquisition system.

6.2.1.2 Wireless mesh network

In principle, there are two types of communication networks, either wired or wireless. To distinguish the wired communication network option from the wireless one, it is important to take into account such limitations:

1. The cost of building a vast and complex wired infrastructure is considerably high.
2. The wired infrastructure is not portable.
3. It is technically difficult to place the wired infrastructure in the remote locations.
4. Any harsh meteorological conditions (snow, low temperatures, fog, strong winds, etc.) could cause substantial power problems.

An alternative could be to use one of the wireless communication technologies, e.g., GSM/UMTS, bluetooth, the radio-based or satellite communication. Each wireless communication technology has its advantages and drawbacks but all of them are considerably portable. In case of vast measuring networks, the major limitation of the GSM/UMTS connection is associated with additional charges for every SIM card. Moreover, from an environmental or research perspective there are important remote locations, which have poor or non-existent coverage (e.g., highly elevated regions in Swiss Alps). In contrast to the GSM/UMTS connection, bluetooth technology is relatively cheap. However, its coverage is significantly limited to the maximum range of 100 meters. Within this study the satellite communication is not a feasible option due to the high installation cost. All the aforementioned technologies have substantial drawbacks. Considering the recent progress in the domain of low power wireless devices, the most relevant and cheap option for this study is wireless mesh network (WMN).

Wireless mesh network is an alternative communication scheme for last-mile broadband Internet access that can provide cheap Internet connectivity delivered to end users at the last mile, an easily deployable multi-hop wireless bridge between distant bases in no-direct line of sight scenarios or a wireless back-haul connecting sensors of different purposes such as environmental monitoring or smart-home applications. Moreover, it has such advantages as a good reliability, scalability, and low upfront investments. However, wireless mesh network also has drawbacks such as numerous hardware and software challenges to be properly deployed. One of the challenges is a proper selection of hardware to operate under a specific power consumption regime [22], e.g., in case when a node is solar powered, it has to harvest and store enough energy during the day-light operation to work uninterruptedly at night. With an acceptable signal, there will be a high throughput and satisfactory performance of ciphering and packet forwarding from the node setup. Moreover, other network adapters will be able to accommodate traffic coming from wireless interfaces. Environmental monitoring pilot projects deployed in remote and mountainous regions [89, 170] and backup backbones installed in urban areas illustrate that wireless mesh networks perfectly integrate into the existing AAA (Authentication, Authorization, Accounting) [21, 144], monitoring and cloud infrastructure schemes of Swiss universities. For the purpose of this work, the existing Wi-Fi based backhauls are used to transport information from environmental sensors to Internet storage facilities in real time.

For the purpose of this work, portable stations were deployed in the Emme river catchment. All the deployed stations serve as the WMN nodes, allowing real-time data communication. The IEEE 802.11 specifications were chosen to serve as a basic link component. For the setup of a wireless mesh network, every node is supposed to act as a relay for multi-hop data forwarding. A robust TCP/IP-based WMN was developed to allow multi-hop communication among all wireless stations. There were four major reasons for that:

1. Careful selection of installation procedures;
2. Appropriate choice of hardware components, e.g., wireless cards, directional antennas of high gain, motherboards, batteries, solar panel;
3. Channel allocation schemes;
4. Auto-configuration mechanisms, which include dynamic routing protocols, e.g., Optimized Link State Routing (OLSR) and IEEE 802.11s.[1]

[1]In some cases, we provide redundant connectivity to improve performance of our network, i.e., when a link fails, our dynamic routing mechanisms switch to backup connections.

Moreover, some nodes act as environmental stations, gathering information about system states (i.e., temperature, water level, and pressure) at specific geographical locations through tailored sensors (e.g., thermometers, barometers), attached to the universal serial bus (USB) or serial ports.

6.2.1.3 Hardware platform

From the hardware perspective, every node uses the Alix3d2 motherboards with two on-board mini-pci slots. The Linux-based ADAM system, which was developed by the Communication and Distributed Systems research group of the University of Bern, serves as the operating system. The key advantages of using the Alix boards are low price, small size, limited power consumption, and an i386-compatible processor on-board. Regardless the Alix3d2 motherboards' little computational capacity (500 MHz CPU, 256 MB RAM), it is still capable of running a general purpose operating system such as Linux, which is able to perform TCP/IP mesh networking and environmental monitoring. To install Winstron DNMA-92 IEEE 802.11abgn adapters, the mini-pci bus and up to two antennas (including MIMO ones) were used. When required, a directional antenna can support long-distance connections, while an omni-directional one provides a short-distance communication.

The Alix motherboard requires only 15 W upon a standard operation. Considering such low power requirements, the boards could be solar powered when the electric power grid is not available. To maintain the 24-hour uninterruptible operation, the solar powered stations are equipped with batteries, which are charged during the day-light operation to provide energy at night. Due to the small size, Alix boards can be placed in special purpose enclosures, which protect electronic devices from dangerous environmental factors, e.g., precipitation, humidity. When a station requires many (i.e., ≥ 2) Wi-Fi interfaces, a few motherboards could be accumulated into a single enclosure, only various boxes of different size are required.

Furthermore, from the administration perspective, the wireless mesh network is easily expandable due to the installed OLSR and IEEE 802.11s. The installation of new nodes requires little attention or technical expertise.

6.2.1.4 Environmental monitoring and data collection

In addition to the hardware platform, the data collection software was developed for the purpose of environmental monitoring. Our studies reveal great similarities between environmental and system/network monitoring provided by Zabbix, Nagios, or SNMP aimed at tracking the status of system/network components. The system/network monitoring is based upon a periodical query of the status of a certain unit. The received time-based information (t, value) is transmitted to a central database and stored as the time-based relation for the future use for network administration, management, or planning. The environmental monitoring works the same way as it also requires periodical information about the environmental system state[2] (i.e., periodical measurements through sensors). All the received data shall be stored, for example, in a hydrogeological database. In this case, the hydrogeological modeler accesses the database to get the required information and provides hydrogeological forecasting of the system state in the near future.

We integrated Zabbix [4] with our mesh network. Zabbix fully implements our requirements. First of all, it is a client-server infrastructure for remote monitoring that uses TCP/IP for the client-server communication. As a long lasting project in the open source

[2]"System state" should be understood as in physics, i.e., temperature, pressure, precipitation rate, etc.

community, it serves a large number of users. Moreover, it is extensively tested and well documented software. Considering all these advantages, it serves well to quickly configure all the necessary operations required by hydrogeomonitoring.

From the implementation perspective, a Zabbix agent is installed on every node in the network. It responds to monitoring queries initiated by the Zabbix server, which in our case resides on the Data Collection Gateway. Normally, the Zabbix server reads out predefined parameters (e.g., traffic on interfaces). It also supports user-defined commands to monitor user-specific hardware components. To support every currently deployed sensor, drivers as remote commands are implemented. Every node in the network possesses drivers to all deployed sensors. When a sensor is physically installed on a given node, the Zabbix server is configured to collect environmental information from the specific resource, e.g., node, bus (i.e., USB, serial port), through a specific driver (remote command). To control the Zabbix server, there are an advanced back-end web interface and a rich logging system. This allows us to periodically check environmental sensors and transport readings to the gateway. For future use, the data are stored in the environmental database.

6.2.2 Data assimilation with EnKF-HGS

Data assimilation within the cloud-based modelling system is done via EnKF-HGS. Overall, the cloud-based data assimilation platform consists of several components: (i) a user interface; (ii) the EnKF-HGS hydrological simulator; (iii) an execution environment. The first component provides a user with an interface to orchestrate the cloud-based simulation platform and operate on the input/output data. The second component is represented by the highly sophisticated hydrological simulator EnKF-HGS described in Section 3.4. While the third component is defined by one of the aforementined cloudification methodologies. The methodology should be carefully selected based on the main objective the user aims to accomplish and the use case.

Cloudification methodology selection

First, we underline the main objectives of the aforementioned methodologies. When the researcher aims to improve the scalability in parameter-based scientific simulations, then the big data inspired cloudification methodology described in Section 4.3 is definitely recommended. Because its key goal is to achieve virtually unlimited scalability of CPU- and memory-intensive scientific simulators and gain benefits provided by the big data paradigm, i.e., performance improvement by imposing data-locality through a MapReduce framework. In case the researcher intends to experiment and find the most suitable solution for the specific scientific application that will allow to optimize the utilization of resources, then we recommend the service-oriented cloudification methodology described in Section 5.3. Because it is directed at optimizing CPU resource utilization, while balancing between the application execution time and operational expenses.

In Table 6.1, we summarize key characteristics of the methodologies under a certain number of parameters. These characteristics could serve as benchmarks for the evaluation of a specific use case.

In general, the methodologies are mutually exclusive. Considering the research goal and use case, the user is expected to choose one over the other.

	Big data inspired cloudification methodology	Service-oriented cloudification methodology
Fault-tolerance	Big data platforms are fault-tolerant by design. This cloudification methodology is suitable in case a researcher prefers to use a built-in fault-tolerance mechanism.	In case a researcher aims to experiment and find the most suitable fault-tolerance technique for the specific scientific application, then the service-oriented methodology is more appropriate.
Resource management	Big data inspired methodology relies on Apache Spark as an execution engine with its underlying resource manager. The user has limited control over job and task scheduling, which are fully managed by the resource manager.	In this methodology, job and task management policies could be customized by the user. While the resource management can be either automatic or manual.
Modularity	The architecture of the cloudified application is restricted to the master-slave execution pattern.	With service-oriented methodology, the application has no predefined execution pattern. The user is expected to implement one, e.g., client-server-worker.
Development time	This methodology is based on a successful industry solution, i.e., Apache Spark. Cloudifying a scientific application with Spark will definitely require reduced development time, without worrying about other users or software compatibility issues.	Because this methodology relies on SOA principles, it facilitates the development by leveraging existing assets, e.g., already developed components, services.
Integration with external heterogeneous services	For the reason that work distribution and task management are automatic in the Spark-based implementation, intermediate invocations of external services might be inefficient.	In view of the fact that the user has control over work distribution and task management policies, it can be relatively easy to execute invocations of external services.
Versatility	This methodology is MapReduce framework dependent. That might potentially restrict the selection of appropriate programming languages and available libraries.	This methodology is platform-independent. It only specifies the required communication channels, which link language-independent components.

Table 6.1: Summary of the key characteristics of two cloudification methodologies.

6.3 Summary

In this chapter, we presented the conceptual framework for cloud-based hydrological modeling and data assimilation. This framework helps to solve a number of challenges and

difficulties associated with modeling, designing, simulating and/or building complex dynamic hydrological systems. The framework offers two key functionalities, i.e., real-time data acquisition and cloud-based environmental modeling using the data assimilation approach. Both functionalities of the conceptual framework have been successfully tested and implemented through a wireless mesh network infrastructure and the cloudified EnKF-HGS simulator. Figure 6.2 illustrates the overall architecture of the proposed conceptual framework, all its components, their current and potential interconnections as well as the possibility to improve the framework applicability with a feedback mechanism.

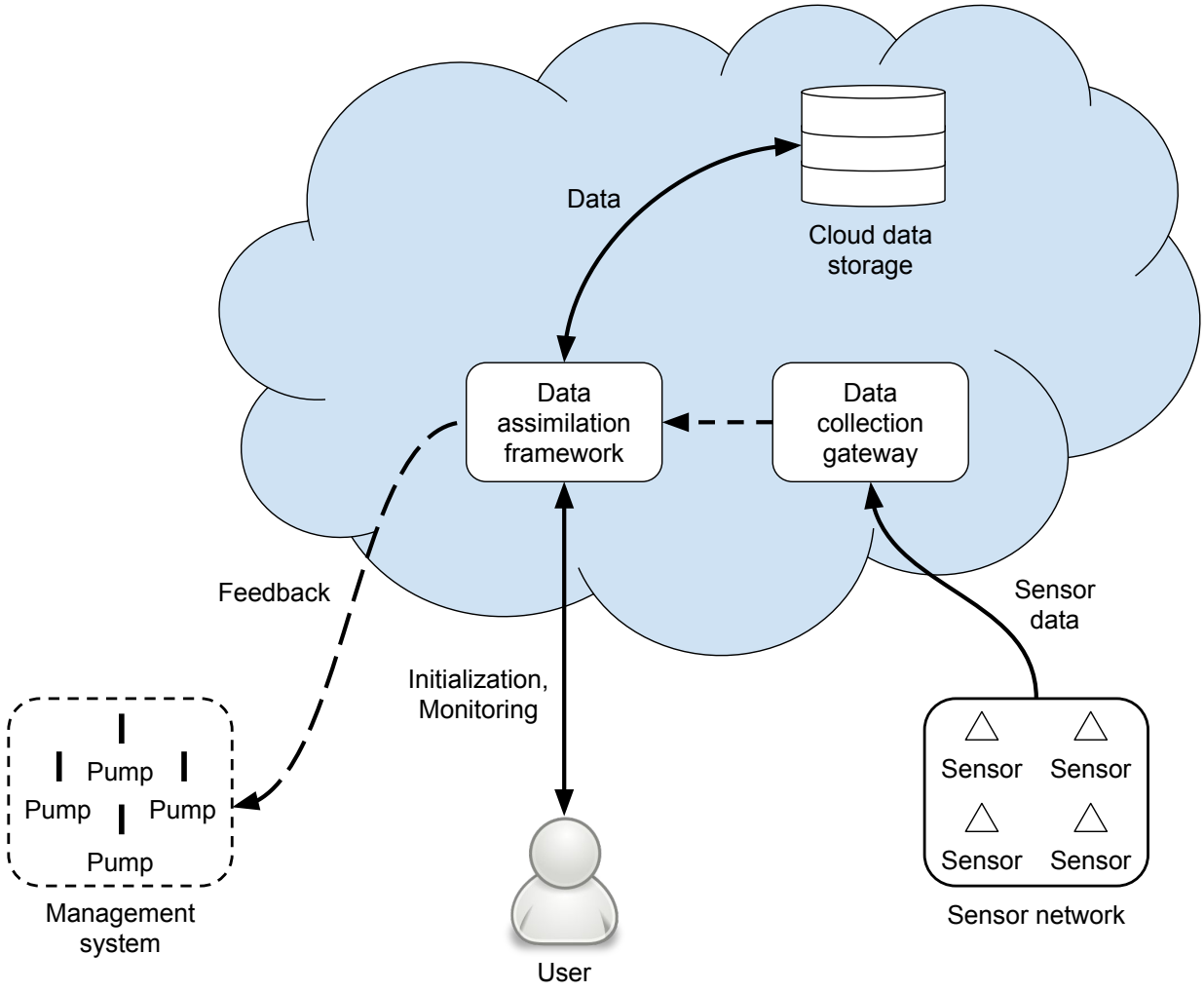


Figure 6.2: Architecture of the proposed conceptual framework. Dashed lines indicate currently not implemented connections.

First, we designed and deployed a sensing mesh network in the valley of the river Emme. It continuously provided us with field observations during one-year period. All the data were uninterruptedly transmitted through the Zabbix infrastructure to the data collection gateway. Second, we have successfully cloudified the data assimilation platform under two different methodologies presented in Sections 4.4 and 5.4. Considering the research goal and specificities of the use case, the researcher has a possibility to choose a cloud execution environment, which suits best. Additionally, assessment and control of the data assimilation platform can be achieved through the user interface, which is easy-to-use for researchers with different computer skills. Next, we envision to implement a stable connection between the data

collection gateway and the data assimilation platform, so that the real-time measurement data could be automatically provided for the EnKF-HGS simulations. When integrated altogether, the framework will provide a researcher with considerable advantages in dynamic regulation of water supply management systems. To improve the framework applicability to hydrological systems, we propose to implement a feedback mechanism, which will provide pumps with optimal pumping rates successfully generated based on the EnKF-HGS simulations.

Chapter 7

Conclusion

7.1 Summary of contributions

In this thesis, we identified the problem of migrating cluster-based scientific simulators to clouds, when these applications have such specific characteristics:

1. They implement the iterative Monte Carlo method.
2. They continuously require higher computing power.
3. They were built without incorporating new technologies – i.e., clouds, big data, service-oriented technologies.
4. Their adaptation to new technologies is substantially impeded by dependencies on external proprietary services.

To tackle this problem, we elaborated two cloudification solutions, tested them by using the hydrological simulator, and integrated our solutions into one conceptual framework.

Contribution 1 - enhancing the "big data inspired cloudification methodology"

The original cloudification methodology proposed by Caíno-Lores et al. [33] showed outstanding results in migrating scientific simulators to big data infrastructures. Hence, we proposed to enhance it by extending it to a wider range of applications, i.e., scientific simulators with the aforementioned characteristics that cannot be defined as suitable following the initial cloudification procedure.

Contribution 2 - service-oriented cloudification methodology and computing-oriented cloud services

Considering the dependency of the EnKF-HGS simulator on two proprietary simulation kernels, we proposed to build a cloudification methodology, which does not require substantial modifications of the original code of the scientific simulator. To facilitate the migration of the EnKF-HGS simulator to clouds, we developed a platform for building cloud-based computing-oriented services (CLAUDEs). The platform allows the researcher to quickly encapsulate parts of simulation logic into self-contained services and concentrate only on the required aspects of the cloud-based execution, e.g., job scheduling, CPU resource utilization.

Contribution 3 - practical application of the cloudification methodologies and a conceptual framework for hydrological data acquisition and management

We applied the aforementioned cloudification solutions to the EnKF-HGS simulator. Based on the research objective and a specific use case, the researcher is expected to choose one methodology over the other. Then we developed the conceptual framework consisting of two functional parts – i.e., the real-time data acquisition infrastructure and the cloudified EnKF-HGS simulator.

7.2 Future directions

This thesis opens several interesting research perspectives, which can further enhance the cloudification methodologies presented in Chapters 4 and 5 and the conceptual framework introduced in Chapter 6.

Big data inspired cloudification methodology

From the big data paradigm perspective, we conclude that in order to gain big data benefits, significant modifications to the original scientific simulator code may be required. Hence, we propose to work towards a hybrid approach directed at converging benefits of slim MPI processes with small memory overhead and encapsulated data parallelism of the big data paradigm.

From the infrastructure perspective, we identified the memory limitation of the Spark-based implementation in case it requires significant amount of memory for processing, caching, and exploiting in-memory solutions for enhanced performance. Hence, we propose to test other MapReduce implementations presented in Subsection 4.2.1, analyze the results, and identify the reason behind the heavy memory consumption.

Service-oriented cloudification methodology

The key goal of the service-oriented cloudification methodology is to optimize CPU resource utilization. Because we have not yet evaluated all the potential trade-offs of the cloudification methodology, we propose to build a cost-efficiency model directed at balancing operational expenses and application performance.

The cloudification methodology substantially relies on the platform for building cloud-based computing-oriented services (CLAUDEs). We identify the necessity for considerable stress testing of the CLAUDE-building platform. For the next version of CLAUDE-0, we plan to implement actual data persistence and component fault tolerance mechanisms. Also, the potential benefit of replacing CLAUDE-0 components with other existing technologies and protocols shall be evaluated. Additionally, an alternative data distribution scheme shall be designed in order to reduce the severe data transmission overhead.

Considering the cloudification methodology goal of the CPU resource utilization optimization, we foresee considerable benefits in developing a novel mechanism by using recent technological advances in lightweight containerization technologies – i.e., Docker. This mechanism could substantially improve the distribution of computational tasks and job scheduling by means of the live container migration feature. The mechanism would tentatively work by combining the lightweight containerization currently available in Docker and the methods provided by the traditional VM-based virtualization. In this context, we propose to investigate these additional points more thoroughly.

Conceptual framework for cloud-based hydrological modeling and data assimilation

To improve the framework applicability to hydrological systems, we propose to implement a stable connection between the data collection gateway and the data assimilation platform. Such connection will automatically provide the platform with real-time measurement data required for simulations. Another possible improvement is to implement a feedback mechanism, which will enable the integration of the cloud-based data assimilation platform with existing water supply management systems.

Appendix A

List of publications

1. Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Applying Big Data Paradigms to a Large Scale Scientific Workflow: Lessons Learned and Future Directions", *Future Generation Computer Systems*, submitted in December 2016
2. Kurtz, W.; Lapin, A.; Schilling, O.; Tang, Q.; Schiller, E.; Braun, T.; Hunkeler, D.; Vereecken, H.; Sudicky, E.; Kropf, P.; Hendricks Franssen, H.; Brunner, P.; "Integrating hydrological modelling, data assimilation and cloud computing for real-time management of water resources", *Environmental Modelling & Software*, Volume 93, Pages 418-435, ISSN 1364-8152, http://dx.doi.org/10.1016/j.envsoft.2017.03.011, 2017
3. Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Methodological Approach to Data-Centric Cloudification of Scientific Iterative Workflows", *16th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, Pages 469-482, DOI 10.1007/978-3-319-49583-5_36, Granada, Spain, December, 2016
4. Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Lessons Learned from Applying Big Data Paradigms to a Large Scale Scientific Workflow", *11th Workshop on Workflows in Support of Large-Scale Science (WORKS16)*, Salt Lake City, Utah, USA, November, 2016
5. Caíno-Lores, S.; Lapin, A.; Kropf, P.; Carretero, J., "Cloudification of a Legacy Hydrological Simulator using Apache Spark", *XXVII Jornadas de Paralelismo (JP2016)*, Salamanca, Spain, September, 2016
6. Lapin, A.; Schiller, E.; Kropf, P., "CLAUDE: Cloud-computing for non-interactive long-running computationally intensive scientific applications", *Proceedings of the 8th GI Conference in Autonomous Systems (AutoSys)*, Pages 221-232, ISBN 978-3-18-384210-0, 2015
7. Lapin, A.; Schiller, E.; Kropf, P., "Integrated Cloud-based Computational Services", *Proceedings of the 7th GI Workshop in Autonomous Systems (AutoSys)*, Pages 280-292, ISBN 978-3-18-383510-2, 2014
8. Lapin, A.; Schiller, E.; Kropf, P.; Schilling, O.; Brunner, P.; Kapic, A.J.; Braun, T.; Maffioletti, S., "Real-Time Environmental Monitoring for Cloud-based Hydrogeological Modeling with HydroGeoSphere" *Proceedings of the High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*, Pages 959-965, DOI 10.1109/HPCC.2014.154, August, 2014
9. Schiller, E.; Kropf, P.; Lapin, A.; Brunner, P.; Schilling, O.; Hunkelet, D., "Wireless Mesh Networks and Cloud Computing for Real Time Environmental Simulations",

10th International Conference on Computing and Information Technology (IC2IT2014), Pages 1-11, DOI 10.1007/978-3-319-06538-0_1, 2014

References

[1] Riak CS. URL http://basho.com/. Accessed on September 2, 2017.

[2] Worldwide LHC Computing Grid. URL http://wlcg.web.cern.ch/. Accessed on September 2, 2017.

[3] TOP500 The List, 1993 - 2017. URL https://www.top500.org/. Accessed on September 2, 2017.

[4] Zabbix, The Enterprise-class Monitoring Platform, 2001-2017. URL http://www.zabbix.com. Accessed on September 2, 2017.

[5] IEEE Standard for Information Technology - Portable Operating System Interface (POSIX(R)). *IEEE Std 1003.1, 2004 Edition The Open Group Technical Standard. Base Specifications, Issue 6. Includes IEEE Std 1003.1-2001, IEEE Std 1003.1-2001/Cor 1-2002 and IEEE Std 1003.1-2001/Cor 2-2004. Shell*, pages 1–3874, Dec 2008. doi: 10.1109/IEEESTD.2008.7394902.

[6] SLURM Workload Manager. Overview, March 2013. URL https://slurm.schedmd.com/overview.html. Accessed on September 2, 2017.

[7] Apache Hadoop, 2014. URL http://hadoop.apache.org/. Accessed on September 2, 2017.

[8] A brief history of cloud computing, March 2014. URL https://www.ibm.com/blogs/cloud-computing/2014/03/a-brief-history-of-cloud-computing-3/. Accessed on September 2, 2017.

[9] SLURM Workload Manager. Customer Testimonials, April 2015. URL https://slurm.schedmd.com/testimonials.html. Accessed on September 2, 2017.

[10] IDC's Latest Forecast for the HPC Market: "2016 is Looking Good", November 2016. URL https://www.top500.org/news/idcs-latest-forecast-for-the-hpc-market-2016-is-looking-good/. Accessed on September 2, 2017.

[11] Amazon EC2 Instance Types, 2017. URL https://aws.amazon.com/ec2/instance-types/. Accessed on September 2, 2017.

[12] GlusterFS, 2017. URL https://www.gluster.org/. Accessed on September 2, 2017.

[13] Computing with HTCondor, August 2017. URL https://research.cs.wisc.edu/htcondor/. Accessed on September 2, 2017.

[14] Daniel J Abadi. Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32(1):3–12, 2009.

[15] Wesal Al Belushi and Youcef Baghdadi. An approach to wrap legacy applications into web services. In *Service Systems and Service Management, 2007 International Conference on*, pages 1–6. IEEE, 2007.

[16] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi, and Jameela Al-Jaroodi. A survey of load balancing in cloud computing: Challenges and algorithms. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 137–142. IEEE, 2012.

[17] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Joaquim Romaguera i Ramió, Max Jacobson, and Ingrid Fiksdahl-King. *A pattern language*. Gustavo Gili, 1977.

[18] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, pages 423–424. IEEE, 2004.

[19] Dustin Amrhein and Scott Quint. Cloud computing for the enterprise: Part 1: Capturing the cloud. *DeveloperWorks, IBM*, 8, 2009.

[20] Rajagopal Ananthanarayanan, Karan Gupta, Prashant Pandey, Himabindu Pucha, Prasenjit Sarkar, Mansi Shah, and Renu Tewari. Cloud analytics: Do we really need to reinvent the storage stack? In *HotCloud*, 2009.

[21] M. Anwander, T. Braun, A. Jamakovic, and T. Staub. Authentication and authorisation mechanisms in support of secure access to wmn resources. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a*, pages 1–6, June 2012. doi: 10.1109/WoWMoM.2012.6263776.

[22] G.H. Badawy, A.A. Sayegh, and T.D. Todd. Solar powered wlan mesh network provisioning for temporary deployments. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pages 2271–2276, March 2008. doi: 10.1109/WCNC.2008.401.

[23] Mark Baker and Rajkumar Buyya. Cluster computing at a glance. *High Performance Cluster Computing*, 1:3–47, 1999.

[24] Kapil Bakshi. Cisco cloud computing-data center strategy, architecture, and solutions. *CISCO White Paper. Retrieved October*, 13:2010, 2009.

[25] G. Bauser, Harrie-Jan Hendricks Franssen, Stauffer Fritz, Hans-Peter Kaiser, U. Kuhlmann, and W. Kinzelbach. A comparison study of two different control criteria for the real-time management of urban groundwater works. *Journal of Environmental Management*, 105:21 – 29, 2012. ISSN 0301-4797. doi: http://dx.doi.org/10.1016/j.jenvman.2011.12.024. URL `http://www.sciencedirect.com/science/article/pii/S0301479711004774`.

[26] G Bruce Berriman, Ewa Deelman, John C Good, Joseph C Jacob, Daniel S Katz, Carl Kesselman, Anastasia C Laity, Thomas A Prince, Gurmeet Singh, and Mei-Hu Su. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *SPIE Astronomical Telescopes+ Instrumentation*, pages 221–232. International Society for Optics and Photonics, 2004.

[27] G. Bruce Berriman, Ewa Deelman, Gideon Juve, Mats Rynge, and Jens-S. Vöckler. The application of cloud computing to scientific workflows: a study of cost and performance. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 371(1983), 2012. ISSN 1364-503X. doi: 10.1098/rsta.2012.0066.

[28] Georgiy V Bobashev, D Michael Goedecke, Feng Yu, and Joshua M Epstein. A hybrid epidemic model: combining the advantages of agent-based and equation-based approaches. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, pages 1532–1537. IEEE Press, 2007.

[29] Philip Brunner and Craig T. Simmons. Hydrogeosphere: A fully integrated, physically based hydrological model. *Ground Water*, 50(2):170–176, 2012. ISSN 1745-6584. doi: 10.1111/j.1745-6584.2011.00882.x. URL `http://dx.doi.org/10.1111/j.1745-6584.2011.00882.x`.

[30] Gerrit Burgers, Peter-Jan van Leeuwen, and Geir Evensen. Analysis scheme in the ensemble Kalman filter. *Monthly Weather Review*, 126(6):1719–1724, 1998.

[31] Marc Bux, Jörgen Brandt, Carsten Lipka, Kamal Hakimzadeh, Jim Dowling, and Ulf Leser. Saasfee: Scalable scientific workflow execution engine. *Proc. VLDB Endow.*, 8 (12):1892–1895, August 2015. ISSN 2150-8097. doi: 10.14778/2824032.2824094.

[32] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6): 599–616, 2009.

[33] Silvina Caíno-Lores, Alberto Garcá Fernández, Félix García-Carballeira, and Jesús Carretero. A cloudification methodology for multidimensional analysis: Implementation and application to a railway power simulator. *Simulation Modelling Practice and Theory*, 55:46–62, 2015. ISSN 1569-190X. doi: http://dx.doi.org/10.1016/j.simpat.2015.04.002.

[34] Scott Callaghan, Philip Maechling, Patrick Small, Kevin Milner, Gideon Juve, Thomas H Jordan, Ewa Deelman, Gaurang Mehta, Karan Vahi, Dan Gunter, Keith Beattie, and Christopher Brooks. Metrics for heterogeneous scientific workflows: A case study of an earthquake science application. *Int. J. High Perform. Comput. Appl.*, 25(3):274–285, August 2011. ISSN 1094-3420. doi: 10.1177/1094342011414743.

[35] Jesus Carretero, Silvina Caino, Felix Garcia-Carballeira, and Alberto Garcia. A multi-objective simulator for optimal power dimensioning on electric railways using cloud computing. In *Proceedings of the 5th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, SIMULTECH 2015, pages 428–438, Portugal, 2015. SCITEPRESS - Science and Technology Publications, Lda. ISBN 978-989-758-120-5. doi: 10.5220/0005573404280438. URL `http://dx.doi.org/10.5220/0005573404280438`.

[36] D Ceperley, GV Chester, and MH Kalos. Monte carlo simulation of a many-fermion study. *Physical Review B*, 16(7):3081, 1977.

[37] Kishore Channabasavaiah, Kerrie Holley, and Edward Tuggle. Migrating to a service-oriented architecture. *IBM DeveloperWorks*, 16:727–728, 2003.

[38] Yanpei Chen, Vern Paxson, and Randy H Katz. WhatâĂŹs new about cloud computing security. *University of California, Berkeley Report No. UCB/EECS-2010-5 January*, 20(2010):2010–5, 2010.

[39] Gen-Tao Chiang, Martin T Dove, C Isabella Bovolo, and John Ewen. Implementing a grid/cloud escience infrastructure for hydrological sciences. In *Guide to e-Science*, pages 3–28. Springer, 2011.

[40] Paolo Costa, Matteo Migliavacca, Peter Pietzuch, and Alexander L Wolf. Naas: Network-as-a-service in the cloud. In *Hot-ICE*, 2012.

[41] Carlo Curino, Evan PC Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational cloud: A database-as-a-service for the cloud. 2011.

[42] Leonardo Dagum and Rameshm Enon. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[43] D. de Oliveira, E. Ogasawara, F. BaiÃčo, and M. Mattoso. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 378–385, July 2010. doi: 10.1109/CLOUD.2010.64.

[44] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1251254.1251264`.

[45] Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, and Lavanya Ramakrishnan. Riding the elephant: Managing ensembles with hadoop. In *Proceedings of the 2011 ACM International Workshop on Many Task Computing on Grids and Supercomputers*, MTAGS '11, pages 49–58, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1145-8. doi: 10.1145/2132876.2132888.

[46] Elif Dede, Zacharia Fadika, Jessica Hartog, Madhusudhan Govindaraju, Lavanya Ramakrishnan, Dan Gunter, and R Canon. Marissa: Mapreduce implementation for streaming science applications. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–8. IEEE, 2012.

[47] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[48] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2835-9.

[49] Peter J Denning and Ted G Lewis. Exponential laws of computing growth. *Communications of the ACM*, 60(1):54–65, 2016.

[50] Parviz Deyhim. Best practices for amazon emr. *Technical report*, 2013.

[51] Javier Diaz, Camelia Munoz-Caro, and Alfonso Nino. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on parallel and distributed systems*, 23(8):1369–1386, 2012.

[52] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 27–33. Ieee, 2010.

[53] Jack J Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, 15(9): 803–820, 2003.

[54] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM international symposium on high performance distributed computing*, pages 810–818. ACM, 2010.

[55] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, and Tony Newling. *Patterns: service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization, 2004.

[56] R. Etemadpour, M. Bomhoff, E. Lyons, P. Murray, and A. Forbes. Designing and evaluating scientific workflows for big data interactions. In *Big Data Visual Analytics (BDVA), 2015*, pages 1–8, Sept 2015. doi: 10.1109/BDVA.2015.7314290.

[57] Constantinos Evangelinos and C Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazonâĂŹs ec2. *ratio*, 2(2.40):2–34, 2008.

[58] Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99(C5):10143–10162, 1994. ISSN 2156-2202.

[59] Geir Evensen. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 53(4):343–367, 2003.

[60] Zacharia Fadika, Elif Dede, Madhusudhan Govindaraju, and Lavanya Ramakrishnan. Benchmarking mapreduce implementations for application usage scenarios. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, pages 90–97. IEEE Computer Society, 2011.

[61] Zacharia Fadika, Elif Dede, Madhusudhan Govindaraju, and Lavanya Ramakrishnan. Mariane: Mapreduce implementation adapted for hpc environments. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, pages 82–89. IEEE Computer Society, 2011.

[62] Kaniz Fatema, Vincent C Emeakaroha, Philip D Healy, John P Morrison, and Theo Lynn. A survey of cloud monitoring tools: Taxonomy, capabilities and objectives. *Journal of Parallel and Distributed Computing*, 74(10):2918–2933, 2014.

[63] M. Feldman. IDCâĂŹs Latest Forecast for the HPC Market: "2016 is looking good", November 2016. URL `https://www.top500.org/news/idcs-latest-forecast-for-the-hpc-market-2016-is-looking-good/`. Accessed on September 1, 2017.

[64] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*, 100(9):948–960, 1972.

[65] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-475-8.

[66] Ian Foster and Carl Kesselman. What is the grid. *A three point checklist*, 20, 2003.

[67] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[68] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009, 2009.

[69] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[70] Dimitrios Georgakopoulos, Michael Papazoglou, et al. *Service-oriented computing*. Number Sirsi) i9780262072960. 2009.

[71] Chaima Ghribi, Makhlouf Hadji, and Djamal Zeghlache. Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 671–678. IEEE, 2013.

[72] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the challenges of scientific workflows. *Computer*, 40(12), 2007.

[73] Ionel Gog, Jana Giceva, Malte Schwarzkopf, Kapil Vaswani, Dimitrios Vytiniotis, Ganesan Ramalingan, Manuel Costa, Derek Murray, Steven Hand, and Michael Isard. Broom: Sweeping out garbage collection from big data systems. *Young*, 4:8, 2015.

[74] Nelson Goodman. *Languages of art: An approach to a theory of symbols*. Hackett publishing, 1968.

[75] Katharina Görlach, Mirko Sonntag, Dimka Karastoyanova, Frank Leymann, and Michael Reiter. Conventional workflow technology for scientific simulation. In *Guide to e-Science*, pages 323–352. Springer, 2011.

[76] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. Mapreduce in the clouds for science. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 565–572. IEEE, 2010.

[77] Song Guo, Fan Bai, and Xiaolin Hu. Simulation software as a service and service-oriented simulation experiment. In *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*, pages 113–116. IEEE, 2011.

[78] A. Gupta and D. Milojicic. Evaluation of hpc applications on cloud. In *2011 Sixth Open Cirrus Summit*, pages 22–26, Oct 2011. doi: 10.1109/OCS.2011.10.

[79] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640–645, Dec 2008.

[80] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. Nist cloud computing standards roadmap. *NIST Special Publication*, 35, 2011.

[81] Mark F Horstemeyer. Multiscale modeling: a review. In *Practical aspects of computational chemistry*, pages 87–135. Springer, 2009.

[82] Michael N Huhns and Munindar P Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet computing*, 9(1):75–81, 2005.

[83] Cisco Global Cloud Index. Forecast and methodology, 2015-2020 white paper, 2016.

[84] Alexandru Iosup and Dick Epema. Grid computing workloads. *IEEE Internet Computing*, 15(2):19–26, 2011.

[85] CSA ISACA. Cloud computing market maturity study results. *ISACA, the Cloud Security Alliance*, 2012.

[86] ISO/IEC 17788. Information technology – cloud computing – overview and vocabulary. International standard, International Organization for Standardization, Geneva, Switzerland, 2014.

[87] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168. IEEE, 2010.

[88] Anubhav Jain, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, Dan Gunter, David Skinner, Gerbrand Ceder, and Kristin A. Persson. Commentary: The materials project: A materials genome approach to accelerating materials innovation. *APL Mater.*, 1(1):011002, 2013. doi: http://dx.doi.org/10.1063/1.4812323.

[89] A. Jamakovic, D.C. Dimitrova, M. Anwander, T. Macicas, T. Braun, J. Schwanbeck, T. Staub, and B. Nyffenegger. Real-world energy measurements of a wireless mesh network. In Jean-Marc Pierson, Georges Da Costa, and Lars Dittmann, editors, *Energy Efficiency in Large Scale Distributed Systems*, Lecture Notes in Computer Science, pages 218–232. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40516-7. doi: 10.1007/978-3-642-40517-4_18. URL `http://dx.doi.org/10.1007/978-3-642-40517-4_18`.

[90] Dean Jeffrey and Ghemawat Sanjay. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[91] G. Johnson. The World: In Silica Fertilization; All Science Is Computer Science, March 2001. URL http://www.nytimes.com/2001/03/25/weekinreview/the-world-in-silica-fertilization-all-science-is-computer-science.html. Accessed on September 3, 2017.

[92] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. Scientific workflow applications on amazon ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59–66. IEEE, 2009.

[93] Mikko Ilmari Jyrkama. *A methodology for estimating groundwater recharge*, volume 65. 2004.

[94] Sol Ji Kang, Sang Yeon Lee, and Keon Myung Lee. Performance comparison of openmp, mpi, and mapreduce in practical problems. *Advances in Multimedia*, 2015:7, 2015.

[95] James M Kaplan, William Forrest, and Noah Kindler. Revolutionizing data center energy efficiency. Technical report, Technical report, McKinsey & Company, 2008.

[96] Evelyn Fox Keller. *Models, simulation, and âĂIJcomputer experimentsâĂİ*. na, 2003.

[97] Ken Kennedy, Charles Koelbel, and Hans Zima. The rise and fall of high performance fortran. *Communications of the ACM*, 54(11):74–82, 2011.

[98] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *IEEE Computer*, 36:41–50, 2003.

[99] Jik-Soo Kim, Sangwan Kim, Seokkyoo Kim, Seoyoung Kim, Seungwoo Rho, Ok-Hwan Byeon, and Soonwook Hwang. Towards a next generation distributed middleware system for many-task computing. *International Journal of Software Engineering and Its Applications*, 7(4):379–388, 2013.

[100] Leonard Kleinrock. An internet vision: the invisible global infrastructure. *Ad Hoc Networks*, 1(1):3–11, 2003.

[101] Wolfgang Kurtz, Harrie-Jan Hendricks Franssen, Hans-Peter Kaiser, and Harry Vereecken. Joint assimilation of piezometric heads and groundwater temperatures for improved modeling of river-aquifer interactions. *Water Resources Research*, 50 (2):1665–1688, 2014. ISSN 1944-7973. doi: 10.1002/2013WR014823. URL http://dx.doi.org/10.1002/2013WR014823.

[102] Manny M Lehman. Laws of software evolution revisited. In *European Workshop on Software Process Technology*, pages 108–124. Springer, 1996.

[103] Zhenqin Li and Harold A Scheraga. Monte carlo-minimization approach to the multiple-minima problem in protein folding. *Proceedings of the National Academy of Sciences*, 84(19):6611–6615, 1987.

[104] Harold C Lim, Shivnath Babu, Jeffrey S Chase, and Sujay S Parekh. Automated control in cloud computing: challenges and opportunities. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 13–18. ACM, 2009.

[105] Cui Lin, Shiyong Lu, Zhaoqiang Lai, Artem Chebotko, Xubo Fei, Jing Hua, and Farshad Fotouhi. Service-oriented architecture for view: a visual scientific workflow management system. In *Services Computing, 2008. SCC'08. IEEE International Conference on*, volume 1, pages 335–342. IEEE, 2008.

[106] G. Lin, B. Han, J. Yin, and I. Gorton. Exploring cloud computing for large-scale scientific applications. In *2013 IEEE Ninth World Congress on Services*, pages 37–43, June 2013. doi: 10.1109/SERVICES.2013.13.

[107] Huan Liu and Dan Orban. Cloud mapreduce: A mapreduce implementation on top of a cloud operating system. In *Proceedings of the 2011 11th IEEE/ACM international symposium on cluster, cloud and grid computing*, pages 464–474. IEEE Computer Society, 2011.

[108] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015. ISSN 1572-9184. doi: 10.1007/s10723-015-9329-8.

[109] Andrew C Lorenc. The potential of the ensemble kalman filter for nwpâĂŤa comparison with 4d-var. *Quarterly Journal of the Royal Meteorological Society*, 129(595):3183–3203, 2003.

[110] Sifei Lu, Reuben Mingguang Li, William Chandra Tjhi, Kee Khoon Lee, Long Wang, Xiarong Li, and Di Ma. A framework for cloud-based large-scale data analytics and visualization: Case study on multiscale climate data. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 618–622. IEEE, 2011.

[111] Zaigham Mahmood. Service oriented architecture: potential benefits and challenges. In *Proceedings of the 11th WSEAS International Conference on COMPUTERS*, pages 497–501. World Scientific and Engineering Academy and Society (WSEAS), 2007.

[112] Zaigham Mahmood. Cloud computing: Characteristics and deployment approaches. In *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pages 121–126. IEEE, 2011.

[113] Carlos Maltzahn, Esteban Molina-Estolano, Amandeep Khurana, Alex J Nelson, Scott A Brandt, and Sage Weil. Ceph as a scalable alternative to the hadoop distributed file system. *login: The USENIX Magazine*, 35:38–49, 2010.

[114] Reed M. Maxwell, Mario Putti, Steven Meyerhoff, Jens-Olaf Delfs, Ian M. Ferguson, Valeriy Ivanov, Jongho Kim, Olaf Kolditz, Stefan J. Kollet, Mukesh Kumar, Sonya Lopez, Jie Niu, Claudio Paniconi, Young-Jin Park, Mantha S. Phanikumar, Chaopeng Shen, Edward A. Sudicky, and Mauro Sulis. Surface-subsurface model intercomparison: A first set of benchmark results to diagnose integrated hydrology and feedbacks. *Water Resources Research*, 50(2):1531–1549, 2014. ISSN 1944-7973. doi: 10.1002/2013WR013725.

[115] A Melnikov and Zeilenga K. Simple authentication and security layer (sasl). RFC 4422, RFC Editor, June 2006. URL `https://tools.ietf.org/html/rfc4422`.

[116] J Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35(4):435–438, 1987.

[117] Girija J Narlikar and Guy E Blelloch. Pthreads for dynamic and irregular parallelism. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–16. IEEE Computer Society, 1998.

[118] Hans E Ngodock, Gregg A Jacobs, and Mingshi Chen. The representer method, the ensemble kalman filter and the ensemble kalman smoother: A comparison study using a nonlinear reduced gravity ocean model. *Ocean Modelling*, 12(3):378–400, 2006.

[119] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Autonomic virtual resource management for service hosting platforms. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 1–8. IEEE Computer Society, 2009.

[120] Frank Nielsen. *Introduction to HPC with MPI for Data Science*. Springer, 2016.

[121] Shuangcheng Niu, Jidong Zhai, Xiaosong Ma, Xiongchao Tang, and Wenguang Chen. Cost-effective cloud hpc resource provisioning by building semi-elastic virtual clusters. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–12. IEEE, 2013.

[122] V. Nuthula and N. R. Challa. Cloudifying apps - a study of design and architectural considerations for developing cloudenabled applications with case study. In *Cloud Computing in Emerging Markets (CCEM), 2014 IEEE International Conference on*, pages 1–7, Oct 2014. doi: 10.1109/CCEM.2014.7015487.

[123] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An overview of the scala programming language. Technical report, 2004.

[124] Aisling OâĂŹDriscoll, Jurate Daugelaite, and Roy D Sleator. âĂŸbig dataâĂŹ, hadoop and cloud computing in genomics. *Journal of biomedical informatics*, 46(5):774–781, 2013.

[125] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[126] ARB OpenMP. Openmp application program interface version 4.0, 2013.

[127] Sixto Ortiz. The problem with cloud-computing standardization. *Computer*, 44(7): 13–16, 2011.

[128] Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40 (11), 2007.

[129] D. Partington, P. Brunner, S. Frei, C. T. Simmons, A. D. Werner, R. Therrien, H. R. Maier, G. C. Dandy, and J. H. Fleckenstein. Interpreting streamflow generation mechanisms from integrated surface-subsurface flow models of a riparian wetland and catchment. *Water Resources Research*, 49(9):5501–5519, 2013. ISSN 1944-7973. doi: 10.1002/wrcr.20405.

[130] H Van Dyke Parunak, Robert Savit, and Rick L Riolo. Agent-based modeling vs. equation-based modeling: A case study and usersâĂŹ guide. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 10–25. Springer, 1998.

[131] D Pellerin, D Ballantyne, and A Boeglin. An introduction to high performance computing on aws. *Amazon Whitepaper*, 2015.

[132] Diego Perez, Philipp Rohlfshagen, and Simon M Lucas. Monte-carlo tree search for the physical travelling salesman problem. In *European Conference on the Applications of Evolutionary Computation*, pages 255–264. Springer, 2012.

[133] Arthur C Petersen. *Simulating nature: a philosophical study of computer-simulation uncertainties and their role in climate science and policy advice*. CRC Press, 2012.

[134] Steven J Plimpton and Karen D Devine. Mapreduce in mpi for large-scale graph algorithms. *Parallel Computing*, 37(9):610–632, 2011.

[135] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. pages 412–421, 1974.

[136] Branko Radojević and Mario Žagar. Analysis of issues with load balancing algorithms in hosted (cloud) environments. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 416–420. IEEE, 2011.

[137] M Abdul Rahman. Scalable scientific workflows management system swfms. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 7(11):291–296, 2016.

[138] I. Raicu, I.T. Foster, and Yong Zhao. Many-task computing for grids and supercomputers. In *Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on*, pages 1–11, Nov 2008. doi: 10.1109/MTAGS.2008.4777912.

[139] Lavanya Ramakrishnan and Beth Plale. A multi-dimensional classification model for scientific workflow characteristics. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, page 4. ACM, 2010.

[140] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 140–146. IEEE, 1998.

[141] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pages 13–24. Ieee, 2007.

[142] Daniel A Reed and Jack Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

[143] Harvey Richardson. High performance fortran: history, overview and current developments. *Thinking Machines Corporation*, 14:13, 1996.

[144] Eryk Schiller, Alexey Monakhov, and Peter Kropf. Shibboleth based authentication, authorization, accounting and auditing in wireless mesh networks. In *LCN*, pages 918–926, 2011.

[145] O.S. Schilling, J. Doherty, W. Kinzelbach, H. Wang, P.N. Yang, and P. Brunner. Using tree ring data as a proxy for transpiration to reduce predictive uncertainty of a model simulating groundwaterâĂŞsurface waterâĂŞvegetation interactions. *Journal of Hydrology*, 519, Part B:2258 – 2271, 2014. ISSN 0022-1694. doi: http://dx.doi.org/10.1016/j.jhydrol.2014.08.063.

[146] Michael Sheehan. Cloud computing expo: introducing the cloud pyramid. *Cloud Computing Journal*, 2008.

[147] Kwang Mong Sim and Weng Hong Sun. Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 33(5):560–572, 2003.

[148] Satish Narayana Srirama and Jaagup Viil. Migrating scientific workflows to the cloud: Through graph-partitioning, scheduling and peer-to-peer data sharing. In *High Performance Computing and Communications, 2014 IEEE Intl Conf on*, pages 1105–1112. IEEE, 2014.

[149] Erich Strohmaier, Jack J Dongarra, Hans W Meuer, and Horst D Simon. Recent trends in the marketplace of high performance computing. *Parallel Computing*, 31(3):261–273, 2005.

[150] Seung-Jin Sul and Andrey Tovchigrechko. Parallelizing blast and som algorithms with mapreduce-mpi library. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 481–489. IEEE, 2011.

[151] Chris Swinerd and Ken R McNaught. Design classes for hybrid simulations involving agent-based and system dynamics models. *Simulation Modelling Practice and Theory*, 25:118–133, 2012.

[152] R Therrien, R McLaren, E Sudicky, and S Panday. A Three-dimensional Numerical Model Describing Fully-integrated Subsurface and Surface Flow and Solute Transport. Technical report, 2010. URL `http://www.ggl.ulaval.ca/fileadmin/ggl/documents/rtherrien/hydrogeosphere.pdf`.

[153] R. Therrien, R.G. McLaren, E.A. Sudicky, and S.M. Panday. *HydroGeoSphere*. Groundwater Simulations Group, 2013.

[154] Wei-Tek Tsai. Service-oriented system engineering: a new paradigm. In *Service-oriented system engineering, 2005. sose 2005. IEEE International Workshop*, pages 3–6. IEEE, 2005.

[155] Aphrodite Tsalgatidou, George Athanasopoulos, Michael Pantazoglou, Cesare Pautasso, Thomas Heinis, Roy Grønmo, Hjørdis Hoff, Arne-Jørgen Berre, Magne Glittum, and Simela Topouzidou. Developing scientific workflows from heterogeneous services. *ACM Sigmod Record*, 35(2):22–28, 2006.

[156] Andrei Tsaregorodtsev, Vincent Garonne, and Ian Stokes-Rees. Dirac: A scalable lightweight architecture for high throughput computing. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 19–25. IEEE, 2004.

[157] K. Vahi, M. Rynge, G. Juve, R. Mayani, and E. Deelman. Rethinking data management for big data scientific workflows. In *Big Data, 2013 IEEE International Conference on*, pages 27–35, Oct 2013. doi: 10.1109/BigData.2013.6691724.

[158] Wil MP van der Aalst, Alistair P Barros, Arthur HM ter Hofstede, and Bartek Kiepuszewski. Advanced workflow patterns. In *International Conference on Cooperative Information Systems*, pages 18–29. Springer, 2000.

[159] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[160] Aad J Van Der Steen. Overview of recent supercomputers. 2008.

[161] Jinesh Varia. *Cloud Architectures-White Paper*. 2008. URL `https://aws.amazon.com/about-aws/whats-new/2008/07/16/cloud-architectures-white-paper/`. (last access: 15th October 2016).

[162] HT Condor Version. 8.0. 6 manual. *Center for High Throughput Computing, University of Wisconsin-Madison. Accessed*, 18, 2013.

[163] Lihong Wang and Steven L Jacques. Monte carlo modeling of light transport in multi-layered tissues in standard c. *The University of Texas, MD Anderson Cancer Center, Houston*, pages 4–11, 1992.

[164] Lizhe Wang, Jie Tao, Marcel Kunze, Alvaro Canales Castellanos, David Kramer, and Wolfgang Karl. Scientific cloud computing: Early definition and experience. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 825–830. Ieee, 2008.

[165] Shu-Ching Wang, Kuo-Qin Yan, Wen-Pin Liao, and Shun-Sheng Wang. Towards a load balancing in a three-level cloud computing network. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 1, pages 108–113. IEEE, 2010.

[166] Y. Wang, R. Goldstone, W. Yu, and T. Wang. Characterization and optimization of memory-resident mapreduce on hpc systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 799–808, May 2014. doi: 10.1109/IPDPS.2014.87.

[167] E Weinan. *Principles of multiscale modeling*. Cambridge University Press, 2011.

[168] Rafael Weingärtner, Gabriel Beims Bräscher, and Carlos Becker Westphall. A distributed autonomic management framework for cloud computing orchestration. In *Services (SERVICES), 2016 IEEE World Congress on*, pages 9–17. IEEE, 2016.

[169] Stephen A White. Process modeling notations and workflow patterns. *Workflow handbook*, 2004:265–294, 2004.

[170] D. Wu and P. Mohapatra. Qurinet: A wide-area wireless mesh testbed for research and experimental evaluations. In *Second International Conference on Communication Systems and Networks (COMSNETS), 2010*, pages 1–10, Jan 2010. doi: 10.1109/COMSNETS.2010.5431993.

[171] Tin-Yu Wu, Wei-Tsong Lee, Yu-San Lin, Yih-Sin Lin, Hung-Lin Chan, and Jhih-Siang Huang. Dynamic load balancing mechanism based on cloud storage. In *Computing, Communications and Applications Conference (ComComAp), 2012*, pages 102–106. IEEE, 2012.

[172] Chaowei Yang, Michael Goodchild, Qunying Huang, Doug Nebert, Robert Raskin, Yan Xu, Myra Bambacus, and Daniel Fay. Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, 4(4):305–329, 2011.

[173] Katherine Yelick, Susan Coghlan, Brent Draney, Richard Shane Canon, et al. The magellan report on cloud computing for science. *US Department of Energy, Washington DC, USA, Tech. Rep*, 2011.

[174] Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer, 2003.

[175] Richard M Yoo, Anthony Romano, and Christos Kozyrakis. Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 198–207. IEEE, 2009.

[176] Edward Yourdon and Larry L Constantine. *Structured design: Fundamentals of a discipline of computer program and systems design*. Prentice-Hall, Inc., 1979.

[177] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[178] Edward N Zalta et al. Stanford encyclopedia of philosophy, 2003.

[179] Z. Zhang, K. Barbary, F. A. Nothaft, E. Sparks, O. Zahn, M. J. Franklin, D. A. Patterson, and S. Perlmutter. Scientific computing meets big data technology: An astronomy use case. In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 918–927, Oct 2015. doi: 10.1109/BigData.2015.7363840.

[180] Zhuoyao Zhang. Processing data-intensive workflows in the cloud. Technical Report Technical Report No. MS-CIS-12-08, University of Pennsylvania Department of Computer and Information Science, 2012.

[181] Y. Zhao, X. Fei, I. Raicu, and S. Lu. Opportunities and challenges in running scientific workflows on the cloud. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, pages 455–462, Oct 2011. doi: 10.1109/CyberC.2011.80.

[182] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.